Friedrich Schiller University Jena
Faculty of Mathematics and Computer Science

Design and Implementation of an Efficient and Robust Algorithm for Smoothing Curves on Surface Meshes

MASTER'S THESIS

for obtaining the academic degree

Master of Science (M.Sc.) in Mathematics

submitted by Markus Pawellek

born on May 7th, 1995 in Meiningen
Student Number: 144645

First Supervisor:	Kai Lawonn
Second Supervisor:	Noeska Smit
Third Supervisor:	Hauke Bartsch

Jena, March 25, 2023

Abstract

The smoothing of surface curves is an essential tool in mesh processing to allow applications, such as surgical planning, to segment, edit, and cut surfaces. As curves are typically designed by domain experts to mark relevant surface regions, its smoothed counterpart should retain close to the original and easily adjustable by the user. Previous solutions that are based on energy-minimizing splines or generalizations of Bézier splines need a large number of control points to achieve this behavior and may suffer from poor performance or numerical instabilities. This thesis aims for the development and a detailed implementation strategy of an efficient and robust algorithm for smoothing discrete curves on triangular surface meshes. The method is based on the optimization of local geodesic curvatures to obtain results close to the initial curve and achieves real-time performance through parallelization on the CPU and GPU. The implementation uses the C++ programming language combined with the OpenGL graphics API and is fully provided as an open-source code repository. Using appropriate benchmarks and the "Thingi10K" dataset, the efficiency and robustness of the algorithm is evaluated. Finally, the curve smoothing approach is applied in a medical context to the automatic segmentation of lung lobes.

Zusammenfassung

Das Glätten von Oberflächenkurven ist ein wesentliches Werkzeug im Mesh Processing, um es Anwendungen, wie zum Beispiel in der chirurgischen Planung, zu ermöglichen, Oberflächen zu segmentieren, bearbeiten und zerschneiden. Da Kurven in der Regel von Fachleuten konstruiert werden, um relevante Oberflächenbereiche zu markieren, sollte deren geglättete Form nahe am Original und durch den Benutzer leicht anpassbar sein. Bisherige Lösungen, die auf energieminimierenden Splines oder Verallgemeinerungen von Bézier-Splines basieren, benötigen eine große Anzahl von Kontrollpunkten, um dieses Verhalten zu erreichen, und weisen unter Umständen eine schlechte Performanz oder numerische Instabilitäten auf. Diese Arbeit zielt auf die Entwicklung und eine detaillierte Implementierungsstrategie eines effizienten und robusten Algorithmus zur Glättung diskreter Kurven auf Oberflächennetzen ab. Die Methode basiert auf der Optimierung lokaler geodätischer Krümmungen, um Ergebnisse nahe der Ausgangskurve zu erzielen, und erreicht Echtzeitperformanz durch die Parallelisierung auf der CPU und GPU. Die Implementierung verwendet die C++-Programmiersprache kombiniert mit der OpenGL-API und wird vollständig als Open-Source Code-Repository bereitgestellt. Unter Verwendung geeigneter Benchmarks und des »Thingi10K«-Datensatzes wird die Effizienz und Robustheit des Algorithmus ausgewertet. Schließlich wird der Ansatz der Kurvenglättung im medizinischen Kontext auf die automatische Segmentierung von Lungenflügeln angewandt.

Acknowledgments

I am grateful to Kai Lawonn, Noeska Smit, and Hauke Bartsch for their supervising, their helpful suggestions, and for our interesting discussions. Also great thanks to Clemens Anschütz for programming assistance and Ann Sommerfeld, Johanna Vielemeyer, and Kerstin Pawellek for assisting in writing the thesis and proofreading.

Contents

List of Figures

List of Tables

List of Definitions and Theorems

List of Code

List of Abbreviations and Acronyms

Abbreviation	Definition
iid	Independently and Identically Distributed
CDF	Cumulative Distribution Function
SLLN	Strong Law of Large Numbers
LTE	Light Transport Equation
API	Application Programming Interface
RAII	Resource Acquisition is Initialization
SFINAE	Specialization Failure is not an Error
STL	Standard Template Library

Symbol Table

Symbol	Definition
Logic	
$\exists \ldots : \ldots$	There exists . . ., such that . . .
$a := b$	a is defined by b.
Set Theory	
$\{\ldots\}$	Set Definition
$\{\ldots \mid \ldots\}$	Set Definition with Condition
$x \in A$	x is an element of the set A.
$A \subset B$	The set A is a subset of the set B.
$A \cap B$	Intersection — $\{x \mid x \in A$ and $x \in B\}$ for sets A, B
$A \cup B$	Union — $\{x \mid x \in A$ or $x \in B\}$ for sets A, B
$A \setminus B$	Relative Complement — $\{x \in A \mid x \notin B\}$ for sets A, B
$A \times B$	Cartesian Product — $\{(x, y) \mid x \in A, y \in B\}$ for sets A and B
A^n	n-fold Cartesian Product of Set A
$\emptyset$	Empty set — $\{\}$.
$\#A$	Number of Elements in the Set A
$\mathcal{P}(A)$	Power Set of Set A
Special Sets	
$\mathbb{N}$	Set of Natural Numbers
$\mathbb{N}_0$	$\mathbb{N} \cup \{0\}$
$\mathbb{P}$	Set of Prime Numbers
$\mathbb{Z}$	Set of Integers
$\mathbb{Z}_n$	Set of Integers Modulo n
$\mathbb{F}_m$	Finite Field with $m \in \mathbb{P}$ Elements
$\mathbb{F}_m^{p \times q}$	Set of $p \times q$-Matrices over Finite Field $\mathbb{F}_m$
$\mathbb{F}_2$	Finite Field of Bits
$\mathbb{F}_2^n$	Set of n-bit Words
$\mathbb{R}$	Set of Real Numbers
$\mathbb{R}^n$	Set of n-dimensional Real Vectors
$\mathbb{S}^2$	Set of Directions — $\left\{x \in \mathbb{R}^3 \mid \lVert x \rVert = 1\right\}$
Functions	
$f : X \to Y$	f is a function with domain X and range Y.
id_X	Identity Function over the Set X
$f \circ g$	Composition of Functions f and g
f^{-1}	Inverse Image of Function f
f^n	n-fold Composition of Function f
Bit Arithmetic	
$x_{n-1} \ldots x_1 x_0$	n-bit Word x of Set $\mathbb{F}_2^n$
$x \leftarrow a$	Left Shift of all Bits in x by a
$x \rightarrow a$	Right Shift of all Bits in x by a
$x \circlearrowleft a$	Circular Left Shift of all Bits in x by a
$x \oplus y$	Bit-Wise Addition of x and y
$x \odot y$	Bit-Wise Multiplication of x and y
$x \mid y$	Bit-Wise Or of x and y

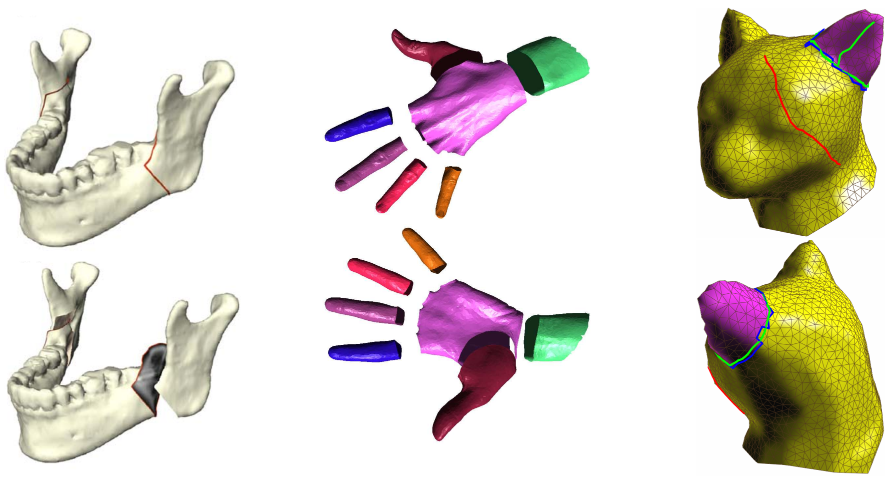
Symbol	Definition
Probability Theory	
$\mathcal{B}(\mathbb{R})$	Borel σ-Algebra over $\mathbb{R}$
$(\Sigma, \mathcal{A})$	Measurable Space over Σ with σ-Algebra $\mathcal{A}$
λ	Lebesgue Measure
$\int_U f \, \mathrm{d}\lambda$	Lebesgue Integral of f over U
$\mathrm{L}^2(U, \lambda)$	Set of Square-Integrable Functions over the Set U with Respect to the Lebesgue Measure λ
$(\Omega, \mathcal{F}, P)$	Probability Space over Ω with σ-Algebra $\mathcal{A}$ and Probability Measure P
$\int_\Omega X \, \mathrm{d}P$	Integral of Random Variable X with respect to Probability Space $(\Omega, \mathcal{A}, P)$
$\int_\Omega X(\omega) \, \mathrm{d}P(\omega)$	$\int_\Omega X \, \mathrm{d}P$
P_X	Distribution of Random Variable X
$\mathbb{E}\, X$	Expectation Value of Random Variable X
$\mathrm{var}\, X$	Variance of Random Variable X
$\sigma(X)$	Standard Deviation of Random Variable X
$\mathbb{1}_A$	Characteristic Function of Set A
δ_ω	Dirac Delta Distribution over $\mathcal{S}^2$ with respect to $\omega \in \mathcal{S}^2$
$\bigotimes_{n \in I} P_n$	Product Measure of Measures P_n Indexed by the Set I
Miscellaneous	
$(x_n)_{n \in I}$	Sequence of Values x_n with Index Set I
$\lvert x \rvert$	Absolute Value of x
$\lVert x \rVert$	Norm of Vector x
$x \mod y$	x Modulo y
$\gcd(\rho, k)$	Greatest Common Divisor of ρ and k
$\max(x, y)$	Maximum of x and y
$\lim_{n \to \infty} x_n$	Limit of Sequence $(x_n)_{n \in \mathbb{N}}$
$\sum_{k=1}^{n} x_k$	Sum over Values x_k for $k \in \mathbb{N}$ with $k \leq n$
$\dim X$	Dimension of X
$\lceil x \rceil$	Ceiling Function
$\langle x \mid y \rangle$	Scalar Product
$[a, b]$	$\{x \in \mathbb{R} \mid a \leq x \leq b\}$
(a, b)	$\{x \in \mathbb{R} \mid a < x < b\}$
$[a, b)$	$\{x \in \mathbb{R} \mid a \leq x < b\}$
Constants	
∞	Infinity
π	$3.1415926535\ldots$ — Pi
Units	
$1\,\mathrm{B}$	$1\,\mathrm{Byte} = 8\,\mathrm{bit}$
$1\,\mathrm{GiB}$	$2^{30}\,\mathrm{B}$
$1\,\mathrm{s}$	$1\,\mathrm{Seconds}$
$1\,\mathrm{min}$	$1\,\mathrm{Minutes} = 60\,\mathrm{s}$
$1\,\mathrm{GHz}$	$1\,\mathrm{Gigahertz} = 10^9\,\mathrm{Hertz}$

1 Introduction

In the area of medicine, examples such as the resection of liver tumors for long-term survival (Alirr and Abd. Rahni 2019) and osteotomy planning (Zachow et al. 2003) (see figure 1.1a), that involves reshaping and realigning bones to repair or fix bone-specific issues, show that the use of computer-aided systems for surgery planning reduces the duration of treatment and heavily increases the chance of long-term survival. Both of the named medical applications simply use curves on the two-dimensional reconstructed surface of scanned medical objects to represent and visualize surgery cuts. The reconstructed surfaces will thereby be provided as triangular meshes and are often referred to as surface meshes. As curves on surface meshes are in general a basic building block for tasks involving mesh processing and segmentation (Ji et al. 2006; Kaplansky and Tal 2009) (see figures 1.1b and 1.1c), they also play a fundamental role in the areas of computer-aided geometric design, computer graphics, and visualization. Building on these research areas, significant applications can not only be found in the medical context but also for machine learning (Benhabiles et al. 2011; Park et al. 2019) and engineering.

Initially chosen curves on surfaces meshes are typically jagged due to the finite precision of the underlying mesh and emit curvature noise that is not neglectable and perceivable by the human eye. Hence, a smoothing process is applied to reduce their overall curvature and attain surface cuts with well-defined properties. In general, the result of curve smoothing might strongly deviate from the initially given curve to fulfill the provided constraints. For medical surface cutting applications, though, the shape of an initial curve is defined by domain experts, such as physicians or bioengineers, and most likely indicates relevant anatomical landmarks or surface regions. Thus, under these circumstances, the smoothing additionally requires the resulting curve to be close to its original such that no essential information is lost during the process. (Lawonn et al. 2014)[1]

[1] In this thesis, citations concerning a whole paragraph will be given after the last sentence of the very paragraph.



(a) *From Zachow et al. (2003).* **(b)** *From Lee et al. (2004).* **(c)** *From Ji et al. (2006).*

Figure 1.1: ***Application Examples for Smooth Curves on Surface Meshes***
The images show typical applications for smooth curves on surface meshes, such as osteotomy planning in medicine (left) and surface mesh cutting (middle) and segmentation (right) in computer-aided geometric design and machine learning from other publications.

Besides their mathematical correctness and convergence, curve smoothing algorithms should exhibit a certain level of adaptivity with respect to the given surface mesh and its initially chosen curve. Surface meshes are most typically an irregular grid of triangular faces that may highly vary in diameter and area. In addition, the initial curve might be extreme concerning its length, curvature, and overall shape. An algorithm to smooth curves on surface meshes needs to adapt to all these situations and still figure out the best possible result that abides to the given criteria. In conjunction with its correctness, this also means that such an algorithm needs to be robust for many different kinds of scenarios, such as self-intersecting curves and noisy surface geometries. Yet another property to take into account is the efficiency of the algorithm. To seamlessly integrate curve smoothing into the user interface of a computer-assisted system for domain-specific applications, it at least needs to provide an interactive up to real-time performance. As a consequence, the implementation and application programming interface (API) design of such algorithms is assumed to be an involved task and error-prone and should not be handled by a custom implementation for each framework.
(Lawonn et al. 2014)

Previous solutions to these problems involving the use of energy-minimizing splines (Hofer and Pottmann 2004) and generalizations of Bézier splines to three-dimensional surfaces (Mancinelli et al. 2022; Martínez, Carvalho, and Velho 2007), while offering a general approach and sophisticated tools, either suffer from poor performance and numerical instabilities or are not sufficient for applications in a medical context. Using an iterative method that is based on the reduction of local geodesic curvature, Lawonn et al. (2014) were able to construct a robust curve smoothing algorithm which preserves closeness to the initial curve. They successfully applied it to the decomposition of cerebral aneurysms and the resection planning for liver surgery. Unfortunately, the initial selection and propagation of desired curvature values when handling critical or splitting vertices is not provided or at least inconsistent.

The above publications compare the quality of generated curves to alternative algorithms and describe the algorithm's programming procedures with respect to a high-level point of view. However, the very low-level details about the composition of data structures, advice for an implementation in a specific programming language, or ways to handle difficult corner cases are typically left out. This makes the comparison of the performance and robustness of algorithms much harder and unreproducible. Furthermore, there is no widely accepted metric to compare the smoothness of two given curves which leads to a highly subjective treatment and evaluation of different smoothing algorithms.

For the implementation of curve smoothing algorithms that allow for high-performance, reproducibility, and robustness, adequate candidates would be the modern standards of the C++ programming language in conjunction with the OpenGL graphics API. C++ is a multi-paradigm language that integrates many different programming styles, such as object-oriented, functional, and data-oriented programming. It is still the de-facto standard for graphics applications and well-known to be one of the fastest languages in the world which incorporates low-level programming facilities, such as inline assembly, and efficient high-level abstraction mechanisms, like template meta programming. OpenGL is the open-source graphics API that allows programs to efficiently communicate and interact with the driver of the graphics card to visualize given data. Through the use of so-called compute shaders, written code is able to run concurrently on the GPU independently of the graphics card's manufacturer or the operating system. (*cppreference.com* 2023; Meyers 2014; *OpenGL* 2023; Reddy 2011; *Standard C++ Foundation* 2023; Stroustrup 2014; Vandevoorde, Josuttis, and Gregor 2018)

In this thesis, precisely in sections 4 and **??**, a robust and efficient algorithm, called *nanoreflex*[2], to smooth discrete curves on triangular surface meshes is developed using the C++ programming language in conjunction with the OpenGL graphics API. The method described here is based on the work of Lawonn et al. (2014) which seems to be the most promising approach for medical purposes. It improves their initial selection and propagation of desired curvatures by using weighted stencils and curvature textures. Incorporating efficient data structures and heuristics shown to work by Mancinelli et al. (2022), *nanoreflex* implements parallelized versions on the CPU and GPU which should be applicable in a wide variety of cases. The source code is provided as an open-source repository on GitHub. The necessary theoretical background to understand the design- and the implementation-specific aspects is given in section 2. Here, we will give a brief introduction to differential geometry and polyhedral manifolds. A mathematical rigorous discussion about the algorithm will be part of section 4 to properly encapsulate all the information specific to the implementations. Section 3 refers to some related work concerning general curves, geodesics and the smoothing of curves on surfaces. Afterwards, in section **??**, *nanoreflex* is applied to the segmentation of lung lobes (Park et al. 2019). In section 5, the algorithm will be used on the "Thingi10K" dataset and other test cases to evaluate its robustness and efficiency. At the end, in section 6, I will give a brief summary of the thesis and a discussion dealing with advantages, disadvantages, and further improvements.

[2]Markus Pawellek (2023a). *nanoreflex. Reactive and Flexible Curve Smoothing on Surface Meshes*. URL: https://github.com/lyrahgames/nanoreflex (visited on 01/15/2023).

2 Mathematical Preliminaries

To systematically approach the topic of curve smoothing algorithms for surface meshes, basic knowledge in the topics of differential geometry for curves and its further application and generalization to polyhedral surfaces is administrable. As modern differential geometry heavily builds on the mathematical tools given in the area of analysis on manifolds, the reader is assumed to be familiar with its basic concepts. For convenience, section A in the appendix provides a brief introduction to clarify and remind of notations.

2.1 Differential Geometry of Curves

The theory of smooth curves in space or smooth surfaces is one of the main concerns of classical differential geometry. Therefore, we refer to some standard textbooks, namely Goldhorn, Heinz, and Kraus (2009), Carmo (2016), Kühnel (2013), and Stahl and Stenson (2013), for a more excessive and thorough introduction on this topic. For the purpose of consistent notation and as a reminder, the main concepts of smooth curves are briefly given in the following in the sense of classical and modern differential geometry based on these books.

In the fields of analysis and numerical mathematics, it is a natural approach to extend a well-founded and -understood theory for smooth objects to their discrete counterparts and vice versa. Hereby, discrete or smooth objects will be typically characterized by the limit of sequences of smooth or discrete objects, respectively. This procedure may then allow for a generalization of properties and statements from one to the other case. Surface mesh curves, as they will be defined in section 2.2, can exactly be seen as such discrete counterparts to the shapes of special classes of smooth parameterized curves (Polthier and Schmies 2006). (Cheney and Kincaid 2008; Elstrodt 2018; Forster 2016; Goldhorn, Heinz, and Kraus 2009)

DEFINITION 2.1: Open and Closed Parameterized Curves

Let $n \in \mathbb{N}$, $k \in \mathbb{N}_\infty$, and $[a, b] \subset \mathbb{R}$ be a compact interval.

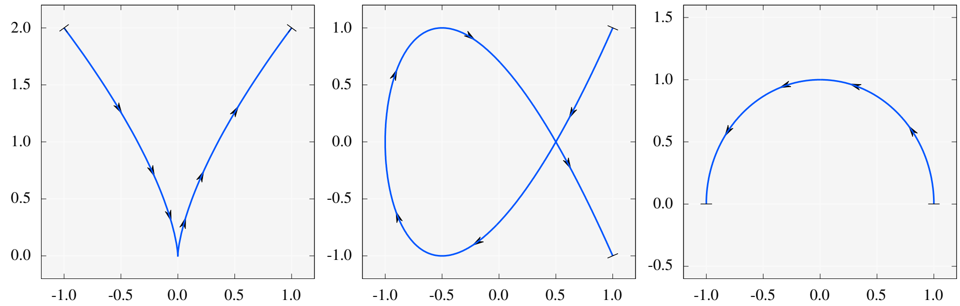
An (open) (n-dimensional) (parameterized) curve (of class C^k) is a k-times continuously differentiable function $\gamma\colon [a, b] \to \mathbb{R}^n$ on the compact interval $[a, b]$.

A closed (n-dimensional) (parameterized) curve (of class C^k) is a k-times continuously differentiable function $\gamma\colon \mathbb{T}^1[a, b] \to \mathbb{R}^n$ on the respective one-dimensional topological torus $\mathbb{T}^1[a, b]$.

Parameterized curves of class C^∞ are also called smooth.

In the sense of this definition, a curve needs to at least provide some kind of smoothness in the form of continuous differentiability. Trying only to rely on a continuity property, the definition would also include pathological cases, such as space-filling curves that cover a whole square in the plane (Kühnel 2013). Other generalizations therefore rely on the use of either rectifiable or piecewise continuously differentiable curves.

Defining closed curves to be smooth functions on the topological space of a one-dimensional torus is a technicality that automatically takes care of the agreement of values and derivatives at the two boundary points which are glued together (Carmo 2016; Stahl and Stenson 2013). The change of topological requirements for a closed curve still allows to interpret it as a



(a) *Semicubical Parabola*

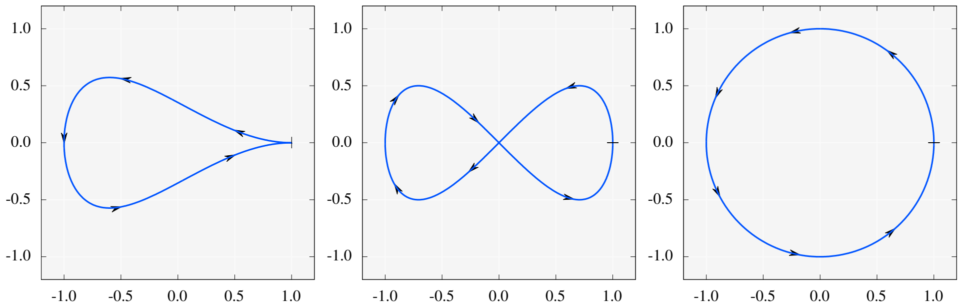
$$\gamma\colon [-1,1] \to \mathbb{R}^2$$
$$\gamma(t) := \begin{pmatrix} t^3 \\ 2t^2 \end{pmatrix}$$

(b) *Lissajous Curve*

$$\gamma\colon [0,2\pi] \to \mathbb{R}^2$$
$$\gamma(t) := \begin{pmatrix} \cos t \\ \cos\left(\frac{3}{2}t\right) \end{pmatrix}$$

(c) *Half-Circle*

$$\gamma\colon [0,\pi] \to \mathbb{R}^2$$
$$\gamma(t) := \begin{pmatrix} \cos t \\ \sin t \end{pmatrix}$$



(d) *Teardrop Curve*

$$\gamma\colon \mathbb{T}^1[0,2\pi] \to \mathbb{R}^2$$
$$\gamma(t) := \begin{pmatrix} \cos(t) \\ \sin(t)\sin^3\left(\frac{1}{2}t\right) \end{pmatrix}$$

(e) *Lissajous Curve*

$$\gamma\colon \mathbb{T}^1[0,2\pi] \to \mathbb{R}^2$$
$$\gamma(t) := \begin{pmatrix} \cos t \\ \frac{1}{2}\sin 2t \end{pmatrix}$$

(f) *Circle*

$$\gamma\colon \mathbb{T}^1[0,2\pi] \to \mathbb{R}^2$$
$$\gamma(t) := \begin{pmatrix} \cos t \\ \sin t \end{pmatrix}$$

Figure 2.1: ***Examples of Smooth Parameterized Curves***
All the plots show a smooth two-dimensional parameterized curve γ of a different class. The first row from left to right shows a general curve, a regular curve with self-intersection, and a regular and simple curve parameterized by arc length. The second row shows closed counterparts, respectively. Hereby, the black arrows indicate the orientation of the chosen parameterization and the black lines its start and end.

general curve. For this reason, when referring to parameterized curves, both closed and open curves are meant. To get an impression of the implications of this definition, in figure 2.1 typical examples of smooth curves are shown.

For most of the considered applications, the actual parameterization of a curve is not essential. Only the shape, given by its image, is used, referred to, and transformed. But using the shape and neglecting the parameterization of a curve does only make sense if the shape at least fulfills the properties of a one-dimensional manifold. Unfortunately, as can be seen in figure 2.1, the shape of parameterized curves may exhibit more general structures, such as sharp edges and self-intersections, and, thus, does not necessarily abide to these constraints. As a consequence, more regular and specialized classes of curves need to be looked at. (Carmo 2016; Goldhorn, Heinz, and Kraus 2009; Kühnel 2013)

DEFINITION 2.2: Regular and Simple Parameterized Curves

Let γ be a parameterized curve. Then γ is said to be simple if it is injective and regular if the following statement holds.

$$\forall t \in \mathcal{D}(\gamma)^\circ : \quad \gamma'(t) \neq 0$$

Furthermore, if $\|\gamma'\| = 1$ then γ is parameterized by arc length.

The definition states that the derivative of a regular curve is, apart from its boundary, never zero and therefore a map of full-rank at every inner point. Hence, a regular curve is an immersion into Euclidean space and, as a result, its shape is an immersed submanifold. Please note, an immersed submanifold still may contain self-intersections. Using a curve that is parameterized by arc length is a way of defining a canonical parameterization for regular curves. It is clear by definition, that every curve parameterized by arc length is also a regular curve.
(Carmo 2016; Goldhorn, Heinz, and Kraus 2009; Kühnel 2013)

In applications that are able to handle self-intersections, regular curves are already sufficiently constrained. Indeed, a robust and stable algorithm for curve smoothing should be able to cope with self-intersections and even remove them if necessary. Still, to properly understand the transition, a regular curve shall become an embedding, such that, according to section A, its shape would be a one-dimensional manifold. To state that a curve is also simple, ensures it exhibits no self-intersections other than the single intersection for closed curves. The results of the ideas of this and the previous paragraph are rigorously summarized in the following proposition. As this statement is a direct corollary, no additional rigorous proof will be given for it. (Carmo 2016; Goldhorn, Heinz, and Kraus 2009; Kühnel 2013)

COROLLARY 2.1: Simple, Regular Curves are Embeddings

Let γ be a smooth parameterized curve that is simple and regular. Then γ is a smooth embedding into Euclidean space and its image is an orientable and connected one-dimensional submanifold with boundary. If γ is closed, then its image has no boundary.

The fundamental part of the concept of geodesics is the curvature. To gain an intuitive approach to a definition of curvature of a regular curve, we need to be able to evaluate its length and arc length in Euclidean space. These will allow us to properly provide a canonical parameterization to simplify the use of regular curves.

DEFINITION 2.3: Length of Curves

Let γ be a parameterized curve on $[a, b]$ or $\mathbb{T}^1[a, b]$. Then the length $L(\gamma)$ and arc length, given by $s_\gamma : [a, b] \to [0, L(\gamma)]$ or $s_\gamma : \mathbb{T}^1[a, b] \to \mathbb{T}^1[0, L(\gamma)]$, respectively, of γ are defined by the following expressions (see section C for consistency proof).

$$L(\gamma) := \int_a^b \|\gamma'(t)\| \,\mathrm{d}t \qquad s_\gamma(t) := \int_a^t \|\gamma'(x)\| \,\mathrm{d}x$$

The definition of the length and arc length mainly stems from a physical interpretation. Hereby, a parameterized curve is a trajectory of a particle moving in space with a specific velocity over time. In this interpretation, the particle's velocity is given by the derivative of the curve's parameterization. For evaluating the distance covered, one needs to integrate over the particle's speed, that is given by the magnitude of its velocity, at each point in time.

A direct consequence for curves parameterized by arc length is that their length can be simply computed by $L(\gamma) = b - a$. This simplification leads to the idea to reparameterize general regular curves, such that they become curves parameterized by arc length and, accordingly, inheriting their good properties. The details for this approach are given by the following lemma for open parameterized curves with its proof provided in the appendix in section C.

LEMMA 2.2: Regular curves can be parameterized by arc length

Let $n \in \mathbb{N}$, $k \in \mathbb{N}_\infty$, $[a, b] \subset \mathbb{R}$ be a compact interval, and $\gamma\colon [a, b] \to \mathbb{R}^n$ be an n-dimensional parameterized curve of class C^k that is regular. Then up to a constant shift, there exists a unique k-times continuously differentiable and bijective function $u\colon [c, d] \to [a, b]$ on a compact interval $[c, d] \subset \mathbb{R}$ with strictly positive derivative, such that the composition $\gamma \circ u$ is a curve parameterized by arc length.

The proof of this lemma shows, that the inverse of the arc length of a regular curve up to some shifting can be used as a unique reparameterization to transform it into a curve parameterized by arc length. According to this, the arc length provides a canonical parameterization.

DEFINITION 2.4: Canonical Parameterization by Arc Length

Let γ be a regular curve. Then its canonical parameterization by arc length $\bar{\gamma}$ is defined by the following expression.

$$\bar{\gamma} := \gamma \circ s_\gamma^{-1}$$

For smooth curves parameterized by arc length, the definition of their curvature is now straightforward, as we understand it as the amount the curve bends at a point when moving along its trajectory. Referring to differential calculus, the magnitude of the second derivative exactly describes this specific amount. (Forster 2016; Goldhorn, Heinz, and Kraus 2009)

DEFINITION 2.5: Curvature of Regular Curves

Let $k \in \mathbb{N}_\infty$ with $k \geq 2$, γ be a curve parameterized by arc length of class C^k, and φ be a regular curve of class C^k. Their respective curvatures $\kappa(\gamma)$ and $\kappa(\varphi)$ are defined by the following expressions.

$$\kappa(\gamma) := \|\gamma''\| \qquad \kappa(\varphi) := \kappa(\bar{\varphi}) \circ s_\varphi$$

The definition involves second-order derivatives and therefore at least requires the curves to be of class C^2. In differential geometry, typically only smooth curves are observed where this states no further constraints (Goldhorn, Heinz, and Kraus 2009).

To finally provide a definition for geodesics, it is no longer sufficient to look at smooth curves freely traversing the Euclidean space. Instead, we will embed curves into smooth surfaces which already exhibit intrinsic curvature. In this scenario, the overall curvature can be decomposed into the geodesic and normal curvature.

DEFINITION 2.6: Geodesic and Normal Curvature of Curves

Let M be a Riemannian submanifold embedded in Euclidean space and γ be a *curve embedded in M and parameterized by arc length. The geodesic curvature $\bar{\kappa}_{\mathrm{g}}(M,\gamma)$ and the normal curvature $\bar{\kappa}_{\mathrm{n}}(M,\gamma)$ are defined as follows.*

$$\bar{\kappa}_{\mathrm{g}}(M,\gamma) := \left\|P_{\mathrm{T}_\gamma M}(\gamma'')\right\| \qquad \bar{\kappa}_{\mathrm{n}}(M,\gamma) := \left\|P_{\mathrm{N}_\gamma M}(\gamma'')\right\|$$

The geodesic curvature describes the amount a curve bends measured by an intrinsic observer of the ambient surface that is moving along the curve without any knowledge of the extrinsic curvature. The normal curvature on the other hand is only observable from the ambient space and depends on the extrinsic curvature and the tangential vector of the curve. As a direct consequence of the definition, the following statement can be deduced without further proof. (Carmo 2016; Goldhorn, Heinz, and Kraus 2009; Kühnel 2013)

COROLLARY 2.3: Relationship of Geodesic and Normal Curvature

Let M be a Riemannian submanifold embedded in Euclidean space and γ be a curve embedded in M and parameterized by arc length. Then the following holds.

$$\kappa^2(\gamma) = \bar{\kappa}_{\mathrm{g}}^2(M,\gamma) + \bar{\kappa}_{\mathrm{n}}^2(M,\gamma)$$

Up to now, all curvature definitions have been non-negative scalar functions that did not provide any orientation. But in the context of orientable Riemannian submanifolds of dimension two, further knowledge about the direction of geodesic bending can be provided (Goldhorn, Heinz, and Kraus 2009). This is especially useful for implementation-specific work. Providing orientations for manifolds and curves allows for a more efficient moving of points along a curve or surface and access to its neighbors.

DEFINITION 2.7: Oriented Curvatures of Curves

Let M be a two-dimensional oriented Riemannian submanifold embedded in $\mathbb{R}^3$ and γ be a curve embedded in M and parameterized by arc length.
Furthermore, let $\nu\colon M \to \mathrm{N}M$ be a differentiable field of positively oriented unit normals and $\mu\colon M \to \mathrm{T}M$ be a differentiable field of normalized tangent vectors such that all values of $(\gamma', \mu\circ\gamma)$ are a positively oriented basis.
The oriented geodesic and normal curvature are defined as follows, respectively.

$$\kappa_{\mathrm{g}}(M,\gamma) = \langle \mu\circ\gamma \,|\, \gamma'' \rangle \qquad \kappa_{\mathrm{n}}(M,\gamma) = \langle \nu\circ\gamma \,|\, \gamma'' \rangle$$

The oriented geodesic curvature is positive when the curve bends to the left and negative when it bends to the right with respect to the chosen positive orientation of the ambient submanifold. The oriented normal curvature is not as important in our context as it only describes the bending of the surface along the trajectory. Despite this, the previous corollary still holds for these oriented formulations.

For an intrinsic observer positioned in the ambient submanifold, the curve will be straight if it does not exhibit any geodesic curvature, because then it neither bends to the right nor to the left at any point. This is one of the characterizing properties of geodesics in classical and modern differential geometry and, for us, will serve as a definition and generalization (Carmo 2016; Goldhorn, Heinz, and Kraus 2009; Kühnel 2013; Polthier and Schmies 2006).

DEFINITION 2.8: Straightest Geodesic

Let M be a Riemannian submanifold embedded in Euclidean space and γ be a curve embedded in M° and parameterized by arc length. Then γ is called a (straightest) geodesic if $\bar{\kappa}_{\mathrm{g}}(M, \gamma) = 0$.

The problem of finding a geodesic for a given ambient submanifold, a starting point, and a starting direction is called the geodesic initial value problem. In a rigorous formulation, it breaks down to a system of second-order ordinary differential equations for which it can be proven that a locally unique solution exists in some sense (Polthier and Schmies 2006). If no starting direction is given, but instead the point to end at, we call it the geodesic boundary value problem. The boundary value problem is not as easy to handle as the initial value problem. There is typically no unique solution despite the fact that there may be no solutions at all (Polthier and Schmies 2006).
(Carmo 2016; Goldhorn, Heinz, and Kraus 2009; Kühnel 2013)

Another important alternative characterization states that geodesics are a critical point of their length functional with respect to variations tangential to the ambient submanifold (Polthier and Schmies 2006). In general, this is understood as the concept of locally shortest geodesics and means that a slight change of the trajectory at any point will only increase the overall length of the curve.

DEFINITION 2.9: Locally Shortest Geodesic

Let M be a Riemannian submanifold embedded in Euclidean space and γ be a curve embedded in M and parameterized by arc length.
Then γ is called a (locally shortest) geodesic if the following statement is fulfilled.

$$\forall \varphi \in \mathrm{C}^\infty(\mathcal{D}(\gamma), \mathrm{T}M), \varphi|_{\partial\mathcal{D}(\gamma)} = 0 \colon \qquad \frac{\partial}{\partial \varepsilon} L(\gamma + \varepsilon\varphi)\bigg|_{\varepsilon=0} = 0$$

Unfortunately, a critical point is not necessarily a local minimum of the length of a curve and by no means a global minimum. Regarding smooth submanifolds without boundary, the concept of locally shortest geodesics is equivalent to the concept of straightest geodesics. But embedding a curve in a polyhedral non-differentiable submanifold, these characterizing properties of geodesics are no longer equivalent (Polthier and Schmies 2006). In such a context, multiple non-trivial generalizations of geodesics are possible.

In the context of submanifolds with an actual boundary, the concept of straightest geodesics is not as easy to generalize as for locally shortest geodesics. Alas, in terms of mathematical literature about differential geometry, manifolds with boundary are either treated as an afterthought or not at all (Carmo 2016; Goldhorn, Heinz, and Kraus 2009; Kühnel 2013). But regarding the application of differential geometry to computer geometry and mesh processing, the analysis and handling of boundaries is inevitable. For example, figure 2.2 schematically shows the Euclidean plane $M \coloneqq \mathbb{R}^2 \setminus U_1(0)$ where the open unit disk has been removed. The boundary ∂M is given by the unit circle. Hence, M is a smooth Riemannian manifold with a non-empty boundary. For the two given points in M, their geodesic connection can no longer be obtained by using the straight line as it is no longer a part of M. So, in the sense of locally shortest geodesics, the connection needs to partially adapt to the boundary (Albrecht and Berg 1991). A simple calculation then shows that locally shortest geodesics are only of class C^1 and

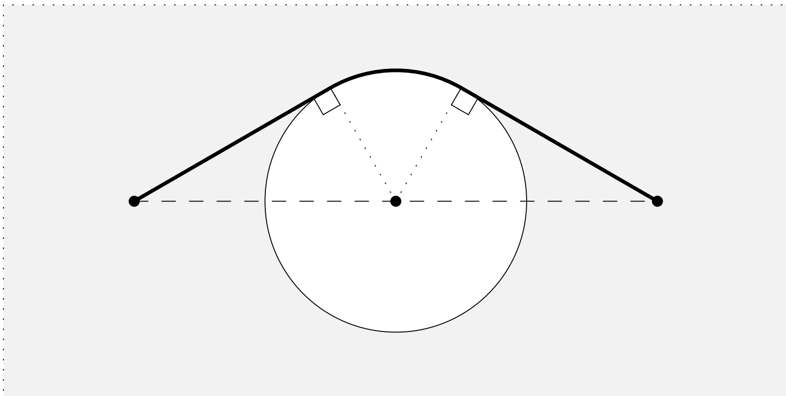


Figure 2.2: ***Geodesics on Smooth Manifolds with Boundary***
The image shows a schematic view of Euclidean plane $M := \mathbb{R}^2 \setminus U_1(0)$ where the open unit disc has been removed. As it is no longer possible to connect the given points by a straight line, a locally shortest geodesic for the boundary value problem must partially adapt to the boundary. The marked geodesic is only of class C^1 and does not fulfill the requirements for a straightest geodesic. Furthermore, the uniqueness of solutions for the initial value problem fails in the presence of boundaries.

no longer smooth (Alexander and Alexander 1981). Thus, they fail to satisfy the requirements of straightest geodesics. As a consequence, the Cauchy uniqueness regarding the initial value problem clearly fails and the description of geodesics in terms of differential equations, as it was done for straightest geodesics, becomes infeasible (Alexander, Berg, and Bishop 1986, 1987).

2.2 Polyhedral Surfaces and Discrete Geodesics

In mathematics, smooth manifolds are well understood and provide a rich set of good properties and generalizations. Unfortunately, arbitrarily complex smooth manifolds are not easily representable in a computer system due to the finite amount of available storage and precision. In the spirit of scientific computing, surfaces of real-world objects are typically approximated by many triangles which are put together to a polyhedral surface. This interpretation is slightly altered in the area of computational geometry and in the context of this thesis. A polyhedral surface is no longer looked at as an approximation to a smooth surface but viewed as an exact description of the shape of interest. (Sharp and Crane 2020)

Polyhedral surfaces no longer fulfill the requirements of any differentiable manifold and, at a first glance, lack many good properties. To cope with the implied disadvantages, they could be generalized by the rarely used concept of smooth manifolds with corners. Here, coordinate charts are defined over the cube space instead of the half space to directly include corners and edges (see appendix A definition A.1 and Joyce (2009)). In the topological case, this concept is equivalent to manifolds with boundaries and does not provide further insights. But because the half space is only homeomorphic and not diffeomorphic to the cube space, this statement does not hold in the differential case of manifolds. Therefore using smooth manifolds with corners to model polyhedral surfaces could lead to a simpler handling of structures and theorems. Unfortunately, according to Joyce (2009), the exact definition of manifolds with corners is not universally agreed upon and still raises questions and problems. Therefore, in this thesis, the focus lies on the definitions and ideas given by Polthier and Schmies (2006) and Martínez,

(a) *counterclockwise* **(b)** *clockwise*

Figure 2.3: ***Orientations of a Triangle***
The images schematically visualize the two distinct orientations of a triangle that correspond to the chosen triangle normal. They are given with respect to the outwards-pointing normal of the Euclidean plane and switch when referring to the inwards-pointing normal.

Velho, and Carvalho (2005).

The geometric primitives of polyhedral surfaces are triangles. By attaching triangles to each other at their edges, arbitrary complex surfaces can be approximated. To facilitate the description of this process and the construction of polyhedral surfaces, what follows is a concise but rigorous concept for topological triangles and their orientation.

DEFINITION 2.10: Triangle

Let $n \in \mathbb{N}$ with $n \geq 2$ and $A, B, C \in \mathbb{R}^n$ affinely independent points. Then the (topological) triangle $\triangle$ (embedded in $\mathbb{R}^n$) is given by the following expression.

$$\triangle := \{uA + vB + wC \mid u, v, w \in [0,1], u + v + w = 1\}$$

The triangle's vertices $\mathcal{V}(\triangle)$ and edges $\mathcal{E}(\triangle)$ are hereby defined as follows.

$$\mathcal{V}(\triangle) := \{A, B, C\} \qquad \mathcal{E}(\triangle) := \left\{\overline{AB}, \overline{BC}, \overline{CA}\right\}$$

In the topological context, the triangle can simply be defined by using barycentric coordinates that build a convex linear combination of its vertices. Its undirected graph on the other hand can be represented by its vertices and edges. Please note that the property of the vertices to be affinely independent is, again, only a technicality to disallow for degenerate triangles consisting of a single point or edge.

Triangles are planar surfaces and, hence, its geodesics are represented by straight lines. By definition they are also convex sets, such that straight lines connecting any two chosen points lie inside the triangle. As a direct consequence, inside a triangle, there is always a unique solution for the geodesic boundary value problem. To show some other basic properties, I first introduce a restricted parameter set.

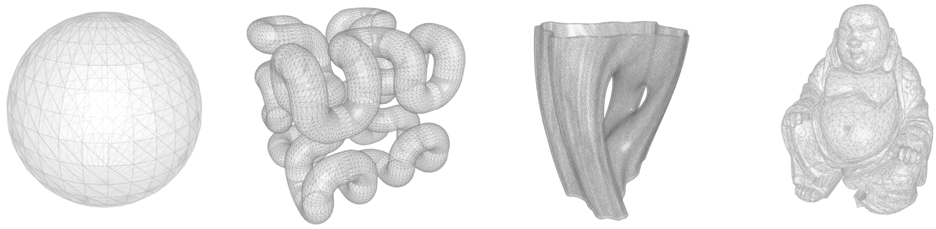
$$D := \left\{(u,v) \in [0,1]^2 \;\middle|\; u + v < 1\right\}$$

Furthermore, the set $\Pi(\triangle)$ of all vertex arrangements is needed to also handle orientation.

$$\Pi(\triangle) := \{\pi\colon \{1,2,3\} \to \mathcal{V}(\triangle) \mid \pi \text{ bijective}\}$$

Then let $\pi \in \Pi(\triangle)$ be arbitrary and define the following coordinate charts and atlases.

$$\varphi_\pi\colon \overline{D} \to \triangle \qquad \varphi_\pi(u,v) := (1-u-v)\pi_1 + u\pi_2 + v\pi_3$$



(a) *Sphere* **(b)** *Hilbert Cube* **(c)** *Julia Vase* **(d)** *Buddha*

Figure 2.4: ***Examples of Polyhedral Surfaces***
*The images show examples for polyhedral surfaces embedded into Euclidean space approximating various complex forms. The models for the Hilbert Cube, the Julia Vase, and the Buddha have been taken from the "Thingi10K" dataset (*Zhou and Jacobson 2016*).*

$$\mathcal{A} := \left\{ \left(\varphi_\pi(D), \varphi_\pi|_D^{-1} \right) \;\middle|\; \pi \in \Pi(\triangle) \right\} \qquad \mathcal{A}^\circ := \left\{ \left(\triangle^\circ, \varphi_\pi|_{D^\circ}^{-1} \right) \;\middle|\; \pi \in \Pi(\triangle) \right\}$$

Given $\mathcal{A}$ as atlas for $\triangle$, it follows directly that a topological triangle is an orientable topological 2-manifold with boundary embedded in $\mathbb{R}^n$. Using $\mathcal{A}^\circ$ as atlas for $\triangle^\circ$, the open triangle is an orientable smooth 2-manifold without boundary.

For orientability, let the following equivalence relation define when two vertex arrangements $\pi, \pi' \in \Pi(\triangle)$ induce the same orientation.

$$\pi \sim \pi' \quad :\Longleftrightarrow \quad \exists \sigma \in \mathrm{S}_3, \operatorname{sgn}\sigma = 1 : \quad \pi' = \pi \circ \sigma$$

The quotient space of vertex arrangements then only consists of two elements which uniquely model the possible orientations of a triangle that are schematically visualized in figure 2.3.

$$\Omega(\triangle) := \Pi(\triangle)/\sim = \{[\pi], [\bar{\pi}]\}$$

Definition 2.11: Oriented Triangle

An oriented triangle is tuple $\triangle_{[\pi]} := (\triangle, [\pi])$ consisting of a triangle $\triangle$ and an orientation $[\pi] \in \Omega(\triangle)$. The set of directed edges of $\triangle_{[\pi]}$ is defined as follows.

$$\mathcal{E}(\triangle_{[\pi]}) := \{\overrightarrow{\pi_1\pi_2}, \overrightarrow{\pi_2\pi_3}, \overrightarrow{\pi_3\pi_1}\}$$

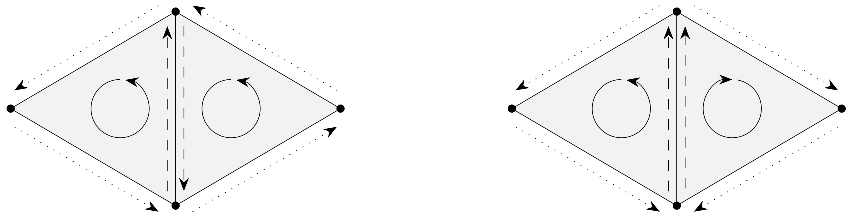
An oriented triangle is a triangle that has been equipped with an orientation. In this sense, it is an oriented topological 2-manifold with an oriented boundary. Its oriented boundary can be characterized by the union of its directed edges. The structure of an oriented triangle can be modeled by a directed graph connecting its vertices by its directed edges. With all these details, a rigorous definition of the polyhedral surfaces and its orientation can now be given.

Definition 2.12: Polyhedral Surface and Surface Mesh

Let $n \in \mathbb{N}$ with $n \geq 2$ and $\mathcal{T} \neq \emptyset$ be a finite set of triangles embedded in $\mathbb{R}^n$. Let $S = \bigcup \mathcal{T}$ be a two-dimensional topological manifold (with boundary), such that for all $\triangle_1, \triangle_2 \in \mathcal{T}$ with $\triangle_1 \neq \triangle_2$ the following holds.

$$\triangle_1 \cap \triangle_2 \in \{\emptyset\} \cup [\mathcal{V}(\triangle_1) \cap \mathcal{V}(\triangle_2)] \cup [\mathcal{E}(\triangle_1) \cap \mathcal{E}(\triangle_2)]$$

In this case, S is called a (topological) polyhedral surface (embedded in $\mathbb{R}^n$).



(a) *compatible* **(b)** *incompatible*

Figure 2.5: ***Compatible and Incompatible Triangle Orientations***
The images schematically visualize compatible and incompatible orientations of two adjacent triangles that share a common undirected edge. For compatible orientations, the triangles are not allowed to share a directed edge such that the boundary of their union is again an oriented topological 1-manifold.

With $\mathcal{V}(S)$ we denote its vertices, with $\mathcal{E}(S)$ its edges, and with $\mathcal{F}(S)$ its faces.

$$\mathcal{V}(S) \coloneqq \bigcup_{\triangle \in \mathcal{T}} \mathcal{V}(\triangle) \qquad \mathcal{E}(S) \coloneqq \bigcup_{\triangle \in \mathcal{T}} \mathcal{E}(\triangle) \qquad \mathcal{F}(S) \coloneqq \mathcal{T}$$

For a clear distinction to the polyhedral surface itself, the following set is called the (topological) surface mesh.

$$\mathcal{M}(S) \coloneqq \bigcup \mathcal{E}(S)$$

The definition makes sure that each pair of triangles may only share a common vertex or edge and does not overlap in any other way. Additionally, requiring that the polyhedral surface is a topological 2-manifold, lets neighborhoods of any point be parameterized by two-dimensional charts. So, an edge, for example, may only belong either to one or two triangles at the same time. Figure 2.4 shows four examples of polyhedral surfaces that show the potential of the complexity a surface may reach. It is also clear by definition, that the topological triangle is a polyhedral surface itself.

I also included a rather uncommon distinction of surface meshes and polyhedral surfaces. Whereas the polyhedral surface describes the topological manifold, the surface mesh is the topological model of the underlying graph connecting the vertices of the surface by edges. This is done mainly for consistency and convenience when classifying curves. Also, my definition of polyhedral surfaces is more restricted than the one of Polthier and Schmies (2006) as it does not allow for an infinite amount of triangles or arbitrary flat intrinsic metrics. Concerning the applications of curve smoothing algorithms, this is not a serious constraint because real-world surfaces in the mesh processing context will mostly have a finite extent and only allow for finitely many faces. Apart from that, regarding all the following considerations, my definition can simply be exchanged with the more general form.

For the construction of orientation, I will build on the tools given above for oriented triangles. The idea is to consistently choose orientations for every triangle such that the boundary of two adjacent triangles keeps to be an oriented topological 1-manifold — the orientations are called compatible. As it can be seen in figure 2.5, the orientations of adjacent triangles are only compatible if they do *not* share any directed edge.

DEFINITION 2.13: Oriented Polyhedral Surface

Let S be a polyhedral surface. Then S is called orientable if there exists a function $\omega\colon \mathcal{F}(S) \to \bigsqcup_{\triangle\in\mathcal{F}(S)} \Omega(\triangle)$ that selects an orientation for each contained triangle with the following property.

$$\forall \triangle_1, \triangle_2 \in \mathcal{F}(S), \triangle_1 \neq \triangle_2\colon \quad \mathcal{E}(\omega(\triangle_1)) \cap \mathcal{E}(\omega(\triangle_2)) = \emptyset$$

In this case, ω is called an orientation of S and (S,ω) an oriented polyhedral surface. The set of all orientations for S is denoted with $\Omega(S)$.

As before, oriented topological triangles are by definition oriented polyhedral surfaces and show a certain consistency in the definition. In the case of compatible orientations, two adjacent triangles exhibit the same outer normal when unfolded into the Euclidean plane. For connected polyhedral surfaces without boundary, the outer normal of each face then consistently points to the outside or inside of the circumscribed volume depending on the chosen orientation. But as polyhedral surfaces are not required to be connected, in general, more than two orientations might exist.

Especially in this thesis, all algorithms will be constructed for oriented polyhedral surfaces to facilitate their implementation and efficiency. Concerning real-world data, this is not an actual restriction. Unorientable surface meshes mostly arise from artificial construction or numerical errors that can be handled by other mesh preprocessing techniques. For applications that need to process surface meshes, the surfaces mostly are the boundary of finite volumes in three-dimensional Euclidean space. These volumes can be viewed as open submanifolds of $\mathbb{R}^3$. Because the Euclidean space itself is oriented, these volumes and, as a consequence, their boundaries need to be oriented, too (see appendix A).

There are artificial cases, such as the polyhedral approximation of a Möbius strip, of non-orientable polyhedral surfaces. In figure 2.6, polyhedral approximations for the torus and the Möbius strip are shown. As it is known from differential geometry, the torus has no boundary and is orientable. Whereas the flat Möbius strip has a boundary and is not orientable, adding thickness to it, makes it the boundary of a connected orientable 3-submanifold of the Euclidean space. Hence, The Möbius strip with thickness has no boundary, is orientable, and homeomorph to the torus. So, the flat Möbius strip can be seen as an ill-formed example.

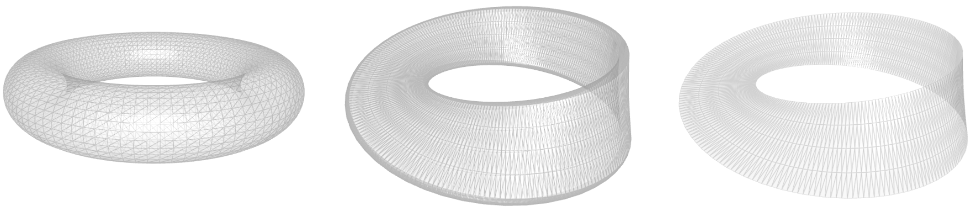
Referring again to Polthier and Schmies (2006), the Gauss curvature of smooth 2-manifolds without boundary embedded into Euclidean space can be generalized by measuring the available angle around a point — the total vertex angle. The introduction of this concept allows for the classification of points, especially vertices, on a polyhedral surface based on their discrete curvature properties and, as a consequence, will lead to statements about geodesics.

DEFINITION 2.14: Total Vertex Angle and Total Gauss Curvature

Let $\triangle$ be a triangle and A, B, C its vertices. Define the following.

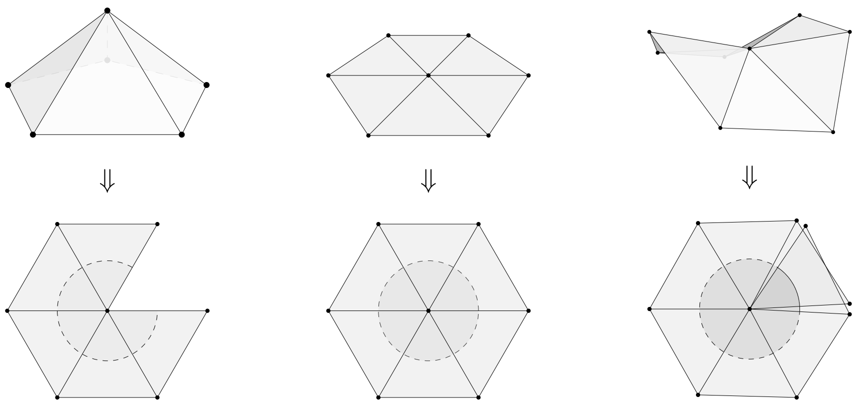
$$\vartheta_\triangle\colon \triangle \to [0, 2\pi] \qquad \vartheta_\triangle|_{\triangle^\circ} := 2\pi \qquad \vartheta_\triangle|_{\partial\triangle\setminus\mathcal{V}(\triangle)} := \pi$$

$$\vartheta_\triangle(A) := \sphericalangle\left(\overrightarrow{AB}, \overrightarrow{AC}\right) \qquad \vartheta_\triangle(B) := \sphericalangle\left(\overrightarrow{BA}, \overrightarrow{BC}\right) \qquad \vartheta_\triangle(C) := \sphericalangle\left(\overrightarrow{CB}, \overrightarrow{CA}\right)$$



(a) *Torus* **(b)** *Möbius Strip with Thickness* **(c)** *Möbius Strip*

Figure 2.6: ***Möbius Strip as Orientable Polyhedral Surface***
The images show examples of polyhedral approximations of the torus (left), the Möbius strip with thickness (middle), and the standard flat Möbius strip (right). The flat Möbius strip is not the boundary of volume and is therefore not automatically orientable. Adding thickness changes the topology and makes it homeomorphic to the torus. Thus, a real-world Möbius strip is not an actual Möbius strip and will still carry orientation.



(a) *Spherical Vertex:* $K_S(v) > 0$ **(b)** *Euclidean Vertex:* $K_S(v) = 0$ **(c)** *Hyperbolic Vertex:* $K_S(v) < 0$

Figure 2.7: ***Classification of Vertices on a Polyhedral Surface***
For a polyhedral surface S, an inner point $v \in S^\circ$ is classified by the value of the Gauss curvature $K_S(v)$ of that point. The first row of images show schematic examples of the three categories, namely spherical, Euclidean, and hyperbolic vertices. The second row unfolds the neighboring triangles into the Euclidean plane. The images have been modeled on Polthier and Schmies (2006) and Crane et al. (2020).

Let S be a polyhedral surface. Then its total vertex angle ϑ_S is defined as follows.

$$\vartheta_S \colon S \to \mathbb{R}_0^+ \qquad \vartheta_S(v) := \sum_{\triangle \in \mathcal{F}_v(S)} \vartheta_\triangle(v) \qquad \mathcal{F}_v(S) := \{\triangle \in \mathcal{F}(S) \mid v \in \triangle\}$$

The (total) Gauss curvature K_S of S is then given by the following expression.

$$K \colon S^\circ \to \mathbb{R} \qquad K_S(x) := 2\pi - \vartheta_S(x)$$

First, nearly all points in the interior of S have a Gauss curvature of zero. Only inner vertices of S exhibit values different from zero. This is consistent with the fact that polyhedral surfaces are not an approximation but the actual object of interest. As it can be seen in figure 2.7, vertices of a polyhedral surface are therefore classified to be spherical, Euclidean, or hyperbolic.

Formally, the Gauss curvature is not defined for the boundary of the polyhedral surface. But it is possible to use the above formula for a generalization to boundary vertices. According

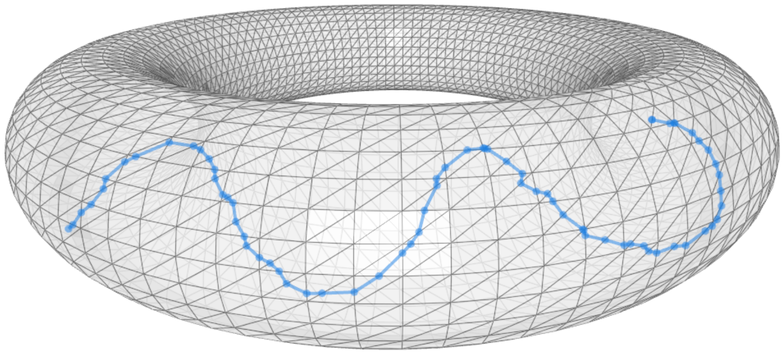


Figure 2.8: ***Example of a Regular Surface Mesh Curve on a Torus***

to Polthier and Schmies (2006), together with the discrete geodesic curvature this would lead to a unique discrete formulation of the Gauss-Bonet theorem. Still, the evaluation of the Gauss curvature formula at boundary vertices results in non-intuitive values that are hard to interpret or use in algorithms. The Gauss curvature is bounded from above but not from below. This is an inconsistency that arises due to discretization process when generalizing to polyhedral surfaces.

In analogy to the previous subsection about curves on smooth manifolds, I will introduce discrete curves on the surface mesh of polyhedral surfaces. This will allow for the discretization of geodesics and the geodesics problems. Furthermore, surface mesh curves build the fundamental building blocks for data structures concerning the construction of the curve smoothing algorithm.

DEFINITION 2.15: Surface Mesh Curve

Let S be a polyhedral surface, $n \in \mathbb{N}$, and $x \in \mathcal{M}(S)^{n+1}$ be a sequence of control points, such that the following property is fulfilled.

$$\forall k \in \mathbb{N}, k \leq n \colon \exists \triangle \in \mathcal{F}(S) \colon \quad x_k, x_{k+1} \in \triangle$$

The set of points given by connecting adjacent control points by a straight line is called the (topological) surface mesh curve $\Gamma(x)$ generated by x. Any piecewise continuously differentiable parameterization $\gamma \colon \mathcal{D}(\gamma) \to \Gamma(x)$, that is equivalent the following, is called a (parameterized) surface mesh curve generated by x.

$$\gamma \colon [0, n] \to S \qquad \gamma(t) := (\lfloor 1 + t \rfloor - t)\, x_{1+\lfloor t \rfloor} + (t - \lfloor t \rfloor)\, x_{1+\lceil t \rceil}$$

The length of $\Gamma(x)$ or γ is then defined to be the following expression.

$$L(\Gamma(x)) := L(\gamma) := \sum_{i=0}^{n} \|x_i - x_{i+1}\|$$

To provide a proper distinction from the vertices of the surface, I called the vertices of a surface mesh curve its control points. Please note that there is no further meaning to that. In this definition, control points of a surface mesh curve are only allowed to lie on edges or vertices of the underlying polyhedral surface. To make sure that the connection of two adjacent control points by a straight line is also part of the polyhedral surface itself, they are forced to be part of the same face. Mathematically, this property is important for consistency.

Unfortunately, surface mesh curves may still exhibit artifacts, as many adjacent control points might be defined as part of the same triangle. These artificial cases do not facilitate the design of curves for mesh processing or other applications and, in general, cannot be handled in an efficient way by algorithms. So, I will strive for using surface mesh curves with a certain regularity. An example of such can be seen in figure 2.8.

DEFINITION 2.16: Regular Surface Mesh Curve

Let S be a polyhedral surface and γ be a surface mesh curve characterized by the control point sequence $x \in \mathcal{M}(S)^{n+1}$ with $n \in \mathbb{N}$. If γ is open, it is called regular if the following property holds.

$$\forall k \in \mathbb{N}, 1 < k \leq n: \quad \forall \triangle \in \mathcal{F}(S): \quad x_{k-1} \notin \triangle \ \vee \ x_{k+1} \notin \triangle$$

In the case that γ is a closed curve (ie. $x_1 = x_{n+1}$), γ is regular if it additionally fulfills the following boundary condition.

$$\forall \triangle \in \mathcal{F}(S): \quad x_2 \notin \triangle \ \vee \ x_n \notin \triangle$$

By forbidding three adjacent control points to lie on the same triangle, regular surface mesh curves always become parameterizable by their arc length as in the smooth case in the previous subsection. This stems from the fact that the set of discontinuities in $\|\gamma'\|$ is a null set and does not change integral values when computing the arc length. As a result, the length functional directly allows for a generalization of locally shortest geodesics to the discrete case of polyhedral surfaces. It is then clear by the triangle inequality for metrics, that a locally shortest geodesic given by a surface mesh curve is a regular surface mesh curve. This statement can be generalized to hold for arbitrary locally shortest geodesic on polyhedral surfaces as these always need to be surface mesh curves.

In analogy to the smooth case, as the underlying topology of closed surface mesh curves changes to a torus, the discretized variant of regularity also needs to forbid points at the start and end to lie on the same triangle. Also, regular surface mesh curves will never contain points that lie on the interior of a boundary edge but instead only allow for boundary vertices. This is especially useful for the discretization of geodesics in the presence of boundaries. Referring to Albrecht and Berg (1991), a discrete geodesic touching the boundary of the underlying polyhedral surface can be partitioned into interior curves with zero geodesic curvature and boundary curves whose control points are solely given by boundary vertices. Figure 2.9 picks up on the boundary example given in previous subsection and figure 2.2. It shows two discretized versions to emphasize on the previous statement.

At first, the concept of discrete surface mesh curves is much more restricted than the standard practice of using piecewise continuously differentiable curves as it was done by Polthier and Schmies (2006). Still, regular surface mesh curves can represent all geodesics, are robust when it comes to singularities, and are more efficient to handle. Should there be need for a more fine-grained curve smoothing inside single faces, a retriangulation should be triggered. Furthermore, in the context of this thesis, polyhedral surfaces are not understood to be approximations of smooth manifolds. So, it would not be consistent to allow for arbitrarily bended curves inside a single face.

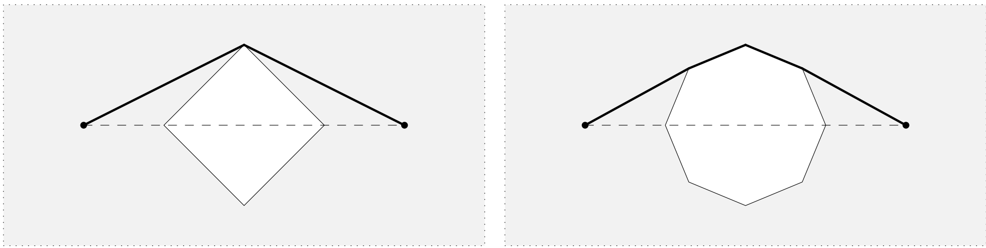


Figure 2.9: ***Discrete Geodesics on Polyhedral Surfaces with Boundary***
The images discretized schemes of the example given in figure 2.2. Here, instead of the open unit circle, regular open polygons have been removed from the Euclidean plane. These topological manifolds are representable by polyhedral surfaces and show the partitioning of a locally shortest geodesic into boundary curves with undefined geodesic curvature and interior straightest geodesics.

To make it possible to use regular surface mesh curves for the definition of discrete geodesics, I will rely on the procedure given by Polthier and Schmies (2006) to discretize the geodesic curvature of smooth curves to regular surface mesh curves.

DEFINITION 2.17: Oriented Curve Angle

Let (S, ω) be an oriented polyhedral surface and γ be a regular surface mesh curve on S. Let $p, x, q \in \mathcal{M}(S)$ be adjacent control points of γ, such that $x \in S^\circ$. Then there is $n \in \mathbb{N}_0$ and a unique sequence $(\triangle_0, \dots, \triangle_{n+1})$ of adjacent and pairwise distinct triangles with $v \in \mathcal{V}(\triangle_0) \cap \mathcal{V}(\triangle_1)$ and $w \in \mathcal{V}(\triangle_n) \cap \mathcal{V}(\triangle_{n+1})$, such that the following properties are fulfilled.

$$p, x \in \triangle_0 \qquad x, q \in \triangle_{n+1} \qquad \forall i \in \mathbb{N}, i \leq n: \quad x \in \triangle_i \ \wedge \ p, q \notin \triangle_i$$

$(\overrightarrow{xp}, \overrightarrow{xv})$ *is a positively oriented basis in* $\omega(\triangle_0)$.

$(\overrightarrow{xw}, \overrightarrow{xq})$ *is a positively oriented basis in* $\omega(\triangle_{n+1})$.

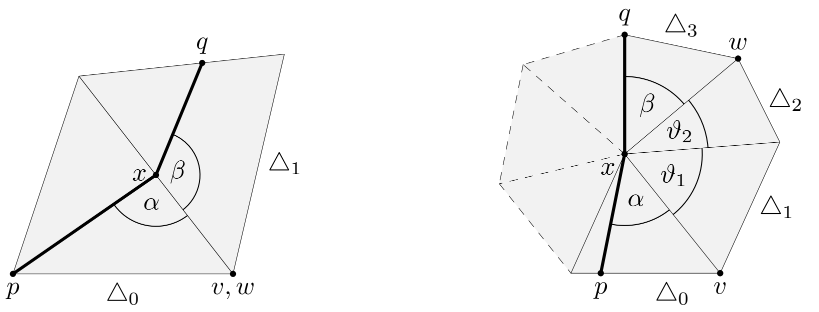
In this case, the (oriented) curve angle $\beta_S(\gamma)$ of γ at x is defined as follows.

$$\beta_S(\gamma)(x) := \sphericalangle(p, x, v) + \sphericalangle(w, x, q) + \sum_{i=1}^{n} \vartheta_{\triangle_i}(x)$$

The definition of the curve angle of a regular surface mesh curve is a little bit more technical and is schematically visualized by figure 2.10. But in essence, it is very similar to the definition of the total vertex angle and rigorously formulates how to measure an angle between two lines on a polyhedral surface. Polthier and Schmies (2006) and Lawonn et al. (2014) use the unoriented curve angle. For the oriented case, the curve angle can only be defined for curve vertices that do not lie on the boundary of S. Also the interpretation of the total vertex angle for boundary points is different. This makes a generalization of the oriented discrete curvature at boundary points difficult.

DEFINITION 2.18: Discrete Geodesic Curvature

Let (S, ω) be an oriented polyhedral surface and γ be a regular surface mesh curve on S. Then for all control points of γ that lie on the interior of γ and S, the



(a) *Edge Case:* $n = 0$, $v = w$

(b) *Vertex Case:* $n = 2$, $\vartheta_i := \vartheta_{\triangle_i}(x)$

Figure 2.10: ***Curve Angle Scheme for Vertices and Edges***
The images show schematically visualize the single components in definition of the oriented curve angle. Hereby, $\alpha := \sphericalangle(p, x, v)$ *and* $\beta := \sphericalangle(w, x, q)$.

discrete geodesic curvature is defined as follows.

$$\kappa_S(\gamma)(x) = -\frac{2\pi}{\vartheta_S(x)}\left(\frac{\vartheta_S(x)}{2} - \beta_S(\gamma)(x)\right)$$

For all points on S but its vertices, the definition of the discrete geodesic curvature simplifies to the following equation as $\vartheta_S(x) = 2\pi$ at these points.

$$\kappa_S(\gamma)(x) = \beta_S(\gamma)(x) - \pi$$

We have chosen a different sign. This definition generalizes the discrete formulation of Gauss-Bonet theorem for polyhedral surfaces. For boundaries, the discrete geodesic curvature is not easily extendable. Referring again to curves in the examples of figure 2.9, the discrete geodesic curvature itself cannot simply be extended to also characterize them as straightest geodesics. One of the possibilities would include to define the geodesic curvature of boundary curves to be zero. But this method would include other artificial cases that are not considered to be a geodesic.

With the introduction of the discrete geodesic curvature, the generalization of straightest geodesics to the discrete case of polyhedral surfaces now becomes straightforward.

DEFINITION 2.19: Discrete Straightest Geodesic

Let (S, ω) be an oriented polyhedral surface and γ be a regular surface mesh curve on S. Then γ is called a (discrete) straightest geodesic if for all control points of γ that lie in the interior of γ and S, their discrete geodesic curvature is zero.

As already noted above, the concepts of straightest and locally shortest geodesics differ on polyhedral surfaces. The main results found by Polthier and Schmies (2006) are summarized by the following proposition.

PROPOSITION 2.4: Properties of Discrete Geodesics

1. *For regular surface mesh curves that do not contain any surface vertex, the concepts of locally shortest and straightest geodesics are equivalent.*

2. *A straightest geodesic through a spherical vertex of S is not a locally shortest*

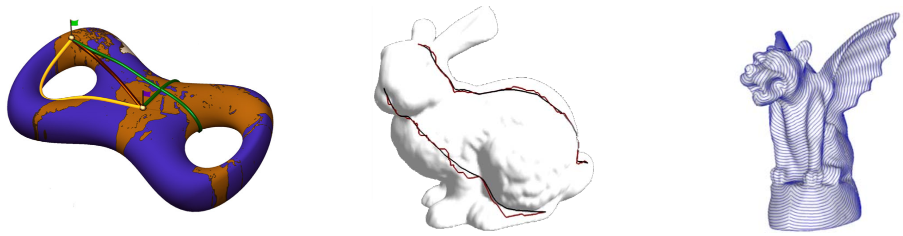
geodesic.

3. *For a given inbound direction to a hyperbolic vertex, there exists a unique straightest geodesic extending it but a family of locally shortest geodesics.*
4. *Straightest geodesics solve the initial value problem but not the boundary value problem.*

3 Related Work

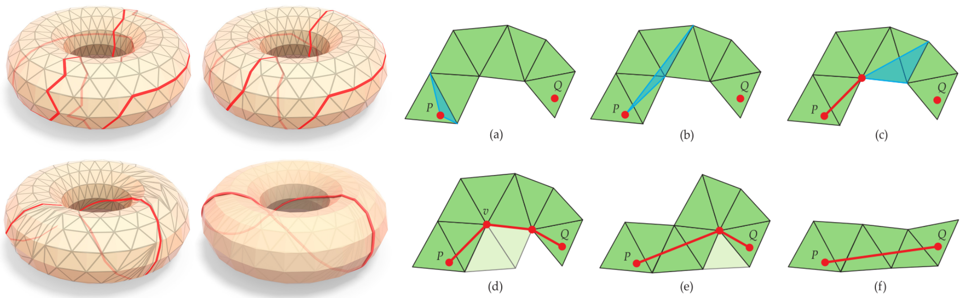
As marked in the introduction in section 1, the smoothing of curves on surface meshes is an essential operation for mesh processing and, as a consequence, for many other domain areas, like computer graphics, image-based medicine, and engineering, that rely on such tools (Ji et al. 2006; Kaplansky and Tal 2009). In most of its applications, initial curves are either provided by means of direct user interaction or by automatic or semiautomatic feature detection algorithms (Lawonn et al. 2014; Zachow et al. 2003). The finite precision of the underlying surface mesh together with all the steps included to define an initial curve usually makes resulting lines contain non-smooth artifacts which may violate given constraints or expected properties and therefore degrade its quality (Kaplansky and Tal 2009; Lawonn et al. 2014). Introducing a smoothing stage into the curve processing pipeline, the mesh segmentation is expected to be of much higher quality which greatly increases its usage for areas like machine learning (Benhabiles et al. 2011) or medicine (Alirr and Abd. Rahni 2019; Zachow et al. 2003). During the last two decades, there have been multiple successful attempts for constructing algorithms to smooth curves on surfaces (Bischoff, Weyand, and Kobbelt 2005; Hofer and Pottmann 2004; Lawonn et al. 2014; Mancinelli et al. 2022). In this section, a brief overview of their major contributions is given.

As stated in the previous section 2, a crucial tool for working with curves on two-dimensional manifolds is the ability to find and generate geodesic paths on triangle meshes in the sense of the initial and boundary value problem. The rigorous mathematical concepts and definitions for the discrete geodesics problems have been elaborated by Mitchell, Mount, and Papadimitriou (1987) and Polthier and Schmies (2006) first published in 1997. It was marked that the generalization of the concept of geodesics from smooth manifolds to polyhedral surfaces is not unique. By introducing the notion of discrete geodesic curvature, Polthier and Schmies (2006) instead fleshed out two main definitions for discrete geodesics on polyhedral surfaces, namely locally shortest geodesics and straightest geodesics. Both of these concepts coincide on smooth manifolds but differ on polyhedral surfaces resulting in varying solutions to the discrete geodesics problems. Additionally, Mitchell, Mount, and Papadimitriou (1987) built an algorithm to solve the discrete boundary value problem for locally shortest geodesics, that uses a continuous version of the algorithm of Dijkstra (1959) to find the shortest path connecting two given points. Furthermore, Polthier and Schmies (2006) provided an iterative algorithm to solve the discrete initial value problem of finding the unique straightest geodesic



(a) *Polthier and Schmies (2006)* **(b)** *Martínez, Velho, and Carvalho (2005)* **(c)** *Surazhsky et al. (2005)*

Figure 3.1: ***Examples of Discrete Geodesics***
The images show examples of the different approaches for the generation of geodesics on polyhedral surfaces described by Polthier and Schmies (2006), Martínez, Velho, and Carvalho (2005), and Surazhsky et al. (2005). All the images have been provided by the authors themselves.



(a) *Sharp and Crane (2020)* **(b)** *Mancinelli et al. (2022)*

Figure 3.2: ***State-of-the-Art Discrete Geodesic Tracing***
The left images shows the approaches for the computation and tracing of geodesics on polyhedral surfaces described by Sharp and Crane (2020). The right image shows the funnel algorithm scheme of Mancinelli et al. (2022). All the images have been provided by the authors themselves.

given a starting point and a direction for which an example can be seen in figure 3.1a. Hereby, they have proven its correctness and also introduced the notion for parallel translation of vectors along polyhedral surfaces for particle transportation and differential equations.

Based on the results of Mitchell, Mount, and Papadimitriou (1987), Kimmel and Sethian introduced the so-called fast marching approach (FMM) to efficiently approximate the shortest geodesic connecting two given points, which used the eikonal equation to build propagating fronts and therefore more efficiently generate distance fields (Kimmel and Sethian 1996, 1998; Sethian 1996). Surazhsky et al. (2005) followed with a formulation of exact and approximate algorithms to solve the discrete initial and boundary value problem, which could be evaluated efficiently by the use of such distance fields. Figure 3.1c shows an example of their work. Extending the idea of distance fields as an intermediate step to the generation of geodesics, Crane, Weischedel, and Wardetzky (2013) also used the gradient of the heat kernel to reconstruct a distance field by solving the Poisson equation and Bommes and Kobbelt (2007) generalized the algorithm of Surazhsky et al. (2005) to not only handle isolated starting points but also general starting polygons on polyhedral surfaces.

Building on the theory of Polthier and Schmies (2006), Martínez, Velho, and Carvalho (2005) provided an algorithm to solve the discrete boundary value problem for locally shortest geodesics. Using paths generated by FMM as initial curve for their algorithm, they were able to iteratively improve this curve with an optimization approach making it eventually converge to the exact solution. An example of their result has been visualized in figure 3.1b. Improving the idea of Martínez, Velho, and Carvalho (2005), Xin and Wang (2007) compared various methods to provide an initial path and presented a visibility-based algorithm to determine an exact locally shortest path on polyhedral surfaces after finitely many steps. Emphasizing this discrete nature of the problem, Sharp and Crane (2020) showed that it is possible to find exact geodesic paths for triangle meshes just by intrinsically flipping the edges of the surface mesh and constructed an algorithm, named *FlipOut*, which finds the locally shortest geodesic in the same isotopy class as the initial curve and that is guaranteed to terminate after a finite number of steps. The algorithm never needs to embed the final, flipped triangulation into the surface again and exhibits real-time performance even for millions of triangles. One can see an example of the *FlipOut* scheme in figure 3.2. To further improve the computation of locally shortest geodesics in a finite number of steps, Mancinelli et al. (2022) combined the

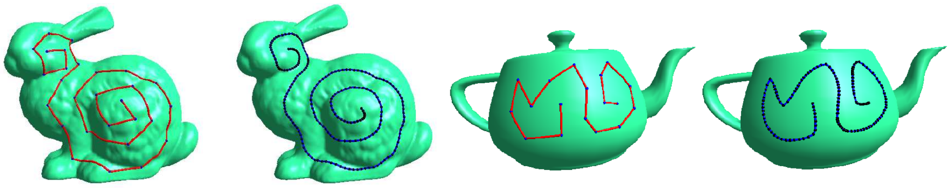


Figure 3.3: ***Examples of Subdivision Curves by Morera, Velho, and Carvalho (2008)***
The images show examples of the curve smoothing approach given by Morera, Velho, and Carvalho (2008) which used subdivision schemes. All images have been provided by the authors themselves. Red lines indicate the initially chosen curve and blue lines the resulting smoothed curve.

funnel algorithm of Lee and Preparata (1984) with the ideas of Xin and Wang (2007) into a three-stage algorithm that surpasses *FlipOut* due to its superior initial extraction of a triangle strip that results from using an optimized shortest path algorithm on the dual graph of the surface mesh. For the initial strip, the shortest path algorithm uses an A*-like distance metric and an underlying double ended queue driven by the small-label-first and large-label-last heuristics to avoid the use of a priority queue. This scheme is also illustrated in figure 3.2. Additional literature covering topics concerned with discrete geodesics on polyhedral surfaces is also well-covered by Crane et al. (2020).

For the actual creation of smooth curves on surfaces, evidence shows that only a few main approaches have emerged. Presumably, the most intuitive way for a curve smoothing algorithm to work is by using subdivision schemes for polygonal lines, also called corner cutting. The algorithm thereby subdivides each line segment and positions newly created points in such a way that the resulting curve is smoother than the previous one. First introduced and used in the planar case by Chaikin (1974) and Dyn, Levin, and Liu (1992), Morera, Velho, and Carvalho (2008) generalized the algorithm to polygonal lines on surfaces. Two examples of this can be seen in figure 3.3. Unfortunately, the subdivision curve may consist of points located anywhere inside the faces of the underlying mesh. Hence, its trajectory is not a surface curve in the strong rigorous mathematical sense and might miss essential parts of the mesh. The objective to construct a robust curve smoothing algorithm dictates that for discrete surfaces all line segments should lie inside a face of the surface.

The smoothing of curves based on features of the surface mesh for automatic mesh segmentation and cutting has been shown to successfully work by Jung and Kim (2004),

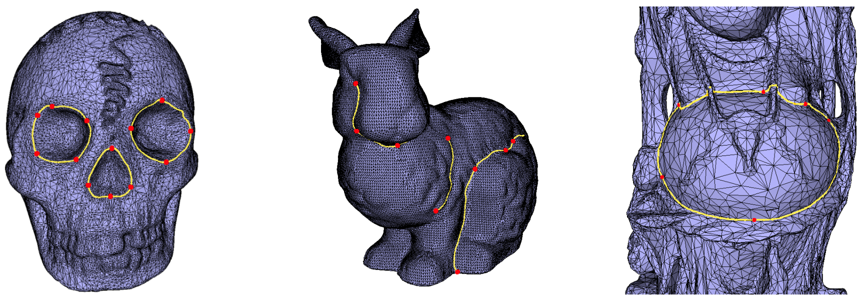


Figure 3.4: ***Examples of Snakes by Lee and Lee (2002)***
The images show examples of the curve smoothing approach given by Lee and Lee (2002) which used a generalization of snakes to detect surface features. All images have been provided by the authors themselves.

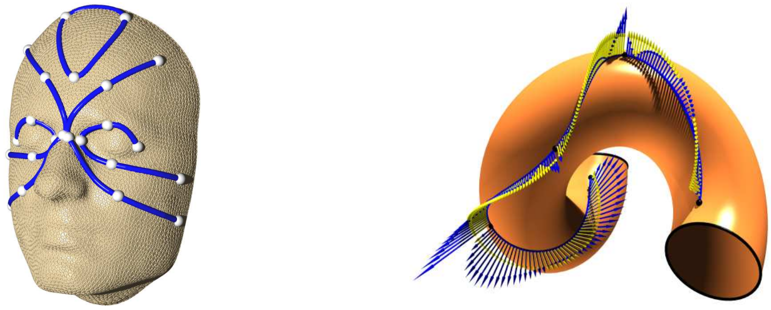


Figure 3.5: ***Examples of Energy-Minimizing Splines by Hofer and Pottmann (2004)***
The images show examples of the design of smooth curves given by Hofer and Pottmann (2004) that used a variational approach to minimize an energy-like functional on the curve. All images have been provided by the authors themselves.

Bischoff, Weyand, and Kobbelt (2005), and Lai et al. (2007). To classify surface features, Lai et al. (2007) used a feature-sensitive curve smoothing which allowed them to obtain smooth boundaries for mesh features. Both publications, Jung and Kim (2004) and Bischoff, Weyand, and Kobbelt (2005), are building upon the previous work of Lee and Lee (2002) and Lee et al. (2004). They generalized so-called snakes for two-dimensional manifolds to represent curves on surfaces that are able to find crucial mesh features by providing an initial curve. Figure 3.4 provides some examples of the results achieved by Lee and Lee (2002). First introduced by Kass, Witkin, and Terzopoulos (1988) for two-dimensional images, snakes are closed curves that evolve over many iterations to the features of the mesh by minimizing internal and external forces based on curvature, length, and distance to features. For snakes, the initial shape is completely unimportant. They are allowed to merge or split, such that a rapid movement towards the features of the mesh is to be expected. As a direct consequence, feature-based curve smoothing does not allow for arbitrary trajectories and would lead to a curve that may not be assumed to be near the original curve, which makes these algorithms bad candidates for general curve smoothing.

Another approach for representing and generating smooth curves on surfaces is through the use of splines. Hofer and Pottmann (2004) and Pottmann and Hofer (2005) determined splines in general manifolds by addressing the design of curves as an optimization problem in the sense of minimizing the curve's overall quadratic energy and using a variational approach to compute a solution. Their approach is not only applicable to curves on surface meshes but can be used for a much broader variety of cases, including for example the design of rigid body motions. Some of their results are shown in figure 3.5. Alas, for a small number of control points, the resulting curve may still not be assumed to exhibit a close distance to the initially selected curve. Overcoming this issue would involve adding many more control points and, eventually, a much higher burden for the user who would need to define those points. According to Mancinelli et al. (2022), the variational approach to solve the optimization problem for the design of curves is also expected to provide a poor performance for surface meshes that consist of millions of triangles.

These results quickly lead to the representation of smooth curves by using generalized Bézier splines. Two essential contributions are given by Martínez, Carvalho, and Velho (2007) and Mancinelli et al. (2022). They lift the concept of Bézier curves in the two-dimensional Euclidean space to geodesic Bézier splines located in the surface. The initially chosen

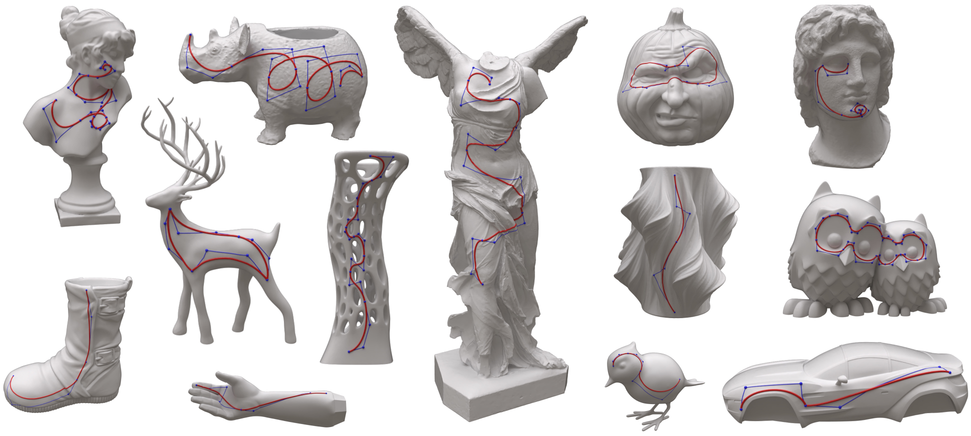


Figure 3.6: ***Examples of Bézier Splines by Mancinelli et al. (2022)***
The images show examples of the design of smooth curves given by Mancinelli et al. (2022) that used generalized Bézier splines to represent a surface curve. Blue lines indicate the initially chosen curve and red lines the resulting spline-based curve. All images have been provided by the authors themselves.

curve samples are thereby used as control points to determine the individual shapes of the Bézier splines. Mancinelli et al. (2022) successfully showed their approach to be superior to other spline alternatives and provided real-time performance when tracing the trajectories of the given splines on the surface for even high-resolution meshes. In addition, they offer a robust implementation of their algorithm in an open-source C++ framework, named *Yocto/GL* (Pellacini, Nazzaro, and Carra 2019), that can be found on GitHub[3]. The framework is a rather large codebase and a specific documentation for the code responsible for generating, tracing, and controlling Bézier splines could not be found, though. A whole gallery of examples that has been provided by Mancinelli et al. (2022) can be seen in figure 3.6. Nevertheless, their approach exhibits similar issues compared to the variational spline approach in that only the use of many control points will make sure that the resulting curve will be near to the initially defined curve. In this sense, the approach of Mancinelli et al. (2022) does not fit our need by being specialized for vector graphics on surface meshes.

Generalizing on the iterative algorithm of Martínez, Velho, and Carvalho (2005), Lawonn et al. (2014) created an algorithm for curve smoothing based on curvature values given for each trajectory point. Each iteration, the algorithm tries to locally fulfill the given curvature constraint, eventually converging to its final smooth curve. The algorithm guarantees a close distance to the initial curve and can also be used with a simplified user interaction where only one parameter has to be adjusted. During the process, all iterations adapt to the resolution of the underlying surface mesh and directly provide the curve on the surface without the need of tracing or projection. The algorithm was shown to be robust against geometric and parametric noise and applied in a medical context, where domain experts evaluated its usability. Lawonn et al. (2014) proved the convergence of the algorithm and also compared the quality of the generated smoothed curves against the spline-based variational approach without any issue. Figure 3.7 shows two examples for this algorithm that visualizes medicine-specific models in the context of mesh cutting. Furthermore, no surface normals or curvature, that would

[3] *Yocto/GL* (2023). *Tiny C++ Libraries for Data-Oriented Physically-based Graphics*. URL: https://github.com/xelatihy/yocto-gl (visited on 11/19/2022).

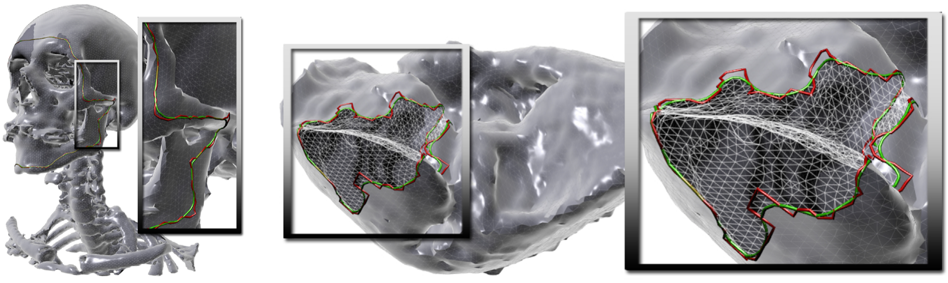


Figure 3.7: ***Examples of Curve Smoothing by Lawonn et al. (2014)***
The images show examples of the curve smoothing algorithm given by Lawonn et al. (2014) that uses a generalization of the algorithm given by Martínez, Velho, and Carvalho (2005) to fit curvature values. Red lines indicate the initially chosen curve and green lines the resulting smoothed curve. All images have been provided by the authors themselves.

need to be evaluated first, is needed for the algorithm. As a result, it may also be formulated for generalized two-dimensional triangular manifolds leaving a wide variety of mesh data structures to choose from (Guibas and Stolfi 1985). Unfortunately, Lawonn et al. (2014) do not provide any language-specific implementation or performance evaluation.

According to the explanations and descriptions above, for the purpose of this thesis, our design and implementation will mainly focus on the approach given by Lawonn et al. (2014). Their algorithm seems to fit our needs in nearly all important aspects. To solve the potential performance issue and get real-time behavior, a parallelization on the CPU and GPU will be carried out. We will also strive for an optimized curve initialization and geodesics generation that builds upon the basic building blocks of the approach given by Mancinelli et al. (2022).

4 Design and Implementation

The C++ code shown in this section does not provide the best design.

4.1 Polyhedral Surface Data Structure

In general, the choice and efficiency of algorithms highly depends on underlying data structures that represent the intermediate data of a program (Knuth 1997; Mehlhorn and Sanders 2008; Smed and Hakonen 2006). Thus, thorough design considerations for the data structures of polyhedral surfaces and surface mesh curves are mandatory to build fast and robust curve smoothing algorithms.

To be able to render a triangular surface mesh in OpenGL, a simple and versatile method is to provide two separate lists for vertices and triangles, respectively (*OpenGL* 2023). Hereby, each triangle only references its three vertices by using indices that may be used to extract a vertex from the vertex list. The orientation of a triangle is defined by the order of its vertex references based on the description given in section 2 in the preliminaries. Furthermore, vertices in OpenGL are quite general and may not only store their own positions. Typically, attributes for pseudo-normals, colors, or texture coordinates are added to use sophisticated shading techniques and improve the quality of illustrations. A concrete implementation of this can be seen in the following code snippet. Additionally, figure 4.1 shows an example to visualize the memory layout of the vertex and faces list.

Code 4.1: Vertices and Faces

```cpp
// Vertices store their position and
// a pseudo-normal to model additional attributes
// and improve rendering quality.
//
struct vertex {
  vec3 position{};
  vec3 normal{};
};

// In OpenGL and other standard applications,
// it is sufficient to use 32-bit unsigned integers
// to reference vertices.
//
using vertex_id = uint32_t;

// Each face is a contiguous array of three vertex indices.
//
struct face : array<vertex_id, 3> {};

// The vertex and face list are contiguous arrays
// which are dynamically allocated and managed on the heap.
//
vector<vertex> vertices{};
vector<face> faces{};
```

As polyhedral surfaces are a special form of 3D surface geometry, I will use file formats,

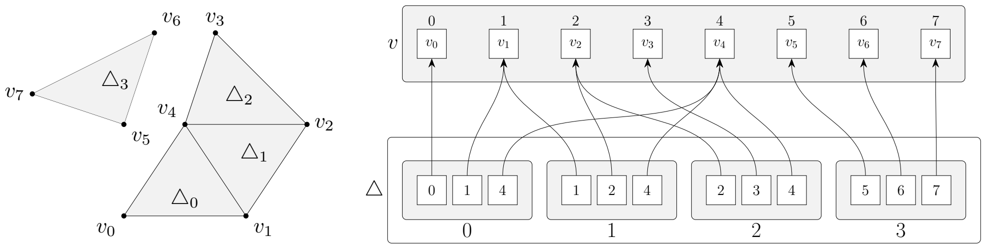


Figure 4.1: ***Data Structure for Faces of a Polyhedral Surface***
The figure shows a schematic example of how faces of a given polyhedral surface (left) are stored in memory. The surface consists of eight enumerated vertices and four faces. As it can be seen on the right, each face references three distinct vertices by their index in the order of their orientation. This way one vertex can be used by multiple triangles.

such as STL (Congress n.d.) and Wavefront OBJ (Bourke n.d.), to provide the possibility for reading complex surface meshes from disk. Especially in the area of computer graphics in conjunction with the C++ programming language, the *Assimp* library (Kulling 2021) is often used to load even more file formats that represent general surfaces and scenes. The facility makes it possible to test algorithms with very complex surface meshes from the “Thingi10K” dataset whose models are mainly provided as STL files.

This representation already allows for an OpenGL-based visualization of polyhedral surfaces with sufficient quality by adding GLSL shaders. But in the context of this thesis, also the efficient movement of points and curves along the surface mesh needs to be possible. Over and over trying to determine adjacent neighbors of vertices or triangles with this data structure quickly becomes infeasible. So, further surface mesh preprocessing steps are needed to generate and store essential adjacency information.

Computing and storing neighbor and adjacency information of surface meshes is a studied and vast area of computational geometry. There are multiple well-known data structures, like Quad-Edge Algebras (Guibas and Stolfi 1985) and triangular neighbors (Shewchuk 1996), for handling graphs that represent the topological structure of general 2D polyhedral surfaces. These data structures have successfully been used to efficiently construct Delaunay triangulations and encode the information of 3D surface meshes.

Unfortunately, the connectivity information provided by the faces and vertices are not suitable for surface mesh curves and smoothing algorithms. In an OpenGL-based rendering pipeline, the vertices of a polyhedral surface must differ as soon as they exhibit at least one distinct attribute. Even though triangles could share the same vertex position, their respective vertices might contain different pseudo-normals to model a crease in the surface and, consequently, these triangles cannot reference the same vertex. Even worse, for various reasons, it is allowed to use multiple instances of the same vertex inside the vertex list which also would lead to distinct vertex references. Especially, in the extreme case of STL files, every triangle uniquely refers to vertices that cannot be referred by other triangles which is schematically visualized in figure 4.2. On the other hand, the values for additional attributes of a vertex, like pseudo-normals or texture coordinates, are unimportant for algorithms handling surface mesh curves and may be neglected as they do not change the topology of the underlying polyhedral surface. Such an algorithm would need to assume two vertices to be equivalent as long as they exhibit the same position. As a consequence, naively generating the graph or dual graph of the surface mesh solely based on the vertices and faces will not result in the correct

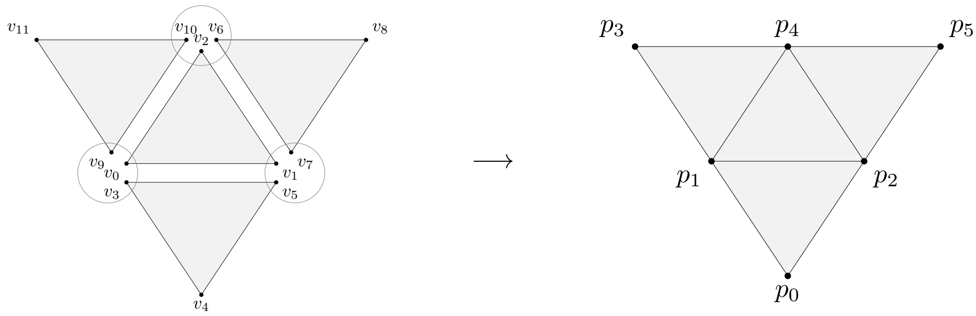


Figure 4.2: ***Connectivity by Using Topological Vertices***
On the right, a simple polyhedral surface with six topological vertices and four faces is shown. In a data structure suitable for OpenGL-based rendering, each triangle might reference its own unique vertices as shown on the left. These vertices are distinct from each other which logically disconnects the four adjacent triangles.

topological connectivity needed for surface mesh curves.

The use of surface mesh curves and smoothing algorithms should neither put a higher burden on the OpenGL pipeline nor require dramatic changes to data structures used for rendering. My solution to that problem generates an additional topological structure that extends the connectivity information given by the basic structure. Hereby, the main idea was that the projection of vertices onto their positions can be modeled by the quotient space and its respective quotient map. In this sense, let $\mathcal{V}$ be the set of all vertices and $p\colon \mathcal{V} \to \mathbb{R}^3$ be the function that maps a vertex to its position in space. Two vertices $v, w \in \mathcal{V}$ are called to be topologically equivalent, denoted as $v \sim w$, if they exhibit the same position.

$$v \sim w \quad :\Longleftrightarrow \quad p(v) = p(w)$$

Then the quotient space $\mathcal{V}/\sim$ consists of equivalence classes whose elements are pairwise topologically equivalent vertices. These equivalence classes will from now on be called topological vertices. The respective quotient map τ of $\mathcal{V}/\sim$ is given as follows.

$$\tau\colon \mathcal{V} \to \mathcal{V}/\sim \qquad \tau(v) := [v]$$

Two triangles are topologically adjacent to each other if they share a topological edge and are not identical. Using the quotient map, a rigorous formulation can be given as follows. Let $n \in \mathbb{N}$ be the number of vertices in the vertex list $V : \mathcal{I} \to \mathcal{V}$ with $\mathcal{I} := \{k \in \mathbb{N}_0 \mid k \leq n\}$ as the index set. Two triangles of the faces list given by index triples (v_1, v_2, v_3) and (w_1, w_2, w_3) are adjacent to each other if there are permutations $\sigma, \pi \in \mathrm{S}_3$ such that the following holds.

$$\tau(V(v_{\sigma(1)})) = \tau(V(w_{\pi(1)})) \qquad \tau(V(v_{\sigma(2)})) = \tau(V(w_{\pi(2)}))$$

$$\tau(V(v_{\sigma(3)})) \neq \tau(V(w_{\pi(3)}))$$

Hence, by constructing the quotient map τ, I will be able to correctly determine topological connections. Still, the naive evaluation of the topological equivalence relation $\sim$ for all vertices is quadratic in complexity. For high-resolution polyhedral surfaces with millions of triangles, this is infeasible and needs to be addressed.

My solution is based on a hash map in which all vertices will be inserted with respect to their position in space. As 3D vectors do not provide a natural order, a custom hash

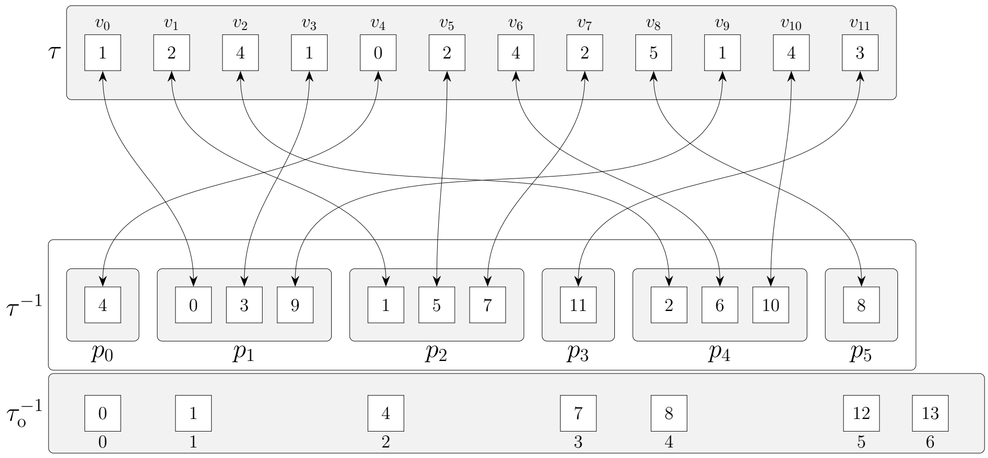


Figure 4.3: ***Topological Quotient Map***
The figure schematically shows the quotient map τ of the polyhedral surface given in figure 4.2 which maps vertices onto their topological counterparts. As this map is not bijective, its inverse τ^{-1} must use another array τ_o^{-1} of offsets.

function needs to be used. Furthermore, the standard equality comparison is exchanged with the topological equivalence relation. This way topologically equivalent vertices are assumed to be the same by the hash map and are inserted at identical positions. Each insertion has an expected constant-time complexity. As a result, the insertion of all vertices exhibits a linear time complexity and offers a more efficient alternative to the naive method of checking pairwise equivalences. The following code snippet demonstrates a simple implementation of this method in C++. Additionally, figure 4.3 schematically visualizes how the quotient map for the example given in figure 4.2 would look.

Code 4.2: Topological Vertex Map

```
// Store the values of the quotient map for each vertex
// in a dynamically allocated contiguous array.
//
vector<vertex_id> topological_vertex_index{};
vertex_id topological_vertex_count = 0;

void generate_topological_vertex_map() {
  // Define whether two vertex references
  // are topologically equivalent.
  //
  const auto equivalent = [&](vertex_id vid1, vertex_id vid2) {
    return vertices[vid1].position == vertices[vid2].position;
  };
  using equivalence = decltype(equivalent);

  // For the most efficient pairwise comparison,
  // we use a hash map and need to define
  // a custom hash function for vertex references.
  //
  const auto hash = [&](vertex_id vid) -> size_t {
```

```
    const auto v = vertices[vid].position;
    return (size_t(bit_cast<uint32_t>(v.x)) << 11) ^
           (size_t(bit_cast<uint32_t>(v.y)) << 5) ^
           size_t(bit_cast<uint32_t>(v.z));
  };
  using hasher = decltype(hash);

  // The hash map itself is only needed during the construction
  // and directly represents the quotient map.
  //
  unordered_map<vertex_id, vertex_id, hasher, equivalence> //
      indices({},                                           //
              forward<decltype(hash)>(hash),                //
              forward<decltype(equal)>(equal));
  indices.reserve(vertices.size());

  topological_vertex_index.resize(vertices.size());
  topological_vertex_count = 0;

  // Iterate over all vertices and add them to the hash map.
  // The first-time insertion will also assign the index.
  // Toplogically equivalent vertices are automatically mapped
  // to the same index and filtered in linear time.
  //
  for (vertex_id vid = 0; vid < vertices.size(); ++vid) {
    const auto it = indices.find(vid);
    if (it != end(indices))
      topological_vertex_index[vid] = it->second;
    else {
      topological_vertex_index[vid] = topological_vertex_count;
      indices.emplace(vid, topological_vertex_count++);
    }
  }
}
```

With the above implementation, the inverse of the quotient map cannot directly be evaluated. This is a major drawback when working with edge-based data structures as these must reference topological vertices and there would be no efficient way to determine their position or shared pseudo-normals. If it is sufficient to access the positions of topological vertices then a simple copy of the positions might be the best way. In all other cases, the following code snippet provides a way to construct the inverse of the quotient map. Again, figure 4.3 visualizes the inverse of the example given in figure 4.2. As the quotient map is not bijective, an additional offset array is used to get the list of all vertices that refer to the same topological vertex.

Code 4.3: Inverse Topological Vertex Map

```
vector<vertex_id> inverse_topological_vertex_offset{};
vector<vertex_id> inverse_topological_vertex_index{};

void generate_inverse_toplogical_vertex_map() {
  const auto &labels = topological_vertex_index;
```

```cpp
  const auto count = topological_vertex_count;
  auto &inverse_offset = inverse_topological_vertex_offset;
  auto &inverse = inverse_topological_vertex_index;

  // Get the count of elements per equivalence class.
  //
  inverse_offset.assign(count + 1, 0);
  for (auto y : labels)
    ++inverse_offset[y + 1];

  // Calculate cumulative sum of counts to get the offsets.
  //
  for (size_t i = 2; i < inverse_offset.size(); ++i)
    inverse_offset[i] += inverse_offset[i - 1];

  // Generate equivalence classes by using offsets as indices.
  //
  inverse.resize(labels.size());
  for (size_t x = 0; x < labels.size(); ++x)
    inverse[inverse_offset[labels[x]]++] = x;

  // Repair the inverse offsets after using them in the previous step.
  //
  for (size_t i = inverse_offset.size() - 1; i > 0; --i)
    inverse_offset[i] = inverse_offset[i - 1];
  inverse_offset[0] = 0;
}
```

With the availability of topological vertices and their correspondence to vertices, an important intermediate construction step is the generation of all surface edges. In a following step, a hash map of directed edges will be constructed with a custom hash function for edges by iterating over the list of faces. Hereby, each triangle will add three directed edges. The values for each directed edge in the hash map will refer to its specific triangle and location inside that triangle.

Code 4.4: Intermediate Topological Edges

```cpp
using face_id = uint32_t;

// Face Adjacencies not only store a face reference
// but also the location to determine at which side
// of the triangle the adjacent edge or face lies.
// As there will never be more than 2^30 faces,
// two bits of the face id will be used to store
// the location that is only allowed to contain the
// values 0, 1, and 2.
// The value 4 is used as invalidation mark for boundaries.
//
struct face_adjacency_id {
  constexpr bool valid() const noexcept { return loc != 4; }
  face_id fid : 30 {};
  face_id loc : 2 = 4;
};
```

```
// A directed edge only references
// two topological vertices in a specific order.
//
struct edge : array<vertex_id, 2> {
  // To generate intermediate directed edges in a hash map,
  // a structure to map values onto needs to be provided.
  // Each info structure store the face adjacencey
  // of the face that generated the edge.
  // Not forbidding unoriented surfaces, at most two
  // triangles are allowed to provide the same directed edge.
  // In all other case, the surface would no longer
  // be a 2D manifold and an exception is thrown.
  //
  struct info {
    constexpr bool oriented() const noexcept { //
      return data[1].valid() == invalid;
    }
    void add_face(face_adjacency_id nid) {
      if (!data[0].valid())
        data[0] = nid;
      else if (!data[1].valid())
        data[1] = nid;
      else
        throw runtime_error(                        //
            "Failed to add face adjacency to edge. " //
            "Additional face would violate violate " //
            "the requirements for a 2D manifold.");
    }
    face_adjacency_id data[2];
  };

  // As before, a custom hash function needs to be provided
  // to be able to add edges to a hash map.
  //
  struct hasher {
    auto operator()(const edge &e) const noexcept -> size_t {
      return (size_t(e[0]) << 7) ^ size_t(e[1]);
    }
  };
};

// The hash map that contains an intermediate
// representation of all directed edges.
// After the generation of face adjacencies,
// it is no longer needed and can be destroyed.
//
unordered_map<edge, edge::info, edge::hasher> topological_edges{};

// Iterate through all oriented faces and
// add their directed edges with their
// respective locations to the hash map.
//
void generate_topological_edges() {
  topological_edges.clear();
  for (face_id fid = 0; fid < faces.size(); ++fid) {
```

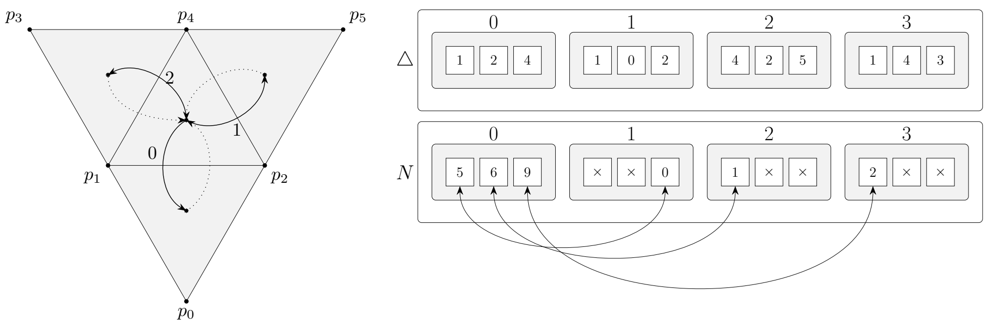


Figure 4.4: ***Topological Face Adjacencies***
The figure schematically visualizes how face adjacencies are stored in memory. On the left, the inner face of the polyhedral surface is surrounded by three other faces. For all these faces, its adjacency structure provides the face reference and the location inside their adjacency structure that points back to the inner face. In this example, the face adjacency is given by $N = 3f + l$ with f as the face reference and l as the location. Please note, in the code provided in this section, instead $N = 4f + l$ in the form of a bit-field is used for efficiency reasons.

```
      const auto &f = faces[fid];
      topological_edges[edge{topological_vertex_index[f[0]], //
                             topological_vertex_index[f[1]]}]
          .add_face({fid, 0});
      topological_edges[edge{topological_vertex_index[f[1]], //
                             topological_vertex_index[f[2]]}]
          .add_face({fid, 1});
      topological_edges[edge{topological_vertex_index[f[2]], //
                             topological_vertex_index[f[0]]}]
          .add_face({fid, 2});
    }
  }
```

Regarding the implementation of surface mesh curves, I will use a flat adjacency list to store references to adjacent triangles of a given triangle in accordance to Mancinelli et al. (2022). As each triangle may at most exhibit three other triangles that are adjacent to it, there is an upper storage bound for neighboring references of each triangle. In the case that a triangle is part of the boundary of the polyhedral surface, a non-existing neighbor is marked as an invalid reference. This representation also comes in handy for the implementation of parallel smoothing algorithms on the GPU. To generate the topological face adjacencies, all that is to do is to basically iterate over all edges and add their mapped information to the face adjacencies. The following code snippet demonstrates this. In figure 4.4, a scheme that visualizes the memory layout of the topological face adjacencies can be seen.

Code 4.5: Topological Face Adjacencies

```
// A triangle at most can have three adjacent faces.
// So, a simple static array with three values
// will be used to represent face adjacencies.
//
```

```
struct face_adjacency : array<face_adjacency_id, 3> {};

// All the face adjacencies are stored in a contiguous array
// that is dynamically allocated and managed on the heap.
//
vector<face_adjacency> topological_face_adjacencies{};

// Generating the actual face adjacencies then only
// consists of iterating over all directed edges
// and copying their mapped information.
//
void generate_topological_face_adjacencies() {
  const auto& edges = topological_edges;
  auto& adjacencies = topological_face_adjacencies;
  adjacencies.assign(faces.size(), face_adjacency_id{});

  for (const auto& [e, info] : topological_edges) {
    if (!info.data[1].valid()) {
      // In this case, the edge is oriented
      // as only a single triangle provides it.
      // So, the reverse edge also needs to be evaluated.
      const auto it = edges.find(edge{e[1], e[0]});
      // If it does not exist, it is a boundary edge.
      if (it == end(edges)) continue;
      const auto& [e2, info2] = *it;
      adjacencies[info.data[0].fid][info.data[0].loc] = info2.data[0];
    } else {
      // In this case, the directed edge is unoriented
      // as two triangles provide the same directed edge.
      adjacencies[info.data[0].fid][info.data[0].loc] = info.data[1];
      adjacencies[info.data[1].fid][info.data[1].loc] = info.data[0];
    }
  }
}
```

Incorporating the location values into face adjacencies allows for an easy navigation along oriented polyhedral surfaces. Figure 4.5 shows the basic operation for face adjacencies that rotates the current direction of the adjacency counterclockwise. It can be implemented by adding 1 to the location and afterwards taking its modulus base 3. This primitive enables us to easily run through the list of triangles around a vertex in clockwise and counterclockwise order.

!!!!To only generate vertex neighbors is also possible but not as versatile as one can think.!!!!

As already described in section 2 in the preliminaries, the vertices and edges of a polyhedral surface model a graph which characterizes the vertex adjacencies. Looking at a polyhedral surface in this way, storing the triangle adjacencies makes the corresponding dual graph available. Using the vertex-based graph, the boundaries of a surface mesh need to be handled by additional facilities as the graph itself does not exhibit a way to encode them. On the other hand, its dual graph copes with boundaries by marking references to be invalid. Furthermore, adding orientation to the vertex-based graph can be cumbersome whereas the orientation for the triangle-based dual graph is given by orientation of the triangle itself.

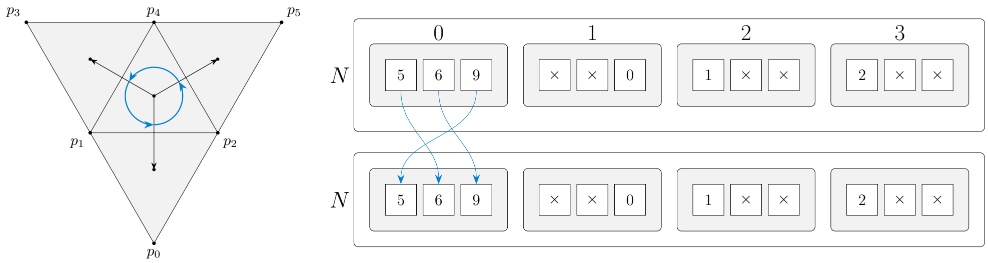


Figure 4.5: ***Counterclockwise Rotation of Topological Face Adjacencies***
The rotation of face adjacencies (shown in blue) can be implemented by adding 1 *to the location and then taking its modulus base* 3*. Using this operation together with all the face adjacencies allows to iterate through the list of faces around a vertex in clockwise and counterclockwise order.*

Experimenting with various designs and implementations of algorithms, that robustly and efficiently smooth discrete curves on surface meshes, requires us to use a whole program pipeline or framework for loading input data, intuitively working with user interaction, and visualizing any intermediate results. As changing specific parts of such a pipeline may affect the performance, outcomes, and overall behavior of an algorithm or program, a brief overview over all of its components is essential to allow for reproducible results and will therefore be given in the following subsections. This will also make it possible for readers to simply reconstruct, adjust, and improve the pipeline for their own domain-specific projects. Afterwards, I will thoroughly elaborate on the design and implementation of chosen data structures and algorithms together with their mathematical primitives for curve smoothing.

The pipeline or framework, described in this thesis, has been manually implemented with the C++ programming language and the OpenGL graphics API using some external libraries to handle low-level tasks. Already described in the introduction in section 1, this choice is well suited for open-source graphics applications and provides programmers with a large freedom when it comes to the implementation of data structures and algorithms. The complete source code of the project, called *nanoreflex*, is provided as an open-source repository on GitHub.

4.2 Overview of the Program Pipeline

In the following, the stages of the implemented program pipeline are stated and summarized. These stages naturally arise from an insight into the generation and tracing of geodesics on surface meshes, described in section 3. Locally shortest or straightest geodesics that follow as a solution from the discrete boundary value problem are in some sense the smoothest curves that cannot be made any smoother. As we typically need to provide an initial curve for those geodesics to be found, the whole generation and tracing of geodesics can be looked at as an extreme smoothing process. Adding parameters to let the user decide on when to stop this process then results in the pipeline given below and schematically shown in figure 4.6.

INSERT YOUR IMAGE HERE!

Figure 4.6: ***Program Pipeline Stages***
The scheme shows all the stages of the implemented program pipeline. The arrows indicate data flow and dependencies.

1. **Surface Mesh Loading:**
   Load a surface mesh given by a specific file format from the storage.
2. **Surface Mesh Preprocessing:**
   (a) Generate a connected mesh object to properly model the topology of the input.
   (b) Generate the pseudo-normals for each vertex.
   (c) Generate the dual graph for the neighbors of each triangle.
   (d) Check that the mesh is a valid orientable two-dimensional topological manifold.
3. **Initial Curve Selection:**
   Let the user choose an initial curve by drawing on the surface.
4. **Parameter Selection:**
   Let the user choose parameters for the curve smoothing algorithm.
5. **Curve Smoothing:**
   Smooth the initial curve according to the constraints given by the selected parameters.
6. **Postprocessing:**
   Optionally apply the smoothed curve in a domain-specific context.

According to figure 4.6, the output of every stage is not only forwarded to the next stage but also specifically visualized and rendered to the screen to provide the user with visual feedback and to catch errors or exceptional behavior as early as possible. Besides, additional user interactions will be processed for all stages to allow for measurements and adjustments and to be able to react to the output of previous stages. The loading of surface meshes from different file formats is typically handled by loader libraries, such as *Assimp*, and will not be further explained. On the other hand, surface mesh preprocessing and the initial curve selection are crucial steps in the whole pipeline which remove geometric degeneracies and artifacts that would otherwise violate assertions needed for the correct execution of the algorithm. Therefore both stages will be discussed in more detail in the following subsections. The postprocessing in our case is optional and only provided for the sake of completeness. The parameter selection is highly dependent on the implementation of the curve smoothing algorithm and, as a result, it will be described together with the curve smoothing itself.

4.3 Surface Mesh Preprocessing

The goal of preprocessing a surface mesh which has been loaded from disk is to transform its data into a valid and efficient representation of a polyhedral surface that should exhibit functionality to draw and move points and discrete curves on its surface. In the case that this transformation is not possible, the stage at least must check for the validity of the given data and emit an error if there might be any violations.

A given Mesh for the algorithm can be quite general. Providing triangles which will introduce numerical difficulties is out of the scope of this thesis.

We will focus on orientable meshes. Please note, that is not an actual restriction for the algorithm but will only speed up the implementation. Furthermore, we look at surfaces which are the boundary of volumes in three-dimensional Euclidean space which can be looked at

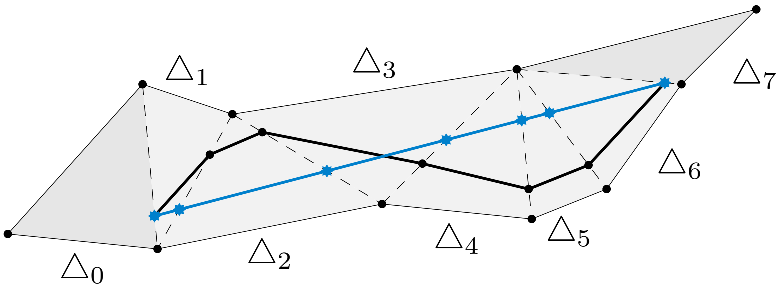
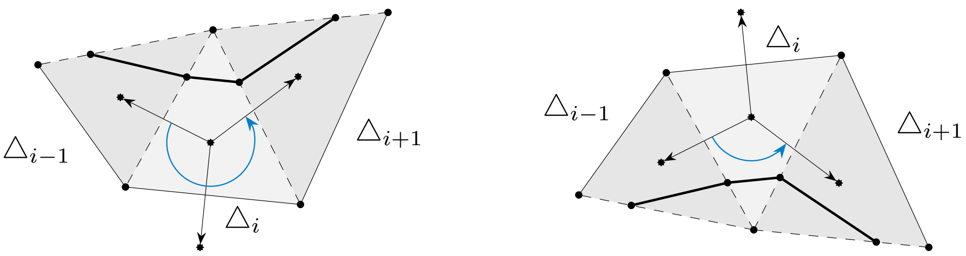


Figure 4.7: ***Surface Mesh Curve***



(a) *Left*

(b) *Right*

Figure 4.8: ***Right and Left Turn Scheme***

as open submanifolds. These volumes are therefore by definition oriented and, hence, their boundary needs to be, too. Still, we need to look boundaries for surfaces as, for example, distance envelopes restrict the surface without boundary to be one. The boundaries need to be handled properly. Also, these requirements are only need to hold for the restriction to the distance envelope. Often, lines will be drawn on oriented/orientable parts of the surface even if the surface is not orientable due to artifacts originating from scanning or generating meshes.

Topological Connections We need the topological connections of a given surface mesh. In the case of general file formats, a scene is typically separated into multiple meshes which are topologically but not smoothly connected. For proper drawing of curves in a whole scene, the topological structure of connections needs to be generated first. Using the STL file format, this is not needed.

Generating the dual graph To generate such a dual graph we refer to this book. It is assumed to be a solved problem. First, we have created a hash map of oriented edges with a custom hash function to mangle vertex indices. The mapped values store the faces indices and their location in the triangle. By inverting the indices of a stored edge, one can easily access the neighboring triangle if it exists. Furthermore, it is easy to check whether the mesh is oriented and a valid two-dimensional manifold.

Check for orientability and validity

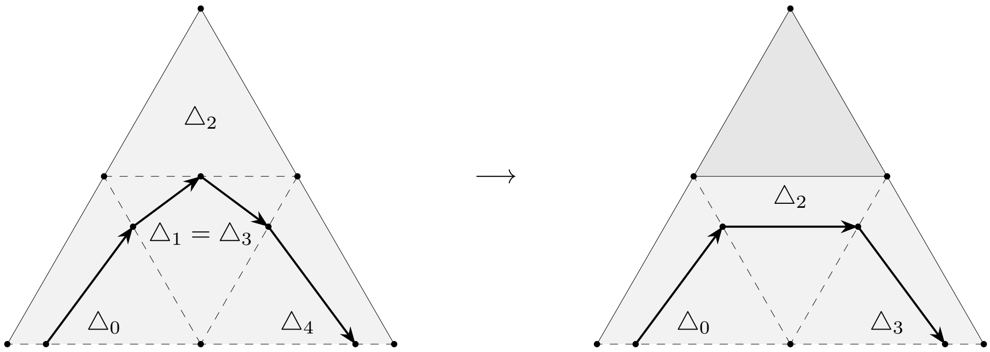


Figure 4.9: ***Regularization of Irregular Triangle Paths***

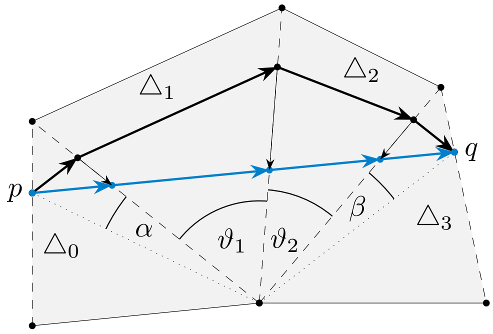


Figure 4.10: ***Unfolding Critical Vertex***

4.4 Discrete Surface Curve Data Structure

There two main attempts for the construction of a data structure representing surface mesh curves. One of them is based on edges and the other on triangles.

We do not look at polyhedral surfaces as an approximation of a real-world smooth manifold.

Advantages of discrete surface mesh curves over more general curves on polyhedral surfaces

- Curve is provided in triangle precision. Better Compression. More points would be waste of memory.
- Robust: Snap to vertex if points get arbitrary near to each other
- curve is always a surface curve of the polyhedral surface and not violating mathematical properties. This also allows for direct rendering without projection.
- Useful for mesh processing
- Interpretation of Discretization is more consistent: surface mesh may be discretized approximationof real-world manifold with the mesh to be the finite grid. then it is only natural to discretize a real world surface curve in the same way.

Advantages of the triangular data structure over the edge-based one:

- Triangles can be easier generalized to higher dimensions than edges.
- Triangles do not need to handle boundaries in complicated ways.

- Triangles allow easy and ordered access to fan around vertex.
- Artifact removal routine for triangles is simpler than for edges.
- Triangles do not exhibit data loss when handling reflex vertices
- Triangles allow for a more general moving of points

4.5 Initial Curve Selection

As for the vast majority of optimization algorithms, the results and efficient working are highly dependent on the chosen starting values. Curve smoothing itself is an optimization process and to solve the boundary value problem, we need an initial value. So, choosing the initial curve in the right way will also heavily change the speed and quality of the algorithm. The algorithm should be able to handle a vast amount unsmooth curves. Still, the handling of artifacts will be taken care of at the start.

User Interface for Selecting and Controlling Initial Curves

Drawing by Ray Tracing

Connecting the Vertices

Closed Initial Curves and Fixed Vertices

Artifact Removal

Smoothed Curvature Values by Stencil Stencil is discrete approximation of solution to Laplacian equation or heat equation which is smooth (infinitely differentiable). Maybe it would be useful to keep the length of the curve and still smooth it.

Vertex Curves

Face Curves

Tracing Geodesics

4.6 Unfolding

To unfold two triangles along their common edge and find the shortest line connecting two points, we will first look back into two dimensions.

Let $p, q \in \mathbb{R}^2$ such that $p_x < q_x$.

$$\Delta x = q_x - p_x \qquad \Delta y = q_y - p_y \qquad \Sigma x = p_x + q_x \qquad \Sigma y = p_y + q_y$$

The straight line function connecting these two points then looks like the following.

$$f(x) = \frac{q_y - p_y}{q_x - p_x}(x - p_x) + p_y = \frac{q_y - p_y}{q_x - p_x}x + \frac{p_y q_x - q_y p_x}{q_x - p_x}$$

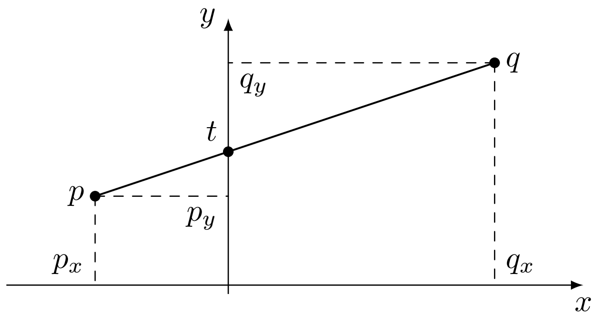
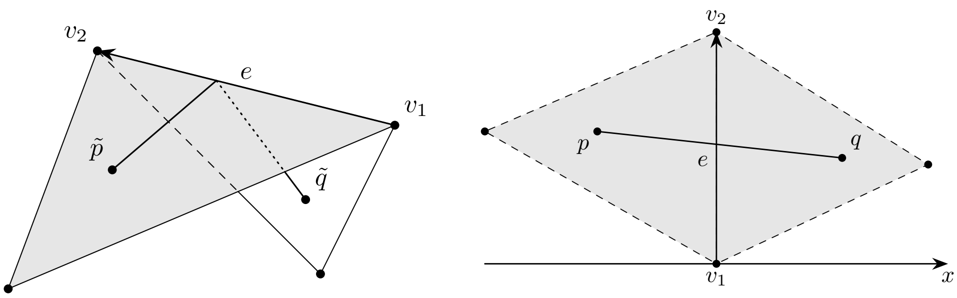


Figure 4.11: ***2D Unfolding Primitive***

Please note that this function is well-defined, as $q_x - p_x > 0$. To get the intersection point with the ordinate, set the argument to zero.

$$t \coloneqq f(0) = \frac{p_y q_x - q_y p_x}{q_x - p_x} = \frac{\Delta x \Sigma y - \Sigma x \Delta y}{2\Delta x}$$



(a) *Folded in 3D space* **(b)** *Unfolded in 2D space*

Figure 4.12: ***Unfolding of two Triangles***

Applying this expression to the case of two triangles that needs to be unfolded, the following results.

$$e \coloneqq \frac{v_2 - v_1}{\|v_2 - v_1\|} \qquad p \coloneqq r_1 - v_1 \qquad q \coloneqq r_2 - v_1$$

$$p_y = \langle e \,|\, p \rangle \qquad q_y = \langle e \,|\, q \rangle$$

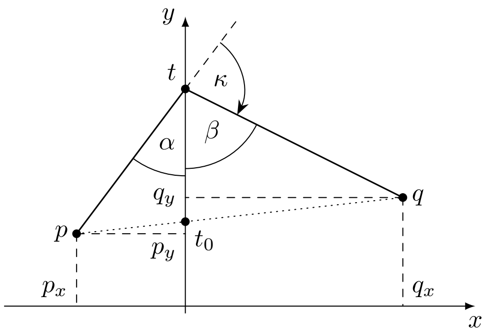
$$p_x = -\|p - p_y e\| \qquad q_x = \|q - q_y e\|$$

Unfolding two adjacent triangles that share a common edge from three-dimensional space into the two-dimensional plane is not unique as there are two cases. In one case, the triangles might overlap and in the other not. The above sign for p_x is therefore important.

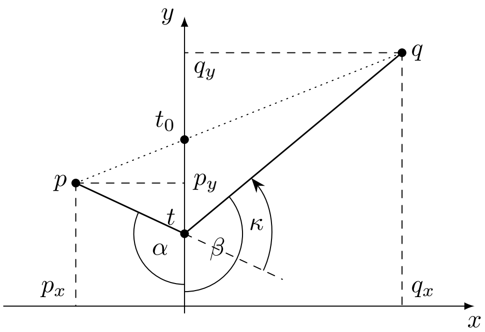
> **LEMMA 4.1:** Unfolding leads to geodesic
>
> *Connecting p, r, and q in that order by straight lines leads to locally shortest and straightest geodesics as long as $\lambda \in (0,1)$. It solves the discrete boundary value problem for locally shortest and straightest geodesics when boundary points lie in the inside of adjacent triangles.*

PROOF:
According to Polthier and Schmies (2006) and Martínez, Velho, and Carvalho (2005), locally shortest and straightest geodesics coincide on polyhedral if they do not contain any vertices. So, it is sufficient to show that the discrete curvature at the crease is zero. But this has be true by construction. □

4.7 Unfolding with Desired Curvature



(a) $\kappa < 0$



(b) $\kappa > 0$

Figure 4.13: ***2D Unfolding Curvature Primitive***

Again, a two-dimensional primitive is used.

$$f(x) = -\cot\alpha(x - p_x) + p_y$$

$$t = f(0) = p_y + |p_x| \cot\alpha$$

Now, combine two points to both sides with a desired discrete curvature. For this, assume $p_x < 0$ and $q_x > 0$. Furthermore, the desired curvature is positive $\kappa > 0$ and taken mathematically positive. $\alpha, \beta \in (0, \pi)$ and $\kappa \in (-\pi, \pi)$ with $\kappa < \alpha$ and $\pi + \kappa > \alpha$

$$\kappa + \pi = \alpha + \beta$$

From the former computation, it is clear that their points on the ordinate have to coincide.

$$p_y - p_x \cot\alpha = q_y + q_x \cot\beta$$

$$\cot\beta = \cot(\pi - (\alpha - \kappa)) = -\cot(\alpha - \kappa)$$

$$\cot(\alpha - \kappa) = \frac{\cos\kappa \cot\alpha + \sin\kappa}{\cos\kappa - \sin\kappa \cot\alpha}$$

$$\varphi := \cot\alpha$$

$$p_y - p_x\varphi = q_y - q_x \frac{\varphi\cos\kappa + \sin\kappa}{\cos\kappa - \varphi\sin\kappa}$$

$$q_x \frac{\varphi\cos\kappa + \sin\kappa}{\cos\kappa - \varphi\sin\kappa} = q_y - p_y + p_x\varphi$$

$$\varphi q_x \cos\kappa + q_x \sin\kappa = (\Delta y + \varphi p_x)(\cos\kappa - \varphi\sin\kappa)$$

$$\varphi q_x \cos\kappa + q_x \sin\kappa = \Delta y \cos\kappa - \varphi\Delta y \sin\kappa + \varphi p_x \cos\kappa - \varphi^2 p_x \sin\kappa$$

$$\varphi^2 p_x \sin\kappa + \varphi(\Delta x \cos\kappa + \Delta y \sin\kappa) + q_x \sin\kappa - \Delta y \cos\kappa = 0$$

$$\varphi = \frac{1}{2p_x \sin\kappa}\left[-(\Delta x\cos\kappa + \Delta y\sin\kappa) \pm \sqrt{(\Delta x\cos\kappa + \Delta y\sin\kappa)^2 - 4p_x\sin\kappa(q_x\sin\kappa - \Delta y\cos\kappa)}\right]$$

$$\varphi = \frac{1}{2p_x \sin\kappa}\left[-(\Delta x\cos\kappa + \Delta y\sin\kappa) \pm \sqrt{(\Sigma x\cos\kappa + \Delta y\sin\kappa)^2 - 4p_x q_x}\right]$$

$$t = \frac{1}{2\sin\kappa}\left[\Delta x\cos\kappa + \Sigma y\sin\kappa \pm \sqrt{(\Sigma x\cos\kappa + \Delta y\sin\kappa)^2 - 4p_x q_x}\right]$$

$$t = t_0 + \Delta t$$

$$t = t_0 + \frac{1}{2\sin\kappa}\left[-2t_0\sin\kappa + \Delta x\cos\kappa + \Sigma y\sin\kappa \pm \sqrt{(\Sigma x\cos\kappa + \Delta y\sin\kappa)^2 - 4p_x q_x}\right]$$

$$t_0 = \frac{1}{2}(\Sigma y - \Sigma x\frac{\Delta y}{\Delta x})$$

$$\Delta t = \frac{1}{2\sin\kappa}\left[\Sigma x\frac{\Delta y}{\Delta x}\sin\kappa + \Delta x\cos\kappa \pm \sqrt{(\Sigma x\cos\kappa + \Delta y\sin\kappa)^2 - 4p_x q_x}\right]$$

$$2\Delta t\sin\kappa \leq 0$$

$$\Delta t = \frac{1}{2\sin\kappa}\left[\Sigma x\frac{\Delta y}{\Delta x}\sin\kappa + \Delta x\cos\kappa - \sqrt{(\Sigma x\cos\kappa + \Delta y\sin\kappa)^2 - 4p_x q_x}\right]$$

4.8 Desired Curvature Stencil

4.9 Desired Curvature Mapping

4.10 Curve Smoothing Algorithm

Curve smoothing is not a discrete problem. Using only a finite amount of steps is probably not possible. So, we use a process of convergence. This has the advantage that not all triangles need to be unfolded at once. This makes the implementation and parallelization much simpler. The discrete algorithms for tracing geodesics are not easy to parallelize.

Idea and Overview

Edge Vertex Relaxation We need to take a look at topological and numerical robustness.

Vertex Vertex Relaxation

Critical Vertex Handling

Artifact Removal and Self-Intersection Handling

Desired Curvature Mapping Desired curvatures need to be constant along edges. Therefore interpolate on angle-basis around vertex. We do not want to loose information of desired curvatures.

Curve Evaluation

Correctness and Convergence Correctness can be shown by showing the convergence to the curvature values. This by definition of the given curvature values smooths the curve. The convergence might only be shown for contracting the curve by smoothing. As the limit is the geodesic, the prove of its convergence is there.

5 Evaluation and Results

6 Conclusions, Limitations, and Future Work

During the course of this thesis, I build on the theory of Polthier and Schmies (2006) and formulated a rigorous mathematical foundation for the handling of discrete geodesics and curve smoothing algorithms on polyhedral surfaces. Hereby, special emphasis has been put on the consideration of boundaries which are are often neglected by the general literature. Even on smooth manifolds, in the presence of boundaries the concept of locally shortest geodesics no longer agrees with the definition of straightest geodesics. This problem cannot be eliminated by simply redefining geodesic curvature on boundaries to be zero as this would lead to unintuitive results. State-of-the-art literature concerned with algorithms to generate geodesic paths and distances considers the concepts for discrete locally shortest and straightest geodesics as the de facto standard for the generalization of geodesics onto polyhedral surfaces. Regarding future work, I also want to consider other formulations of geodesics which may lead to a more intuitive interpretation and the removal of inconsistencies.

Following the mathematical foundation, different metrics for measuring the smoothness and similarity of curves have been proposed to allow for objective and quantitative comparisons of different curve smoothing states and techniques. The largest problem is the uniform weighting along the length of a curve. A curve moving with respect to its curvature gets much shorter at parts with a higher curvature value. A general length mapping would be more intuitive.

In the practical part of this work, I designed and implemented an algorithm for the smoothing of surface mesh curves. All the implementation-specific details have been given as C++ code snippets to allow for reproducibility. My algorithm builds on and improves the work of Martínez, Velho, and Carvalho (2005) and Lawonn et al. (2014) by using data structures from Mancinelli et al. (2022) and a consistent curvature propagation. It is based on an iterative approach to locally optimize geodesic curvature values in conjunction with surface-dependent penalty values to keep the outcome close to its original. As such, it can characterized to be part of the variational domain of solutions to curve smoothing. Together with the defined smoothing metrics, I was able to proof that the algorithm indeed smoothes an initial curve and that it converges.

I have stated the concrete data structures for polyhedral surfaces and surface mesh curves and also added primitive operations. This allowed for general construction on how to move a surface mesh curve along a polyhedral surface. Also, I propose a more robust method for generating desired geodesic curvature values by using a weighting stencil. Instead of an inconsistent curvature propagation, I have introduce curvature mapping. The largest limitation of consistently moving surface mesh curves along polyhedral surface is the lack of mathematical convergence properties concerning discrete geodesic curvature.

The algorithms have been implemented as parallelized version on the CPU and GPU with the standard thread library of the C++ STL and the compute shaders of OpenGL, respectively. I have designed tests based on the “Thingi10K” dataset to evaluate their robustness and efficiency. Even for polyhedral surfaces consisting of millions of triangles, surface mesh curves exhibit typically only exhibit thousands of control points. Most of curve smoothing or straightening algorithms are not parallelizable. Due to iterative nature and the convergence behavior, I was able to show that CPU-based and GPU-based parallelization indeed allows for faster determination of smoothed curves. Still, the smoothing of curves seems not to be the bottleneck of the application. So, a parallelization for only a few thousands points might quickly become infeasible. As there are data and flow dependencies in a curve smoothing procedure, parallelization is limited as it needs to constantly communicate with the CPU. A bigger emphasis should lie on the robustness.

In the future work, I want to apply the curve smoothing to segmentation of lung lobes. Curves should not only be shortened but it should be possible to expand. The main problem here lies the proof of convergence. Anisotropic metrics More generic testing More applications in medical domains

References

Albrecht, Felix and I. D. Berg (1991). “Geodesics in Euclidean Space with Analytic Obstacle”. In: *Proceedings of the American Mathematical Society* 113 (1), pp. 201–207. DOI: 10.2307/2048459.

Alexander, Ralph and S. Alexander (1981). “Geodesics in Riemannian Manifolds-with-Boundary”. In: *Indiana University Mathematics Journal* 30 (4), pp. 481–488. DOI: 10.1512/iumj.1981.30.30039.

Alexander, Stephanie B., I. David Berg, and Richard L. Bishop (1986). “Cauchy Uniqueness in the Riemannian Obstacle Problem”. In: *Differential Geometry Peñíscola 1985*. Springer Berlin Heidelberg, pp. 1–7. ISBN: 978-3-540-44844-0. DOI: 10.1007/BFb0076617.

— (1987). “The Riemannian Obstacle Problem”. In: *Illinois Journal of Mathematics* 31 (1), pp. 167–184. DOI: 10.1215/ijm/1255989406.

Alirr, Omar and Ashrani Aizzuddin Abd. Rahni (August 2019). “Survey on Liver Tumour Resection Planning System: Steps, Techniques, and Parameters”. In: *Journal of Digital Imaging* 33. DOI: 10.1007/s10278-019-00262-8.

Benhabiles, Halim et al. (December 2011). “Learning Boundary Edges for 3D-Mesh Segmentation”. In: *Computer Graphics Forum* 30, pp. 2170–2182. DOI: 10.1111/j.1467-8659.2011.01967.x.

Bischoff, Stephan, Tobias Weyand, and Leif Kobbelt (January 2005). “Snakes on Triangle Meshes”. In: *Proceedings of Bildverarbeitung für die Medizin*, pp. 208–212. DOI: 10.1007/3-540-26431-0_43.

Bommes, David and Leif Kobbelt (January 2007). “Accurate Computation of Geodesic Distance Fields for Polygonal Curves on Triangle Meshes”. In: *Proceedings of the Vision, Modeling, and Visualization Conference*, pp. 151–160. URL: https://www-sop.inria.fr/members/David.Bommes/publications/geodesic.pdf (visited on 11/16/2022).

Bourke, Paul (n.d.). *Object Files (.obj)*. URL: http://paulbourke.net/dataformats/obj/ (visited on 03/10/2023).

Carmo, Manfredo P. Do (2016). *Differential Geometry of Curves and Surfaces*. Revised & Updated Second Edition. Dover Publications. ISBN: 978-0-486-80699-0.

Chaikin, George (December 1974). “An Algorithm for High-Speed Curve Generation”. In: *Computer Graphics and Image Processing* 3, pp. 346–349. DOI: 10.1016/0146-664X(74)90028-8.

Cheney, Ward and David Kincaid (2008). *Numerical Mathematics and Computing*. Sixth Edition. Thomson Brooks/Cole. ISBN: 978-0-495-11475-8.

Congress, Library of (n.d.). *STL (STereoLithography) File Format Family*. URL: https://www.loc.gov/preservation/digital/formats/fdd/fdd000504.shtml (visited on 03/10/2023).

cppreference.com (2023). URL: https://en.cppreference.com/w/ (visited on 01/15/2023).

Crane, Keenan, Clarisse Weischedel, and Max Wardetzky (September 2013). “Geodesics in Heat: A New Approach to Computing Distance Based on Heat Flow”. In: *ACM Transactions on Graphics* 32. DOI: 10.1145/2516971.2516977.

Crane, Keenan et al. (2020). *A Survey of Algorithms for Geodesic Paths and Distances*. DOI: 10.48550/ARXIV.2007.10430. URL: https://arxiv.org/abs/2007.10430 (visited on 02/18/2023).

Dijkstra, Edsger W. (1959). “A Note on Two Problems in Connexion with Graphs”. In: *Numerische Mathematik* 1, pp. 269–271. URL: https://www.semanticscholar.org/paper/A-note-on-two-problems-in-connexion-with-graphs-Dijkstra/45786063578e814444b8247028970758bbbd0488 (visited on 11/17/2022).

Dyn, Nira, D. Levin, and D. Liu (April 1992). “Interpolatory Convexity-Preserving Subdivision Schemes for Curves and Surfaces”. In: *Computer-Aided Design* 24, pp. 211–216. DOI: 10.1016/0010-4485(92)90057-H.

Elstrodt, Jürgen (2018). *Maß- und Integrationstheorie*. Eighth Edition. Springer Spektrum. ISBN: 978-3-662-57938-1. DOI: 10.1007/978-3-662-57939-8.

Engelke, Wito et al. (August 2018). “Autonomous Particles for Interactive Flow Visualization”. In: *Computer Graphics Forum* 38. DOI: 10.1111/cgf.13528.

Forster, Otto (2016). *Analysis 1. Differential- und Integralrechnung einer Veränderlichen*. Springer Spektrum. ISBN: 978-3-658-11544-9. DOI: 10.1007/978-3-658-11545-6.

Goldhorn, Karl-Heinz, Hans-Peter Heinz, and Magarita Kraus (2009). *Moderne mathematische Methoden der Physik*. Vol. 1. Springer-Verlag Berlin Heidelberg. ISBN: 978-3-540-88543-6. DOI: 10.1007/978-3-540-88544-3.

Guibas, Leonidas and Jorge Stolfi (April 1985). “Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams”. In: *ACM Transactions on Graphics* 4, pp. 74–123. DOI: 10.1145/282918.282923. URL: http://sccg.sk/~samuelcik/dgs/quad_edge.pdf (visited on 11/07/2020).

Hertzmann, Aaron and Denis Zorin (2000). “Illustrating Smooth Surfaces”. In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’00. ACM Press/Addison-Wesley Publishing Co., 517–526. ISBN: 1581132085. DOI: 10.1145/344779.345074.

Hofer, Michael and Helmut Pottmann (August 2004). “Energy-Minimizing Splines in Manifolds”. In: *ACM Transactions on Graphics* 23, pp. 284–293. DOI: 10.1145/1015706.1015716.

Ji, Zhongping et al. (September 2006). “Easy Mesh Cutting”. In: *Computer Graphics Forum* 25, pp. 283–291. DOI: 10.1111/j.1467-8659.2006.00947.x.

Jin, Shuangshuang, Robert Lewis, and David West (February 2005). “A Comparison of Algorithms for Vertex Normal Computation”. In: *The Visual Computer* 21, pp. 71–82. DOI: 10.1007/s00371-004-0271-1.

Joyce, Dominic (2009). *On manifolds with corners*. DOI: 10.48550/ARXIV.0910.3518.

Jung, Moonryul and Haengkang Kim (November 2004). “Snaking Across 3D Meshes”. In: *Proceedings of Pacific Graphics*, pp. 87–93. DOI: 10.1109/PCCGA.2004.1348338.

Kaplansky, Lotan and Ayellet Tal (October 2009). “Mesh Segmentation Refinement”. In: *Computer Graphics Forum* 28, pp. 1995–2003. DOI: 10.1111/j.1467-8659.2009.01578.x.

Kass, Michael, Andrew Witkin, and Demetri Terzopoulos (January 1988). “Snakes: Active Contour Models”. In: *IEEE Proceedings on Computer Vision and Pattern Recognition* 1, pp. 321–331. URL: https://sites.pitt.edu/~sjh95/related_papers/Kass1988_Article_SnakesActiveContourModels.pdf (visited on 11/16/2022).

Kimmel, Ron and J. A. Sethian (1996). *Fast Marching Methods for Computing Distance Maps and Shortest Paths*. Tech. rep. Lawrence Berkeley National Laboratory. URL: https://escholarship.org/uc/item/7kx079v5 (visited on 11/16/2022).

— (August 1998). “Computing Geodesic Paths on Manifolds”. In: *Proceedings of the National Academy of Sciences of the United States of America* 95, pp. 8431–5. DOI: 10.1073/pnas.95.15.8431.

Kindlmann, Gordon et al. (November 2003). "Curvature-Based Transfer Functions for Direct Volume Rendering: Methods and Applications". In: vol. 2003, pp. 513–520. ISBN: 0-7803-8120-3. DOI: 10.1109/VISUAL.2003.1250414.

Knuth, Donald E. (1997). *The Art of Computer Programming. Fundamental Algorithms*. Third Edition. Vol. 1. Addison-Wesley. ISBN: 978-0-201-89683-1.

Kulling, Kim (2021). *The Asset-Importer-Lib Documentation*. URL: https://assimp-docs.readthedocs.io/en/v5.1.0/ (visited on 03/10/2023).

Kühnel, Wolfgang (2013). *Differentialgeometrie*. Sixth Edition. Springer Spektrum. ISBN: 978-3-658-00614-3. DOI: 10.1007/978-3-658-00615-0.

Lai, Yu-Kun et al. (January 2007). "Robust Feature Classification and Editing". In: *IEEE Transactions on Visualization and Computer Graphics* 13, pp. 34–45. DOI: 10.1109/TVCG.2007.19.

Lawonn, Kai et al. (2014). "Adaptive and Robust Curve Smoothing on Surface Meshes". In: *Computers & Graphics* 40, pp. 22–35. DOI: 10.1016/j.cag.2014.01.004.

Lee, D. and F. Preparata (September 1984). "Euclidean Shortest Paths in the Presence of Rectilinear Barriers". In: *Networks* 14, pp. 393–410. DOI: 10.1002/net.3230140304.

Lee, Yunjin and S. Lee (September 2002). "Geometric Snakes for Triangular Meshes". In: *Computer Graphics Forum* 21, pp. 229 –238. DOI: 10.1111/1467-8659.t01-1-00582.

Lee, Yunjin et al. (January 2004). "Intelligent Mesh Scissoring Using 3D Snakes". In: pp. 279–287. DOI: 10.1109/PCCGA.2004.1348358.

Lévy, Bruno et al. (July 2002). "Least Squares Conformal Maps for Automatic Texture Atlas Generation". In: *ACM Transactions on Graphics* 21, pp. 362–371. DOI: 10.1145/566654.566590.

Ma, Li and Dezhong Chen (June 2007). "Curve Shortening in a Riemannian Manifold". In: *Annali Di Matematica Pura Ed Applicata* 186, pp. 663–684. DOI: 10.1007/s10231-006-0025-y.

Mancinelli, Claudio et al. (May 2022). "b/Surf: Interactive Bzier Splines on Surface Meshes". In: *IEEE Transactions on Visualization and Computer Graphics* PP. DOI: 10.1109/TVCG.2022.3171179.

Martínez, Dimas, Paulo de Carvalho, and Luiz Velho (November 2007). "Geodesic Bezier Curves: A Tool for Modeling on Triangulations". In: *Brazilian Symposium on Computer Graphics and Image Processing*, pp. 71–78. ISBN: 978-0-7695-2996-7. DOI: 10.1109/SIBGRAPI.2007.38.

Martínez, Dimas, Luiz Velho, and Paulo de Carvalho (October 2005). "Computing Geodesics on Triangular Meshes". In: *Computers & Graphics* 29, pp. 667–675. DOI: 10.1016/j.cag.2005.08.003.

MathWorld, Wolfram (2022). *Teardrop Curve*. URL: https://mathworld.wolfram.com/TeardropCurve.html (visited on 11/28/2022).

Max, Nelson (January 1999). "Weights for Computing Vertex Normals from Facet Normals". In: *Journal of Graphics Tools* 4. DOI: 10.1080/10867651.1999.10487501.

McCool, Michael, Arch D. Robison, and James Reinders (2012). *Structured Parallel Programming: Patterns of Efficient Computation*. Morgan Kaufmann – Elsevier. ISBN: 978-0-12-415993-8.

Mehlhorn, Kurt and Peter Sanders (2008). *Algorithms and Data Structures. The Basic Toolbox*. Springer-Verlag Berlin Heidelberg. ISBN: 978-3-540-77977-3. DOI: 10.1007/978-3-540-77978-0.

Meyer, Mark et al. (November 2001). “Discrete Differential-Geometry Operators for Triangulated 2-Manifolds”. In: *Proceedings of Visualization and Mathematics* 3. DOI: 10.1007/978-3-662-05105-4_2.

Meyers, Scott (2014). *Effective Modern C++*. O’Reilly Media. ISBN: 978-1-491-90399-5.

Mitchell, Joseph, David Mount, and Christos Papadimitriou (August 1987). “The Discrete Geodesic Problem”. In: *SIAM Journal on Computing* 16, pp. 647–668. DOI: 10.1137/0216045.

Morera, Dimas Martínez, Luiz Velho, and Paulo Cezar Pinto Carvalho (2008). “Subdivision Curves on Triangular Meshes”. In: URL: https://www.semanticscholar.org/paper/Subdivision-Curves-on-Triangular-Meshes-Morera-Velho/595d28aacea33ba038d36e7dc403c156a9248905 (visited on 11/16/2022).

Munzner, Tamara (2014). *Visualization Analysis and Design*. A K Peters Visualization Series. CRC Press. URL: https://www.cs.ubc.ca/~tmm/vadbook/ (visited on 11/16/2022).

OpenGL (2023). *The Industry’s Foundation for High Performance Graphics*. URL: https://www.opengl.org/ (visited on 01/15/2023).

Park, Jongha et al. (May 2019). “Fully Automated Lung Lobe Segmentation in Volumetric Chest CT with 3D U-Net: Validation with Intra- and Extra-Datasets”. In: *Journal of Digital Imaging* 33. DOI: 10.1007/s10278-019-00223-1.

Patterson, David A. and John L. Hennessy (2014). *Computer Organization and Design. The Hardware/Software Interface*. Fifth Edition. Morgan Kaufmann – Elsevier. ISBN: 978-0-12-407726-3.

Pawellek, Markus (2023a). *nanoreflex. Reactive and Flexible Curve Smoothing on Surface Meshes*. URL: https://github.com/lyrahgames/nanoreflex (visited on 01/15/2023).

— (2023b). *reflex. Reactive and Flexible Curve Smoothing on Surface Meshes*. URL: https://github.com/lyrahgames/reflex (visited on 01/15/2023).

Pellacini, Fabio, Giacomo Nazzaro, and Edoardo Carra (2019). “Yocto/GL: A Data-Oriented Library For Physically-Based Graphics”. In: *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*. Ed. by Marco Agus, Massimiliano Corsini, and Ruggero Pintus. The Eurographics Association. ISBN: 978-3-03868-100-7. DOI: 10.2312/stag.20191373.

Polthier, Konrad and Markus Schmies (2006). “Straightest Geodesics on Polyhedral Surfaces”. In: *ACM SIGGRAPH 2006 Courses*. SIGGRAPH ’06. Association for Computing Machinery, 30–38. DOI: 10.1145/1185657.1185664.

Pottmann, Helmut and Michael Hofer (October 2005). “A Variational Approach to Spline Curves on Surface”. In: *Computer Aided Geometric Design* 22, pp. 693–709. DOI: 10.1016/j.cagd.2005.06.006.

Reddy, Martin (2011). *API Design for C++*. Morgan Kaufmann – Elsevier. ISBN: 978-0-12-385003-4.

Rusinkiewicz, Szymon (October 2004). “Estimating Curvatures and Their Derivatives on Triangle Meshes”. In: pp. 486–493. ISBN: 0-7695-2223-8. DOI: 10.1109/TDPVT.2004.1335277.

Sethian, J. A. (March 1996). “A Marching Level Set Method for Monotonically Advancing Fronts”. In: *Proceedings of the National Academy of Sciences of the United States of America* 93, pp. 1591–5. DOI: 10.1073/pnas.93.4.1591.

Sharp, Nicholas and Keenan Crane (December 2020). “You Can Find Geodesic Paths in Triangle Meshes by Just Flipping Edges”. In: *ACM Transactions on Graphics* 39.6, pp. 1–15. DOI: 10.1145/3414685.3417839.

Shewchuk, J. R. (1996). “Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator”. In: *Applied Computational Geometry Towards Geometric Engineering*. Springer Berlin Heidelberg, pp. 203–222. DOI: 10.1007/978-3-030-81354-3_2.

Smed, Jouni and Harri Hakonen (2006). *Algorithms and Networking for Computer Games*. John Wiley & Sons. ISBN: 978-0-047-01812-5.

Stahl, Saul and Catherine Stenson (2013). *Introduction to Topology and Geometry*. Second Edition. John Wiley & Sons. ISBN: 978-1-118-10810-9.

Standard C++ Foundation (2023). URL: https://isocpp.org/ (visited on 01/15/2023).

Stroustrup, Bjarne (2014). *The C++ Programming Language*. Fourth Edition. Addison-Wesley – Pearson Education. ISBN: 978-0-321-95832-7.

Surazhsky, Vitaly et al. (July 2005). “Fast Exact and Approximate Geodesics on Meshes”. In: *ACM Transactions on Graphics* 24, pp. 553–560. DOI: 10.1145/1073204.1073228.

Vandevoorde, David, Nicolai M. Josuttis, and Douglas Gregor (2018). *C++ Templates: The Complete Guide*. Second Edition. Addison-Wesley – Pearson Education. ISBN: 978-0-321-71412-1.

Williams, Anthony (2019). *C++ Concurrency in Action*. Second Edition. Manning Publications. ISBN: 978-1-61-729469-3.

Xin, Shi-Qing and Guo-Jin Wang (December 2007). “Efficiently Determining a Locally Exact Shortest Path on Polyhedral Surfaces”. In: *Computer-Aided Design* 39, pp. 1081–1090. DOI: 10.1016/j.cad.2007.08.001.

Yocto/GL (2023). *Tiny C++ Libraries for Data-Oriented Physically-based Graphics*. URL: https://github.com/xelatihy/yocto-gl (visited on 11/19/2022).

Yu, Chris, Henrik Schumacher, and Keenan Crane (April 2021). “Repulsive Curves”. In: *ACM Transactions on Graphics* 40, pp. 1–21. DOI: 10.1145/3439429.

Zachow, Stefan et al. (2003). “Draw and Cut: Intuitive 3D Osteotomy Planning on Polygonal Bone Models”. In: *International Congress Series* 1256, pp. 362–369. DOI: 10.1016/S0531-5131(03)00272-3.

Zhou, Qingnan and Alec Jacobson (2016). “Thingi10K. A Dataset of 10,000 3D-Printing Models”. In: *arXiv preprint arXiv:1605.04797*. DOI: 10.48550/arXiv.1605.04797.

A Analysis on Manifolds

Topological manifolds are the basis of differentiable manifolds and describe the class polyhedral surfaces lie in.

DEFINITION A.1: Topological Manifold

Let $n \in \mathbb{N}$. Then a topological n-dimensional manifold M (with boundary) is a second countable Hausdorff space which is locally homeomorphic to $\mathbb{H}^n$.
That is, for all $p \in M$, there exists an open neighborhood U of p in M, an open set $V \subset \mathbb{H}^n$, and a homeomorphism $\varphi\colon U \to V$, called a (coordinate) chart.

For the purpose of this thesis, all spaces will be a second countable Hausdorff space. The definition is given for completeness. A topological manifold with boundary is generalization of the standard concept of topological manifolds without boundary. A topological manifold without boundary is a topological manifold with boundary, whereby its boundary is given by the empty set.

DEFINITION A.2: Smooth Manifold

Let $n \in \mathbb{N}$, $k \in \mathbb{N}_\infty$, and M be a topological n-dimensional manifold. Then, given two charts (U, φ) and (V, ψ) of M, their transition map, also known as coordinate transformation, is defined by the following composition.

$$\varphi|_{U\cap V} \circ \psi|_{U\cap V}^{-1} \colon \psi(U \cap V) \to \varphi(U \cap V)$$

The charts (U, φ) and (V, ψ) are said to be C^k-compatible if either $U \cap V = \emptyset$ or their transition map is a C^k-diffeomorphism.
A C^k-atlas of M is a family of C^k-compatible charts that covers all of M.
By equipping the topological manifold M with a maximal C^k-atlas, we obtain a differentiable n-dimensional manifold of class C^k (with boundary).
For $k = \infty$, it is called a smooth n-dimensional manifold (with boundary).

DEFINITION A.3: Smooth Maps

Let M and N be two manifolds and $F\colon M \to N$. For two given charts (U, φ) of M and (V, ψ) of N, we define the coordinate representation of F by the following composition.

$$\psi \circ F \circ \varphi|_{U\cap F^{-1}(V)}^{-1} \colon \varphi\left(U \cap F^{-1}(V)\right) \to \psi(V)$$

F is a differentiable map of class C^k, if for all pairs of given charts, its coordinate representations is an element of $\mathrm{C}^k(\mathbb{R}^m, \mathbb{R}^n)$.
We call F a C^k-diffeomorphism if it is bijective and in both directions a differentiable map of class C^k.

$$\mathrm{C}^k(M, N) := \left\{ F\colon M \to N \;\middle|\; F \text{ is differentiable of class } \mathrm{C}^k \right\}$$

$$\mathrm{C}^k(M) := \mathrm{C}^k(M, \mathbb{R})$$

DEFINITION A.4: Tangential Space

Let M be a smooth manifold and $p \in M$. A tangential vector in p is a linear and smooth functional $X\colon \mathrm{C}^\infty(M) \to \mathbb{R}$, such that for all $f, g \in \mathrm{C}^\infty(M)$ the following holds.

$$X(fg) = f(p)X(g) + g(p)X(f)$$

The set T_pM of all tangential vectors in p is called tangential space in p. The tangent bundle is then as disjoint union.

$$\mathrm{T}M := \bigsqcup_{p \in M} \mathrm{T}_pM$$

The tangential space is for all points isomorphic to $\mathbb{R}^n$. For a chosen chart, we can easily construct basis vectors. Every tangential vector is a tangent vector of a curve running through that point.

DEFINITION A.5: Differential

$$\mathrm{d}F(p)\colon \mathrm{T}_pM \to \mathrm{T}_{F(p)}N \qquad \mathrm{d}F(p)(X)(f) := X(f \circ F)$$
$$\mathrm{d}F\colon \mathrm{T}M \to \mathrm{T}N$$

DEFINITION A.6: Smooth Embedding

Let M and N be two manifolds and $F\colon M \to N$. F is an immersion if for all $p \in M$ its differential $\mathrm{d}F(p)$ is injective. Furthermore, it is called an embedding if $F\colon M \to F(M)$ is a homeomorphism.

DEFINITION A.7: Embedded Submanifold

Let M be a manifold. Then a subset $S \subset M$ is called an embedded submanifold of dimension k if for all $p \in S$, there exists a chart (U, φ) in M with $p \in U$, such that the following holds.

$$\varphi(U \cap S) = \{x \in \varphi(U) \mid \forall p \in \mathbb{N}, k < p \leq n\colon x_p = 0\}$$

THEOREM A.1: The image of an embedding is an embedded submanifold

The image of an embedding is an embedded submanifold.

DEFINITION A.8: Vector Bundle

Let M be a manifold. Then a vector bundle E rank k over M is a manifold equipped with differentiable and surjective projection $\pi\colon E \to M$, such that the *following properties hold.*

- *For all* $p \in M$*, the fiber* $\pi^{-1}(p) \subset E$ *is isomorphic to* $\mathbb{R}^k$.
- *locally trivial: For all* $p \in M$*, there is a neighborhood* U *of* p *in* M *and a diffeomorphism* $\varphi\colon \pi^{-1}(U) \to U \times \mathbb{R}^k$*, such that* $\pi = \pi_1 \circ \varphi$ *and the restriction of* φ *to the fibers* $\pi^{-1}(p)$ *is a linear isomorphism to* $\mathbb{R}^k$.

A map $\sigma\colon M \to E$ *with* $\pi \circ \sigma = \mathrm{id}_M$ *is called a section.*

DEFINITION A.9: Tensorbundle

For a finite dimensional real vector space V*, a* k*-tensor is a multilinear functional* $T\colon V^k \to \mathbb{R}$*. The space of all* k*-tensors over* V *is denoted by* $\mathrm{T}^k(V)$*. For a manifold* M*, we define the* k*-tensor bundle as follows.*

$$\mathrm{T}^k M \coloneqq \bigsqcup_{p \in M} \mathrm{T}^k(\mathrm{T}_p M)$$

DEFINITION A.10: Riemannian Manifold

A Riemannian manifold is a smooth manifold M *equipped with a positive-definite inner product* g_p *on the tangent space* $\mathrm{T}_p M$ *at each point* $p \in M$*. Thereby, for any chosen chart the coordinate functions of* g *need to be smooth.*

The inner metric of a Riemannian manifold takes the place of the scalar product and measures angles.

THEOREM A.2: For every smooth manifold, there exists a Riemannian metric

For every smooth manifold, there exists a Riemannian metric.

DEFINITION A.11: Normal Space

Let (M, g) *be a Riemannian manifold,* $S \subset M$ *a Riemannian submanifold and* $p \in S$.

$$\mathrm{N}_p S \coloneqq \{x \in \mathrm{T}_p M \mid \forall y \in \mathrm{T}_p S\colon g(x, y) = 0\}$$

The normal space is the orthogonal complement and can in general only be defined for embedded manifolds. For orthogonality, we need a Riemannian manifold. A more general definition is possible by using quotient spaces. As a consequence, the normal space is isomorphic to $\mathbb{R}^{n-k}$.

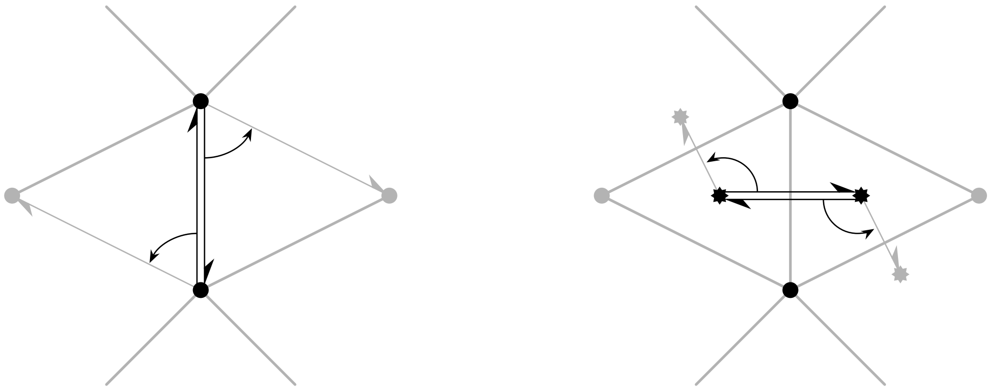


Figure B.1: ***Quad-Edge Scheme***

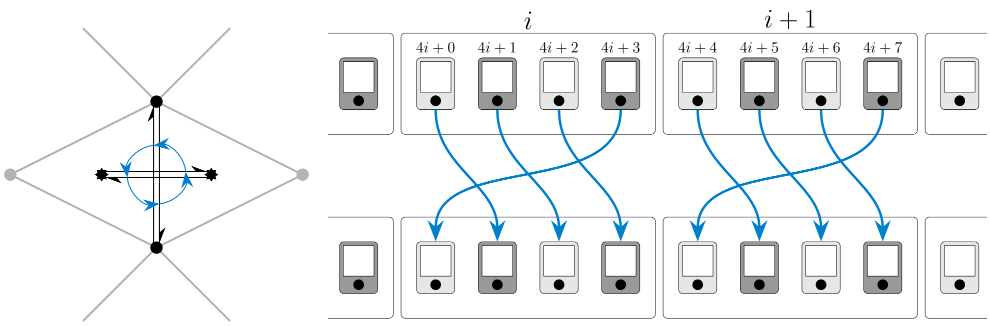


Figure B.2: ***Quad-Edge Rotation Scheme***

B Quad-Edge Algebra

C Mathematical Proofs

DEFINITION: Length of Curves

Let γ be a parameterized curve on $[a,b]$ or $\mathbb{T}^1[a,b]$. Then the length $L(\gamma)$ and arc length, given by $s_\gamma\colon [a,b] \to [0, L(\gamma)]$ or $s_\gamma\colon \mathbb{T}^1[a,b] \to \mathbb{T}^1[0, L(\gamma)]$, respectively, of γ are defined by the following expressions.

$$L(\gamma) := \int_a^b \|\gamma'(t)\| \, \mathrm{d}t \qquad s_\gamma(t) := \int_a^t \|\gamma'(x)\| \, \mathrm{d}x$$

PROOF: (Consistency Definition 2.3)

For every parameterized curve γ, the length and arc length are well-defined because the function $\|\gamma'\|$ is continuous on a compact set and, consequently, uniformly continuous. Hence, the integral expressions defining $L(\gamma)$ and s_γ are finite and well-behaved. Furthermore, according to the fundamental theorem of calculus (Elstrodt 2018; Forster 2016), s_γ is continuously differentiable with derivative $s'_\gamma = \|\gamma'\|$ and surjective. □

LEMMA: Regular curves can be parameterized by arc length

Let $n \in \mathbb{N}$, $k \in \mathbb{N}_\infty$, $[a,b] \subset \mathbb{R}$ be a compact interval, and $\gamma\colon [a,b] \to \mathbb{R}^n$ be an n-dimensional parameterized curve of class C^k that is regular. Then up to a constant shift, there exists a unique k-times continuously differentiable and bijective function $u\colon [c,d] \to [a,b]$ on a compact interval $[c,d] \subset \mathbb{R}$ with strictly positive derivative, such that the composition $\gamma \circ u$ is a curve parameterized by arc length.

PROOF: (Lemma 2.2)

The restriction of $\|\cdot\|$ to $\mathbb{R}^n \setminus \{0\}$ is an infinitely differentiable function. The regularity of γ implies that $\|\gamma'\|$ and, as a direct consequence, the derivative of s_γ are strictly positive on (a,b). Hence, s_γ is bijective with k-times continuously differentiable inverse. The derivative of s_γ^{-1} must also be strictly positive in the interior and continuous as a composition of continuous functions.

$$\left(s_\gamma^{-1}\right)' = \frac{1}{s'_\gamma \circ s_\gamma^{-1}} = \frac{1}{\|\gamma'\| \circ s_\gamma^{-1}} > 0$$

Define $\tilde{\gamma} := \gamma \circ s_\gamma^{-1}$. Then by differential calculus, it follows, that $\tilde{\gamma}$ is k-times continuously differentiable and parameterized by arc length.

$$\|\tilde{\gamma}'\| = \left\|\gamma' \circ s_\gamma^{-1} \cdot \left(s_\gamma^{-1}\right)'\right\| = \left\|\frac{\gamma' \circ s_\gamma^{-1}}{s'_\gamma \circ s_\gamma^{-1}}\right\| = \left\|\frac{\gamma' \circ s_\gamma^{-1}}{\|\gamma'\| \circ s_\gamma^{-1}}\right\| = \frac{\left\|\gamma' \circ s_\gamma^{-1}\right\|}{\left\|\gamma' \circ s_\gamma^{-1}\right\|} = 1$$

This shows that s_γ^{-1} fulfills the required properties and the existence of a parameterization by arc length. To show its uniqueness up to a constant shift, assume an arbitrary function u as given in the lemma. Applying the same reasoning as for s_γ, it is clear that u^{-1} also is k-times continuously differentiable with strictly positive derivative on the interior. By calculation, we may then show its connection to the arc length.

$$1 = \left\|(\gamma \circ u)'\right\| = \left\|\gamma' \circ u\right\| \cdot u' = \frac{\|\gamma' \circ u\|}{\left(u^{-1}\right)' \circ u} = \frac{\|\gamma'\|}{\left(u^{-1}\right)'} \qquad \Longrightarrow \qquad \|\gamma'\| = \left(u^{-1}\right)'$$

To integrate this formula, let $t \in [a,b]$ be arbitrary and use again the fundamental theorem of calculus.

$$s_\gamma(t) = \int_a^t \left\|\gamma'(x)\right\| \, \mathrm{d}x = \int_a^t \left(u^{-1}\right)'(x) \, \mathrm{d}x = u^{-1}(t) - u^{-1}(a) = u^{-1}(t) - c$$

$$u = s_\gamma^{-1} \circ s_\gamma \circ u = s_\gamma^{-1}(\cdot - c) \circ u^{-1} \circ u = s_\gamma^{-1}(\cdot - c)$$

This shows that, up to a constant shift, u is the same as s_γ^{-1} and the uniqueness of the parameterization by arc length. $\square$

D Further Code

Statutory Declaration

I declare that I have developed and written the enclosed Master's thesis completely by myself, and have not used sources or means without declaration in the text. Any thoughts from others or literal quotations are clearly marked. The Master's thesis was not used in the same or in a similar version to achieve an academic grading or is being published elsewhere.
On the part of the author, there are no objections to the provision of this Master's thesis for public use.

Jena, March 25, 2023

Markus Pawellek