Discrete Geometric Methods for Surface Deformation and Visualisation

Doctoral thesis approved by the

Department of Computer Science

of the

University of Kaiserslautern (TU Kaiserslautern)

for the award of the Doctoral Degree

Doctor of Engineering (Dr.-Ing.)

to

M.Sc. Andreas Silvan Berres

Date of viva: 2015/10/22

Dean: Prof. Dr. Klaus Schneider
First reviewer: Prof. Dr. Hans Hagen
Second reviewer: Prof. Dr. Ken Joy

D 386

Contents

Acknowledgements

There are some people, without whom this dissertation would not have been possible. I wholeheartedly thank all of these people, those who were named individually, as well as everyone who is part of one of the mentioned groups.

First and foremost, I would like to thank Prof. Dr. Hans Hagen for giving me the opportunity to continue my studies in this exciting field, and to participate in the International Research Training Group. He was the best boss I could have hoped for – he always believed in my abilities, supported me throughout my entire studies, and has given me invaluable career advice.

Furthermore, I would like to thank Prof. Dr. Ken Joy for hosting me at his lab during my research visits at UC Davis. I very much enjoyed the group discussions during the weekly roundtable meetings, as well as the conversations we have had outside of these meetings.

During my research visits at UC Davis, I also got the chance to closely collaborate with Dr. Harald Obermaier. I would like to thank him for the countless constructive discussions we have had, in person and in writing.

Moreover, I would like to thank Prof. Dr. Tom Peters for our interesting discussions and for hosting me at his lab during my research visit at UConn.

Finally, I would like to thank Prof. Dr. Christoph Garth, Dr. Daniel Engel, and Dr. Peter Salz, with whom I have had valuable discussions about my work, and who have given me helpful advice on academic writing.

Doing a Ph.D. requires not only great advisors, but also a great working environment. I would like to thank all members of the *Computer Graphics and HCI* and *Computational Topology* groups for being such pleasant colleagues. I am proud to call a lot of them my friends, and I am deeply grateful for their continued support. A special thank you goes to Peter, Lars, Susie, and Laura, who have opened their homes to me when my apartment was a construction site.

This list would not be complete without Mady Gruys, the heart of our group. She always has an open door and an open ear, no matter what the topic may be. For this, I would like to thank her.

A great working environment should be complemented with a well-balanced personal life. That balance is difficult to achieve, however, thanks to my friends and family, I have achieved a good balance. I would like to thank them for their continued love and support:

My parents have nurtured my interest in science since I was a child. They have continuously supported my studies, and they have given me advice and even feedback on my work. Without them, I would not be the person I am today, and I am proud to have them as my parents.

I would like to thank my friends and my siblings for inspiring conversations, sharing our good and bad experiences, and for always being there for me. They have been supportive and understanding when I had to put my studies before spending valuable time together. Life without them would not be the same, despite the physical distance separating me from some of them. Special thanks go to Til and Betty, Dagmar, Frank, Brandon, Christian, Tobias, Peter, Susie and Chris, Laura, Maurice, Ben, and Jonas.

Finally, I would like to thank my sports teams, the *Roller Girls of the Apocalypse* (Kaiserslautern) and the *Sac City Rollers* (Sacramento), for giving me an outlet for stress that is not only effective, but also a lot of fun. Being around so many strong women (and men) has been empowering, and I would like to thank them for being great teammates and great role models.

Abstract

Industrial design has a long history. With the introduction of Computer-Aided Engineering, industrial design was revolutionised. Due to the newly found support, the design workflow changed, and with the introduction of virtual prototyping, new challenges arose. These new engineering problems have triggered new basic research questions in computer science.

In this dissertation, I present a range of methods which support different components of the virtual design cycle, from modifications of a virtual prototype and optimisation of said prototype, to analysis of simulation results.

Starting with a virtual prototype, I support engineers by supplying intuitive discrete normal vectors which can be used to interactively deform the control mesh of a surface. I provide and compare a variety of different normal definitions which have different strengths and weaknesses. The best choice depends on the specific model and on an engineer's priorities. Some methods have higher accuracy, whereas other methods are faster.

I further provide an automatic means of surface optimisation in the form of minimising total curvature. This minimisation reduces surface bending, and therefore, it reduces material expenses. The best results can be obtained for analytic surfaces, however, the technique can also be applied to real-world examples.

Moreover, I provide engineers with a curvature-aware technique to optimise mesh quality. This helps to avoid degenerated triangles which can cause numerical issues. It can be applied to any component of the virtual design cycle: as a direct modification of the virtual prototype (depending on the surface definition), during optimisation, or dynamically during simulation.

Finally, I have developed two different particle relaxation techniques that both support two components of the virtual design cycle. The first component for which they can be used is discretisation. To run computer simulations on a model, it has to be discretised. Particle relaxation uses an initial sampling, and it improves it with the goal of uniform distances or curvature-awareness. The second component for which they can be used is the analysis of simulation results. Flow visualisation is a powerful tool in supporting the analysis of flow fields through the insertion of particles into the flow, and through tracing their movements. The particle seeding is usually uniform, e.g. for an integral surface, one could seed on a square. Integral surfaces undergo strong deformations, and they can have highly varying curvature. Particle relaxation redistributes the seeds on the surface depending on surface properties like local deformation or curvature.

Chapter 1

Introduction

In industrial surface design, the traditional approach to designing a model typically started with a blank sheet of paper. It involved various trial-and-error iterations of designing a masterpiece, and manually calculating static geometric and physical properties. Physical prototypes had to be built, and experiments were conducted on the prototype to test its physical properties [LHK^{+}08]. The model was then optimised based on expert knowledge and experience. This process was tedious and expensive due to its high amount of material expenses, and the cost of experimenting on a physical prototype, e.g. material, equipment, and personnel.

Computer-Aided Engineering (CAE) improved this process through the introduction of virtual prototyping. It eliminated the need to design the masterpiece drawing on paper, and having to repeatedly draw similar versions of the model. Instead, industrial surface designers can use Computer-Aided Design (CAD) tools to design a model once, and they can easily make small changes to optimise it. It is possible to interactively view the CAD model as a 3D object from different angles, where originally, each additional view was time-consuming to draw. The capability to virtually model a prototype allows engineers to quickly conduct evaluations for different sets of parameters, based on the virtual prototype, static geometric and physical properties. As each physical prototype can take days or weeks to produce, this results in a significant reduction of not only time, but also cost.

Using virtual prototypes, engineers can evaluate dynamic physical properties early on in the design process without having to wait for the model to be manufactured to do experiments. They can examine the impact of small changes, and see how the model behaves in different scenarios based on an underlying physical model. Most importantly, virtual prototyping facilitates identifying and fixing problems early on in the design process. This substantially reduces cost as it becomes much more expensive if a problem is found during the physical prototyping phase, on the shop floor, or even by a customer [McL01].

Over the past decade, computer hardware has improved tremendously, both in terms of processing power, and in terms of storage capacities. This has led to an increasing amount of research in parallel computing and simulation, which has facilitated the development of complex physical models. Using these models, it is possible to conduct virtual experiments under physically accurate conditions. Simulations can be run within a much shorter time span than the time required to build a physical model, conduct experiments, and measure physical quantities.

The possibilities arising from these new technologies lead to interesting new application problems. These application problems, in turn, trigger basic research questions in computer science.

Computer simulation plays a valuable role in providing essential results for application problems such as statistical mechanics problems [AT89]. It can be used to evaluate and to improve models of the real world by comparing the results of physical experiments to those of simulations of the model. Furthermore, it can also be used to evaluate theories by predicting an outcome, and then comparing the predictions to simulation results. As such, simulation builds a bridge between theory and experimental work, and it can provide insight into physical reality.

Historically, computer simulation evolved from physical manipulation and analysis of models, e.g. of gelatine balls [MH36] or metal ball bearings [PMKW78] representing molecules in a liquid. However, using physical objects is time-consuming, expensive, and limited by real-world physical properties like gravity [AT89].

Simulation is based on a mathematical model of reality which is defined through a system of differential equations. Solving this system of equations is computationally expensive. To reduce complexity, the continuous model defined through the system of equations can be discretised and solved numerically. This is particularly relevant in applications like flow simulation, which require a discretisation of the continuous model. In flow visualisation, scientists create visual representations of a vector field's simulated flow behaviour. They achieve this by inserting massless seed particles into the flow, and tracing their movement over time. A single particle can be traced individually, or one can trace group of particles that are seeded on a line or on a surface. This raises a number of questions. First of all, it is not clear what constitutes a good initial seeding. This is the step during which discretisation occurs: a set of discrete points is created from a (potentially) continuous model. Second, all interesting application-oriented flow fields are non-uniform, and scientists are very interested in finding singularities in the field. With a non-uniform vector field, it is worth investigating if it makes sense to adapt the discretisation over the course of simulation to improve sampling. Finally, there is the question of *how* to adapt the discretisation.

For virtual prototyping, a combination of CAD tools and simulation is employed. First, a model is created using a CAD tool. This model usually attempts to mimic the visuals imagined by the product designer. Then, it is exported into a simulation environment to determine its physical properties. The simulation results are then evaluated to find out strengths and weaknesses. Based on this evaluation, the model is adapted manually, fully automated, or with CAE support. This process is iterated until one obtains a model, which fulfils visual design criteria, and which possesses required physical properties.

Adapting the model requires user interaction with large and complex models and datasets. If these models are given analytically, the entire experimentation process has to be repeated every time the model is adjusted. In many applications, models are (also) given in discretised form. This confronts the user with the painstaking process of moving each point individually. There is a wide range of options to support the user, ranging from fully automated optimisation to user-guided modification with a large degree of direct control. One possibility lies within global optimisation of a parameter of interest such as a surface's curvature or bending energy. Another possibility is to locally optimise the model based on information from a point's neighbourhood. While this may not result in a globally optimal solution, the methods are a lot more efficient, and it is easier to predict results of local optimisation than to predict those of global optimisation. In addition, optimisation parameters are easier to modify, resulting in easier steering of the modification. Finally, one can give the user a large degree of direct control of the model without requiring them to deal with individual surface points. This can be achieved through intuitive controls, e.g. through a coarser version of the model which acts as a control structure for the surface.

In this thesis, I focus on the interconnection of Computer-Aided Geometric Design (CAGD) methods for surface transformation, and visualisation, keeping in mind the potential for virtual prototyping. I combine discrete geometric methods with visualisation, through which I contribute to a variety of fields (CAGD, CAE, Visualisation, Simulation). The methods I have developed tie into different parts of the virtual design cycle, which is depicted in Figure 1.1. I support CAGD in their development of virtual prototypes. Furthermore, I provide CAE with a variety of semi-automatic and automatic surface optimisation methods. Finally, I supply strategies to improve discretisation and mesh quality during simulation.

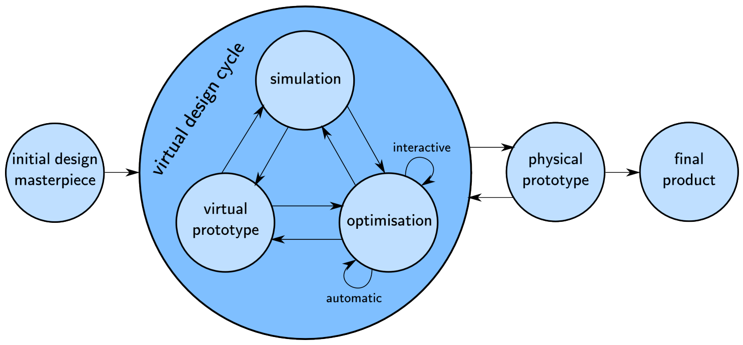


Figure 1.1: Virtual prototyping workflow.

In Chapter 3, I present the results emerging from the work presented in [BHH15]. The main contribution of this work to the field of CAGD lies in a global deformation method which preserves geometric properties, in particular total curvature. This serves to minimise material usage and bending energy. Surface prototyping generates a so-called masterpiece of a new product. Usually, a masterpiece is very similar to the final product, so only small modifications are supposed to be made. I present a linear deformation technique for this kind of modification, preserving total curvature of the masterpiece under infinitesimal deformation. Total curvature is directly connected to the topology of the surface, which is also directly related to the index sum of the singularities of a vector field acting on this surface. This means that the proposed deformations modify the functionality of this surface in a controlled way.

In Chapter 4, I present the results emerging from a work which is currently in submission [BH15]. The main contribution of this work to the field of CAE lies in the development of an intuitive tool to modify a model. Intuitive interaction tools are indispensable in supporting the adoption of virtual prototyping into the manufacturing process, as lack of skill in its use is one of the main reasons for late adoption [McL01]. I employ visualisation and discrete geometric methods to facilitate computer-guided optimisation. Bézier surface patches are particularly well-suited for industrial modelling. First of all, they provide an intuitive control over surface shape. Second, they possess a number of very useful properties such as the variation-diminishing property which ensures that the surface does not wiggle more than the control mesh. Finally, surface patches seam together very well, as two patches with identical control points along the shared border will automatically have identical border curves. This ensures that there is at least C^0 continuity along the border. In this chapter, I define a variety of different discrete normal operators on control points of Bézier surface patches. I compare their performance on various challenging Bézier curve polygons and demonstrate the suitability of the techniques on Bézier surface patches.

In Chapter 5, I present the results emerging from the work presented in [BOJH15], as well as further advances on this project. There are two main contributions of this work. First of all, I contribute to the field of simulation through the development of a discretisation, and surface deformation based on geometric properties. Second, I contribute to the field of CAE through a local method to optimise the discretisation, and the surface as a whole. Simulation of flow behaviour requires a discretisation of the flow information to discrete points. Time surfaces are a common tool in flow visualisation. They visualise advection and deformation in flow fields through a triangulated mesh that is comprised of a discretised surface. Time surfaces are a versatile tool to visualise advection and deformation in flow fields. Due to complex flow behaviours involving stretching, shearing, and folding, straightforward mesh-based representations of these surfaces tend to develop artifacts and degenerate quickly. Common counter-measures rely on refinement and adaptive insertion of new flow particles into the surface representation. This leads to an unpredictable increase in memory requirements and has a strong impact on parallel surface extraction techniques. I propose a novel time surface extraction technique that keeps the number of required flow particles constant, while providing a high level of fidelity and enabling straightforward load balancing. My solution implements a 2D particle relaxation procedure that makes use of local surface metric tensors to model surface deformations. Furthermore, I propose a feature-dependent relaxation procedure, which makes use of local curvature information to produce a surface sampling that faithfully represents surface features. I combine these techniques with an accurate bicubic surface representation to provide an artifact-free surface visualisation. I demonstrate and evaluate benefits of the proposed methods with respect to surface mesh quality, and performance based on three different benchmark datasets.

Chapter 2

Mathematical Background

2.1 Differential Geometry

Differential geometry is a view on geometry that is primarily concerned with the curvature of a surface. The difference to other views is best illustrated by an example [Hag09].

Example 2.1. Consider a circle. Let $x, y \in \mathbb{R}$ be x and y coordinates, $r \in \mathbb{R}$ the radius and $\varphi \in \mathbb{R}$ an angle.

Fundamental geometry A circle is the set of all points which have the same distance from a fixed centre point.

Algebraic geometry A circle with centre $(0,0)$ consists of all points (x,y) satisfying $x^2 + y^2 = r^2$ for a fixed r.

Analytic geometry A circle consists of all φ satisfying $\binom{r \cdot \sin \varphi}{r \cdot \cos \varphi}$ for a fixed r.

Differential geometry A circle is function that has the curvature $\kappa = \frac{1}{r}$ and the torsion $\tau = 0$.

2.2 Topology

A *topology* [Lee11, p. 20] is defined by neighbourhood structures and connectivity of a set.

Definition 2.1 (topology)**.** If X is a set, a *topology on X* is a collection $\mathbf{T}$ of [open] subsets of X satisfying the following properties:

1. X and $\emptyset$ are elements of $\mathbf{T}$.
2. $\mathbf{T}$ is closed under finite intersections: if $U_1, \ldots, U_n$ are elements of $\mathbf{T}$, then their intersection $\bigcup_{i=0}^{n} U_i$ is an element of $\mathbf{T}$.
3. $\mathbf{T}$ is closed under arbitrary unions: if $(U_a)_{a \in A}$ is any (finite or infinite) family of elements of $\mathbf{T}$, then their union $\bigcup_{a \in A} U_a$ is an element of $\mathbf{T}$.

A *topological space* [Lee11, p. 20] is a space that is defined by a topology. Unlike a metric space, it does not necessarily permit to measure distances between elements.

Definition 2.2 (topological space)**.** A pair $(X, \mathbf{T}_X)$ consisting of a set X and a topology $\mathbf{T}_X$ on X is called a *topological space*.

In a *Hausdorff space*, points can be separated by open subsets.

Definition 2.3 (Hausdorff space)**.** A topological space X is said to be a *Hausdorff space* [Lee11, p. 31] if, given any pair of distinct points $x, y \in X$, there exist neighbourhoods U of x and V of y with $U \cap V = \emptyset$.

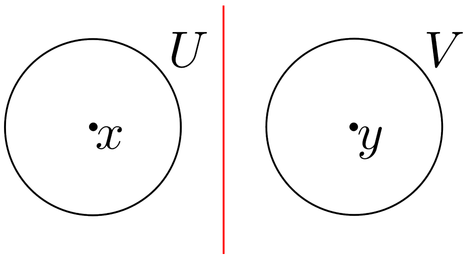


Figure 2.1: The points x and y can be separated by their neighbourhoods.

2.2.1 Neighbourhood

Definition 2.4 (one-ring and two-ring of neighbours)**.** The *one-ring of neighbours* $N_1(\mathbf{p}_i)$ is defined as all points $\mathbf{p}_j$ which are connected to a point $\mathbf{p}_i$ via one edge:

$$N_1(\mathbf{p}_i) = \bigcup_{\exists \text{edge}(\mathbf{p}_i, \mathbf{p}_j)} \mathbf{p}_j .$$

Based on this neighbourhood definition, one can define the *two-ring of neighbours* $N_2(\mathbf{p}_i)$ as all points that are connected to a point $\mathbf{p}_i$ via at most two edges:

$$N_2(\mathbf{p}_i) = \bigcup_{\mathbf{p}_j \in N_1(\mathbf{p}_i)} N_1(\mathbf{p}_j) .$$

Definition 2.5. The one-ring of triangles around a point $\mathbf{p}$ is called a *star set* [LP82]. Figure 2.2 illustrates a star set, where φ_i is the enclosed angle between two neighbouring edges fanning out around the vertex, A_i are areas of triangles, and $\mathbf{a}_i$ is the triangle edge opposing $\mathbf{p}$.

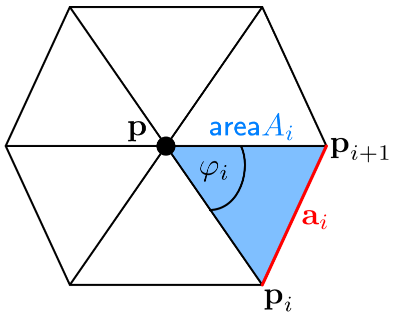


Figure 2.2: Star set attached to $\mathbf{p}$. [LP82]

2.3 Manifolds

A *manifold* is a topological space that locally resembles a metric space, e.g. Euclidean space or Minkowski space (a space from relativity theory in which the scalar product is singular).

Definition 2.6 (manifold)**.** An *n-dimensional (topological) manifold* [Lee11, p. 39] is a Hausdorff space that is locally Euclidean of dimension n. I.e. the space is completely separable, and since it is locally Euclidean, it possesses an integral form that is invariant.

Remark 2.1. The *dimension* of a manifold is determined by the number of independent parameters needed to specify a point.

1-manifold One parameter, e.g. curve, circle. This also applies to curves in space. You can write $(x,y,z) = (f(t),g(t),h(t))$ for continuous functions f,g,h.

2-manifold Two parameters, e.g. surface, sphere, torus, cylinder.

A *differentiable manifold* allows to examine derivatives etc.

Definition 2.7 (differentiable manifold)**.** A *differentiable manifold* consists of a topological manifold and a C^k atlas.

Definition 2.8 (smooth manifold)**.** A *smooth manifold* is a differentiable manifold with a C^∞ atlas.

A *Riemannian manifold* (M,g) is a manifold M with a metric g which allows for measurements on the manifold independent of any embeddings.

Definition 2.9 (Riemannian manifold)**.** A *Riemannian manifold* [Lee11, p. 9] is a manifold on which there is a rule for measuring distances and angles, subject to certain natural restrictions to ensure that these quantities behave analogously to their Euclidean counterparts.

2.3.1 Tangent Bundles

At each point of an n-dimensional differentiable manifold, one can find a *tangent space* consisting of all possible derivative vectors at that point.

A *tangent bundle* is the union of all tangent spaces of a manifold. It can be obtained by taking all tangent spaces and arranging them in a smooth and non-overlapping manner.

Definition 2.10 (tangent bundle)**.** The collection of all tangent spaces T_pM at all points p of an n-dimensional manifold M can be turned into another, 2n-dimensional manifold, the *tangent bundle* TM.

$$TM = \bigsqcup_{p\in M} T_pM = \bigcup_{p\in M} \times T_pM\,.$$

TM is a pair $(p,\vec{v})$, where p is a point of M and $\vec{v}$ is a tangent to M at p.

2.3.2 Parametrisation

Definition 2.11 (parametrisation)**.** In differential geometry, one usually looks at the parametric forms of curves and surfaces rather than their implicit forms.

A *parametric curve* $\mathbf{C}$ [Far02a, p. 179] can be given as

$$\mathbf{C} = \mathbf{C}(t) = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}, t \in [a,b] \subset \mathbb{R}.$$

It is *regular* iff $\dot{\mathbf{C}}(t) \neq 0$.

A *parametric surface* S [Far02a, p. 349] can be given as

$$\mathbf{S} = \mathbf{S}(u,w) = \begin{pmatrix} x(u,w) \\ y(u,w) \\ z(u,w) \end{pmatrix}, \mathbf{u} = \begin{pmatrix} u \\ w \end{pmatrix} \in [a,b] \times [c,d] \subset \mathbb{R}^2.$$

It is *regular* iff $\mathbf{S}_u \times \mathbf{S}_w \neq \mathbf{0}$.

A *parametrised C^r surface* is a C^r-differentiable mapping $\mathbf{S} : U \to \mathbb{E}^3$ of an open domain $U \subset \mathbb{E}^2$ into the Euclidean space $\mathbb{E}^3$, whose differential $\mathrm{d}\mathbf{S}$ is one-to-one for each $\mathbf{q} \in U$.

Remark 2.2.

(a) A change of variables of $\mathbf{S}$ is a diffeomorphism $\tau : \tilde{U} \to U$, where $\tilde{U}$ is an open domain in $\mathbb{E}^2$, such that τ's differential $\mathrm{d}\tau$ always has rank $= 2$, if the determinant of its Jacobian matrix $\det(\tau^*)$ is positive and orientation-preserving.

(b) Relationship: the change of variables defines an equivalence relation on the class of all parametrised surfaces. An equivalence class of parametrised surfaces is called a surface in $\mathbb{E}^3$.

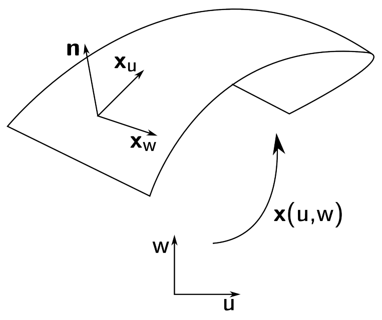


Figure 2.3: Standard surface parametrised with u and w.

Definition 2.12 (derivative)**.** For a curve $\mathbf{C}(t)$, derivatives with respect to the parameter t are written as $\dot{\mathbf{C}}(t), \ddot{\mathbf{C}}(t)$ [Far02a, p. 179].

$$\dot{\mathbf{C}}(t) = \begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \end{pmatrix}.$$

For a surface, the first derivatives are written as $\mathbf{S}_u(u,w), \mathbf{S}_w(u,w)$, e.g.

$$\mathbf{S}_u(u,w) = \begin{pmatrix} x_u(u,w) \\ y_u(u,w) \\ z_u(u,w) \end{pmatrix} .$$

The second derivatives are $\mathbf{S}_{uu}(u,w), \mathbf{S}_{uw}(u,w), \mathbf{S}_{wu}(u,w)$ and $\mathbf{S}_{ww}(u,w)$, where $\mathbf{S}_{uw}(t)$ is first derived in u-direction, and then in w-direction.

In the remainder of this thesis, the following shorthand is used:

$$\mathbf{S}_u := \frac{\partial \mathbf{S}}{\partial u}, \mathbf{S}_w := \frac{\partial \mathbf{S}}{\partial w}, \mathbf{S}_{uv} := \frac{\partial^2 \mathbf{S}}{\partial u \partial w},$$

or alternatively,

$$\mathbf{S}_i, \mathbf{S}_j, \mathbf{S}_{ij}, \qquad i,j \in \{u,w\}.$$

The differential $\partial \mathbf{S}$ is one-to-one if and only if $\frac{\partial \mathbf{S}}{\partial u}$ and $\frac{\partial \mathbf{S}}{\partial w}$ are linearly independent.

Remark 2.3 (symmetry of second derivatives). It is possible to change the order of partial derivatives [Wik12, Satz_von_Schwarz], i.e.

$$\mathbf{S}_{uw}(t) = \mathbf{S}_{wu}(t)$$

holds.

Definition 2.13 (arc length)**.** A curve's *arc length parametrisation* [Far02a, p. 180]

$$s = s(t) = \int_a^t ||\dot{\mathbf{x}}|| \mathrm{d}t$$

is independent of any regular parametrisation (i.e. $\dot{\mathbf{x}}(t) \neq 0$) because

$$\dot{\mathbf{x}} \mathrm{d}t = \frac{\partial \mathbf{x}}{\partial u} \frac{\partial u}{\partial t} \partial t = \frac{\partial \mathbf{x}}{\partial u} \partial u$$

holds. $\mathrm{d}s = ||\dot{\mathbf{x}}|| \mathrm{d}t$ is called the *arc element* of the curve.

The derivative with respect to the arc length of a curve is denoted by a prime: $\mathbf{x}', u', w'$.

2.4 Moving Frames

The moving frames are a good indicator of a curve's or surface's properties. They move along the curve, or over the surface and change depending on local properties.

Definition 2.14 (Frenet frame)**.** The *Frenet frame* [Far02a, p. 181] of a curve consists of the three vectors $\mathbf{t}, \mathbf{m}, \mathbf{b}$, where

$$\mathbf{t} = \frac{\dot{\mathbf{x}}}{||\dot{\mathbf{x}}||} \qquad \text{is the } \textit{tangent} \text{ vector,}$$

$$\mathbf{m} = \mathbf{b} \times \mathbf{t} \qquad \text{is the } \textit{main normal} \text{ vector, and}$$

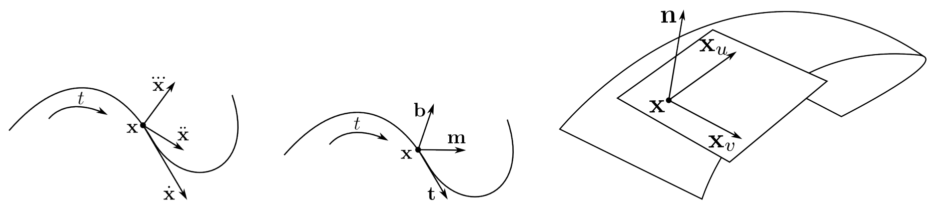
$$\mathbf{b} = \frac{\dot{\mathbf{x}} \times \ddot{\mathbf{x}}}{||\dot{\mathbf{x}} \times \ddot{\mathbf{x}}||} \qquad \text{is the } \textit{binormal} \text{ vector.}$$

Since the vectors are orthonormal (orthogonal to each other and normalised), a local orthonormal system is obtained.

Definition 2.15 (Frenet-Serret formulae)**.** The *Frenet frame* yields the *Frenet-Serret formulae* [Far02a, p. 183]:

$$\begin{array}{rccc} \mathbf{t}' = & & +\kappa\mathbf{m} & \\ \mathbf{m}' = & -\kappa\mathbf{t} & & +\tau\mathbf{b} \\ \mathbf{b}' = & & -\tau\mathbf{m} & \end{array}$$

Definition 2.16 (Gauß frame)**.** For a given surface, the vectors $\mathbf{x}_u, \mathbf{x}_v$ span the tangent plane to the surface $\mathbf{S}$ at $\mathbf{x}$. Their cross product $\mathbf{x}_u \times \mathbf{x}_v$ coincides with the surface normal at $\mathbf{x}$. The *Gauß frame* [Far02a] of a surface consists of the three vectors $\mathbf{x}_u, \mathbf{x}_v, \mathbf{n}$, where $\mathbf{n} = \frac{\mathbf{x}_u \times \mathbf{x}_v}{||\mathbf{x}_u \times \mathbf{x}_v||}$ is the *unit normal field*. Since $\mathbf{x}_u, \mathbf{x}_v$ are not normalised, one can only obtain a local affine system which is dependent on the parametrisation.



(a) Local affine frame for curves. (b) Frenet frame for curves (c) Gauß frame for surfaces.

Figure 2.4: Moving frames for curves and surfaces.

2.5 Curvature and Torsion

The *curvature* describes the curviness of a curve at a given point.

Definition 2.17 (curvature)**.** The *curvature* [Far02a, p. 184] is the change of direction between two tangent vectors $\mathbf{t}(s)$ and $\mathbf{t}(s+\Delta s)$.

$$\kappa = \kappa(t) = \frac{||\dot{\mathbf{x}} \times \ddot{\mathbf{x}}||}{||\dot{\mathbf{x}}||^3}.$$

Alternatively, it can also be computed from the arc length parametrisation:

$$\kappa = \kappa(s) = ||\mathbf{x}''|| = \frac{\partial \varphi}{\partial s},$$

where φ is the angle between $\mathbf{t}(s)$ and $\mathbf{t}(s+\Delta s)$.

At the point $\mathbf{p}$ (at parameter t), an *osculating circle* can be drawn. This is a circle with radius $r = \frac{1}{\kappa(t)}$, which has the same curvature at $\mathbf{p}$, and which has a tangent that is tangential to the curve.

Example 2.2. The circle has a constant curvature. It is always $\kappa = \frac{1}{r}$. This is obvious, because the osculating circle is identical to the curve itself at every point.

The *torsion* describes, how much a curve twists out of the plane at a given point. Torsion and curvature of a spatial curve are equivalent to the curvature of a planar curve.

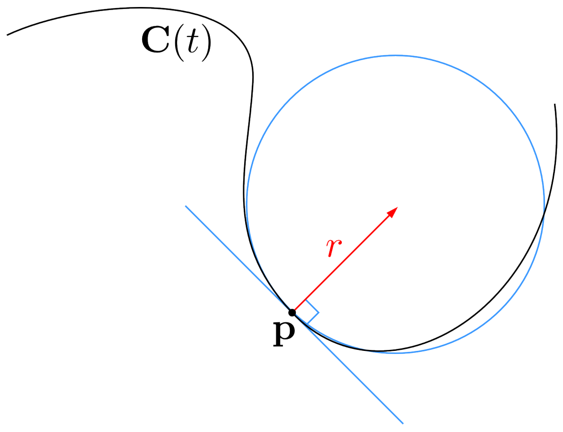


Figure 2.5: A curve $\mathbf{C}(t)$ with an osculating circle at $\mathbf{p}$. [Wik12, Osculating_circle]

Definition 2.18 (torsion)**.** The *torsion* [Far02a] is the change of direction between two binormal vectors $\mathbf{b}(s)$ and $\mathbf{b}(s+\Delta s)$.

$$\tau = \tau(t) = \frac{\det[\dot{\mathbf{x}}(t), \ddot{\mathbf{x}}(t), \dddot{\mathbf{x}}(t)]}{||\dot{\mathbf{x}}(t) \times \ddot{\mathbf{x}}(t)||}.$$

Alternatively, it can also be computed from the arc length parametrisation:

$$\tau = \tau(s) = \frac{1}{\kappa^2} \det[\mathbf{x}', \mathbf{x}'', \mathbf{x}'''] = -\frac{\partial \vartheta}{\partial s},$$

where ϑ is the angle between $\mathbf{t}(s)$ and $\mathbf{t}(s+\Delta s)$.

We can define a tangent plane which is spanned by the tangents of the surface. This tangent plane, in conjunction with the surface normal, defines a local coordinate system on the manifold.

Definition 2.19.

(a) The tangent plane is a two-dimensional linear subspace $T_{\mathbf{u}}\mathbf{S}$ of $\mathbb{E}^3$ generated by $\operatorname{span}\{\mathbf{S}_u, \mathbf{S}_w\}$, and it is called the *tangent space of* $\mathbf{S}$ *at* $\mathbf{u} = (u,w) \in U$.

(b) Elements of $T_{\mathbf{u}}\mathbf{S}$ are called *tangent vectors*.

(c) The vector field $\mathbf{n} := \frac{[\mathbf{S}_u, \mathbf{S}_w]}{||[\mathbf{S}_u, \mathbf{S}_w]||}$, where $[.,.]$ is the cross product, is called a *unit normal field*.

(d) The map $\mathbf{n} : U \to \mathbf{S}^2 \subset \mathbb{E}^3$ is called *Gauß map*, and the moving frame $\{\mathbf{S}_u, \mathbf{S}_w, \mathbf{n}\}$ is called the *Gauß frame* of the surface as displayed in Figure 2.6.

Some properties of the surface can be determined using the first and second fundamental forms. The first fundamental form allows to make measurements on the surface: lengths of curves, angles between tangent vectors, areas of regions, etc. without referring back to the ambient space $\mathbb{E}^3$.

Definition 2.20 (First fundamental form)**.** Let $\mathbf{S} : U \to \mathbb{E}^3$ be a surface. The bilinear form of $T_{\mathbf{u}}\mathbf{S}$ induced by the scalar product $\langle \cdot, \cdot \rangle$ of $\mathbb{E}^3$ by restriction is called the *first fundamental form* $I_{\mathbf{u}}$ of the surface [Far02a, p. 350][Kre68, p. 68].

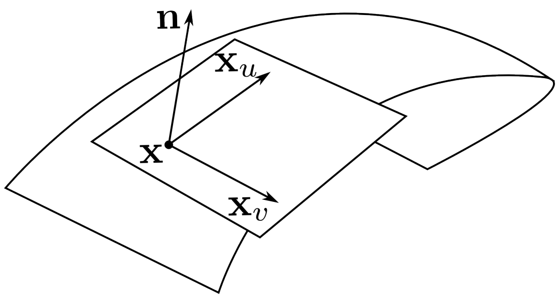


Figure 2.6: Gauß frame $\{\mathbf{S}_u, \mathbf{S}_w, \mathbf{n}\}$ for the surface S.

Remark 2.4. Properties of the first fundamental form:

(a) The matrix representation of the first fundamental form with respect to the basis $\{\mathbf{S}_u, \mathbf{S}_w\}$ of $T_{\mathbf{u}}\mathbf{S}$ is given by

$$\begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix} = \begin{pmatrix} \langle \mathbf{S}_u, \mathbf{S}_u \rangle & \langle \mathbf{S}_u, \mathbf{S}_w \rangle \\ \langle \mathbf{S}_w, \mathbf{S}_u \rangle & \langle \mathbf{S}_w, \mathbf{S}_w \rangle \end{pmatrix}. \tag{2.1}$$

(b) Let

$$g := \det(g_{ij})$$

denote the determinant of the first fundamental form.

(c) The first fundamental form is symmetric, positive definite, and a geometric invariant.

(d) The derivative of a surface $\mathbf{S}$ with respect to t is
$\dot{\mathbf{S}} = \mathbf{S}_u \dot{u} + \mathbf{S}_w \dot{w}$.
The *first fundamental form* corresponds to the squared arc element $\mathrm{d}s^2$:

$$\begin{aligned} \mathrm{d}s^2 &= ||\dot{\mathbf{S}}||^2 \mathrm{d}t^2 \\ &= (\mathbf{S}_u^2 \dot{u}^2 + 2\mathbf{S}_u \mathbf{S}_w \dot{u}\dot{w} + \mathbf{S}_w^2 \dot{w}^2)\mathrm{d}t^2 \\ &= g_{11}\mathrm{d}u^2 + 2g_{12}\mathrm{d}u\mathrm{d}w + g_{22}\mathrm{d}w^2 , \end{aligned}$$

where

$$g_{11} = \mathbf{S}_u \mathbf{S}_u \,, g_{12} \qquad = \mathbf{S}_u \mathbf{S}_w \,, g_{22} = \mathbf{S}_w \mathbf{S}_w \,.$$

(e)

$$\begin{pmatrix} g_{11} & g_{12} \\ g_{12} & g_{22} \end{pmatrix} \tag{2.2}$$

is called the *metric tensor*, and its inverse,

$$\begin{pmatrix} g^{11} & g^{12} \\ g^{21} & g^{22} \end{pmatrix} = \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix}^{-1} = \frac{1}{\det(g_{jk})} \begin{pmatrix} g_{22} & -g_{12} \\ -g_{21} & g_{11} \end{pmatrix}. \tag{2.3}$$

is called the *conjugate metric tensor* [Kre68, p. 113].

Definition 2.21 (Second fundamental form)**.** Let $\mathbf{u}(t)$ define a curve on the surface $\mathbf{S}(\mathbf{u})$. We know that $\mathbf{t}' = \kappa\mathbf{m}$ and $\mathbf{t} = \mathbf{S}', u' = \frac{\partial u}{\partial s}, w' = \frac{\partial w}{\partial s}$, therefore

$$\mathbf{t}' = \mathbf{S}'' = \mathbf{S}_{uu}(u')^2 + 2\mathbf{S}_{uw}u'w' + \mathbf{S}_{ww}(w')^2 + \mathbf{S}_u u'' + \mathbf{S}_w w''\,.$$

Let φ denote the angle between the main normal $\mathbf{m}$ and the surface normal $\mathbf{n}$. Then

$$\kappa\cos\varphi = \kappa\mathbf{mn} \overset{\mathbf{t}'=\kappa\mathbf{m}}{=} \mathbf{t}'\mathbf{n} = \mathbf{nS}_{uu}(u')^2 + 2\mathbf{nS}_{uw}u'w' + \mathbf{nS}_{ww}(w')^2 + \underbrace{\mathbf{nS}_u}_{=0} u'' + \underbrace{\mathbf{nS}_w}_{=0} w''\,.$$

Furthermore, $\mathbf{nS}_u = 0$ implies $\mathbf{n}_u\mathbf{S}_u + \mathbf{nS}_{uu} = 0$ etc., thus,

$$\begin{aligned} h_{11} &= -\mathbf{S}_u\mathbf{n}_u && = \mathbf{nS}_{uu}\,, \\ h_{12} &= -\frac{1}{2}(\mathbf{S}_u\mathbf{n}_w + \mathbf{S}_w\mathbf{n}_u) && = \mathbf{nS}_{uw}\,, \\ h_{22} &= -\mathbf{S}_w\mathbf{n}_w && = \mathbf{nS}_{ww}\,. \end{aligned}$$

The *second fundamental form* [Far02a, p. 352ff] corresponds to $\kappa\cos\varphi \mathrm{d}s^2$:

$$\kappa\cos\varphi\mathrm{d}s^2 = h_{11}\mathrm{d}u^2 + 2h_{12}\mathrm{d}u\mathrm{d}w + h_{22}\mathrm{d}w^2\,.$$

The second fundamental form permits to study surface curvature and torsion. One especially interesting consequence of the second fundamental form can be found in the Weingarten equations which will prove useful when considering the main theorem of this paper.

Definition 2.22. Let $\mathbf{S} : U \to \mathbb{E}^3$ be a surface and $\mathbf{u} \in U$.

(a) The linear map $L : T_\mathbf{u}\mathbf{S} \to T_\mathbf{u}\mathbf{S}$ defined by $L := -\mathrm{d}\mathbf{n}_\mathbf{u} \cdot \mathrm{d}\mathbf{S}_\mathbf{u}$ is called the *Weingarten map*, or *shape operator*.

(b) The bilinear form $I\!I_\mathbf{u}$ defined by $I\!I_\mathbf{u}(A,B) := \langle L(A), B\rangle$ for each $A, B \in T_\mathbf{u}\mathbf{S}$ is called the *second fundamental form* of the surface.

Remark 2.5. Properties of the second fundamental form:

(a) The matrix representation of $I\!I_\mathbf{u}$ with respect to the canonical basis $\{\mathbf{e}_1, \mathbf{e}_2\}$ of $T_\mathbf{u}\mathbb{E}^2$ (identified with $\mathbb{E}^2$) and the associated basis $\{\mathbf{S}_u, \mathbf{S}_w\}$ of $T_\mathbf{u}\mathbf{S}$ is given by

$$\begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} = \begin{pmatrix} \langle -\mathbf{n}_u, \mathbf{S}_u\rangle & \langle -\mathbf{n}_u, \mathbf{S}_w\rangle \\ \langle -\mathbf{n}_w, \mathbf{S}_u\rangle & \langle -\mathbf{n}_w, \mathbf{S}_w\rangle \end{pmatrix} = \begin{pmatrix} \langle \mathbf{n}, \mathbf{S}_{uu}\rangle & \langle \mathbf{n}, \mathbf{S}_{uw}\rangle \\ \langle \mathbf{n}, \mathbf{S}_{wu}\rangle & \langle \mathbf{n}, \mathbf{S}_{ww}\rangle \end{pmatrix}, \tag{2.4}$$

i.e.

$$h_{ij} := \langle -\mathbf{n}_i, \mathbf{S}_j\rangle = \langle \mathbf{n}, \mathbf{S}_{ij}\rangle\,.$$

One can assume that $h_{12} = h_{21}$ since the surfaces under consideration are C^r-continuous.

(b) Let

$$h := \det(h_{ij})$$

denote the determinant of the second fundamental form.

(c) We call two geometric objects *congruent* to each other iff there is an isometric transformation (i.e. only translation, rotation, and reflection are employed) from one to the other. Congruences preserve lengths and angles.

(d) The second fundamental form is invariant under congruences of $\mathbb{E}^3$ and orientation-preserving changes of variables.

Definition 2.23 (principal curvatures)**.** The minimal and maximal curvatures κ_1, κ_2 at a point are called the *principal curvatures* of a surface at $\mathbf{p}$.

The *Gauß curvature* $K = \kappa_1 \kappa_2$ is computed as

$$\kappa_1 \kappa_2 = \frac{h_{11}h_{22} - h_{12}^2}{g_{11}g_{22} - g_{12}^2}.$$

The *mean curvature* is computed as $H = \frac{1}{2}(\kappa_1 + \kappa_2)$, where

$$\kappa_1 + \kappa_2 = \frac{h_{22}g_{11} - 2h_{12}g_{12} + h_{11}g_{22}}{g_{11}g_{22} - g_{12}^2}.$$

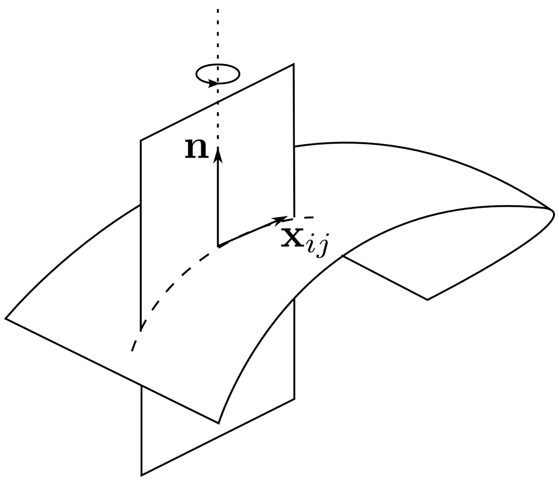


Figure 2.7: κ_1, κ_2 are determined by rotating a plane around a point's normal vector and finding the minimal and maximal curvature at the given point.

Definition 2.24 (point characteristics)**.** Depending on the signs of κ_1, κ_2, the point on the surface has different characteristics.

$K > 0$	*elliptic point*	κ_1 and κ_2 have the same sign.
$K < 0$	*hyperbolic point*	κ_1 and κ_2 have different signs.
$K = 0, H \neq 0$	*parabolic point*	either $\kappa_1 = 0$ or $\kappa_2 = 0$
$K = 0, H = 0$	*flat point*	$\kappa_1 = \kappa_2 = 0$

Curvature is of great interest in the context of differential geometry. The minimal and maximal curvatures κ_1, κ_2 at a surface point are the basis for the more interesting definitions of mean curvature and Gauß curvature. In Chapter 3, I examine total curvature of surfaces under deformation in normal direction.

Considering surface curves, one can derive the geometric interpretations of the second fundamental form:

Let $\mathbf{e} := \lambda^1 \mathbf{S}_u + \lambda^2 \mathbf{S}_w$ be a tangent vector with $\|\mathbf{e}\| = 1$. If the intersection of a surface with a plane is given by $\mathbf{n}$ and $\mathbf{e}$, an intersection curve $\mathbf{C}_y$ emerges with the following properties:

$$\mathbf{C}_y'(s) = \mathbf{e} \text{ and } \mathbf{e}_2 = \pm\mathbf{n},$$

where $\mathbf{e}_2$ is the principal normal vector of the space curve $\mathbf{C}_y$.

The implicit function theorem implies the existence of this so-called *normal section curve*. To calculate the minimal and maximal curvature of a normal section curve (the so-called *normal section curvature*), one can use the method of *Lagrange multipliers*.

As a result of these considerations, one can define various notions of curvature:

Definition 2.25. Let $\mathbf{S} : U \to \mathbb{E}^3$ be a surface and $\mathbf{e} = \lambda^1 \mathbf{S}_u + \lambda^2 \mathbf{S}_w$ a tangent vector of $\mathbf{S}$ at $\mathbf{u}$.

(a) The Weingarten map L is self-adjoint.

(b) The *normal section curvature* $\kappa_{\mathbf{n}}(\lambda^1, \lambda^2)$ can be computed as:

$$\kappa_{\mathbf{n}}(\lambda^1, \lambda^2) = \frac{h_{ij}\lambda^i\lambda^j}{g_{ij}\lambda^i\lambda^j}\,.$$

Unless the normal section curvature is the same for all directions (umbilical points), there are two perpendicular directions $\mathbf{e}_1$ and $\mathbf{e}_2$, in which $\kappa_{\mathbf{n}}$ attains its absolute maximum and its absolute minimum.

(c) $\mathbf{e}_1$ and $\mathbf{e}_2$ are the *principal directions*.

(d) The corresponding normal section curvatures, κ_1 and κ_2, are called *principal curvatures* of the surface.

(e) Let $\mathbf{S} : U \to \mathbb{E}^3$ be a surface and $\mathbf{C}_y : I \to \mathbb{E}^3$ be a surface curve. We denote by $\hat{\mathbf{C}}_y(t)$ the orthogonal projection of $\mathbf{C}_y(t)$ on the tangent plane $T_{\mathbf{u}}\mathbf{S}$ at (an arbitrary) point $\mathbf{p} := \mathbf{S}(\mathbf{u})$. The *geodesic curvature* κ_g of y at $\mathbf{p}$ is defined as the curvature of the projected curve $\hat{\mathbf{C}}_y(t)$ at $\mathbf{p}$. A curve $\mathbf{C}_y(t)$ on a surface $\mathbf{S}$ is called *geodesic* if its geodesic curvature κ_g vanishes identically.

(f) $\kappa_g = \det(\dot{\mathbf{C}}_y, \ddot{\mathbf{C}}_y, \mathbf{n})$, where dots denote derivatives with respect to the arc length of $\mathbf{C}_y$.

(g) $H := \operatorname{trace}(L) = \frac{1}{2} \cdot (\kappa_1 + \kappa_2)$ is called the *mean curvature*.

(h) $K := \kappa_1 \cdot \kappa_2 = \det(L) = \frac{\det(II)}{\det(I)}$ is called the *Gauß curvature*.

(i) *Total Gauß curvature*, or short, *total curvature*, is defined as $K_{\text{tot}} = \iint_{\mathbf{S}} K \mathrm{d}\mathbf{S}$.

Remark 2.6 (Geodesics and curvature). (a) An arc of minimum length on a surface joining two arbitrary points must be an arc of a geodesic.

(b) Assuming the boundary of a surface is given, and a surface patch of minimal area has to be fit, the minimal curvature of this patch has to vanish. In this case, the mean curvature $H \equiv 0$ will also vanish.

Definition 2.26 (Bending Energy)**.** Do Carmo [Do 76, p. 307] defined the *energy of a curve* as

$$E_{\mathbf{C}}(\varphi) = \int_a^b |\varphi(t)|^2 \partial t\,.$$

Similarly, based on the notion of curvature, I can define *bending energy* of a surface $\mathbf{S}$ as follows

$$E(\mathbf{S}) = \int_{\mathbf{S}} \|\kappa_{\min}\|^2 + \|\kappa_{\max}\|^2 \partial\mathbf{S}\,.$$

2.6 Metrics

Metrics are tools to permit comparisons between elements or sets of elements.

Definition 2.27 (metric)**.** A distance function $d(P,Q)$ is called a *metric* [Kre68, p. 6] if it has the following properties:

1. $d(P,Q)$ is real, finite and non-negative.
2. $d(P,Q) = 0$ iff $P = Q$.
3. $d(P,Q) = d(Q,P)$.
4. $d(P,Q) \leq d(P,R) + d(R,Q)$

where P, Q, R are points.

A very well-known example of a metric is the *Euclidean distance*:

Definition 2.28 (Euclidean distance)**.** For a given vector

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = a_1 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} + a_2 \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = a_1 \cdot \mathbf{x}_u + a_2 \cdot \mathbf{x}_w\,,$$

the Euclidean distance describes the length of the vector as follows:

$$d(\mathbf{a},\mathbf{a}) = \sqrt{\langle \mathbf{a},\mathbf{a} \rangle} = \sqrt{a_1^2 + a_2^2}\,.$$

Once the surface the distance is measured on, is not flat but curved, additional information is needed:

$$\langle \mathbf{a},\mathbf{a} \rangle = \langle a_1\mathbf{x}_u + a_2\mathbf{x}_w, a_1\mathbf{x}_u + a_2\mathbf{x}_w \rangle = a_1^2 \underbrace{\langle \mathbf{x}_u, \mathbf{x}_u \rangle}_{g_{11}} + 2a_1a_2 \underbrace{\langle \mathbf{x}_u, \mathbf{x}_w \rangle}_{g_{12}} + a_2^2 \underbrace{\langle \mathbf{x}_w, \mathbf{x}_w \rangle}_{g_{22}}\,.$$

Remark 2.7 (Euclidean Norm). The Euclidean distance is also known as the *Euclidean Norm*. Given a vector $\mathbf{x} = (x_1, \ldots, x_n)^T$, its norm is denoted by

$$||\mathbf{x}||_2 = \sqrt{\sum_{i=0}^{n} x_i^2}\,.$$

2.7 Morse Theory

The basic concept of Morse theory can be illustrated with a mountainous landscape M that is flooded (since it is a theoretical landscape, the ground is not porous so all water will remain on top). The water level (or *level set*) is a value $a \in \mathbb{R}$ A function $f : M \to \mathbb{R}$ describes the elevation, and $f^{-1}(\mathbb{R})$ corresponds to a contour line. The region covered by water at elevation a corresponds to $f^{-1}(-\infty, a]$. The topology of this region does not change except when a is the height of a *critical point* [EH10].

Definition 2.29 (critical point)**.** A *critical point* is a point at which the gradient $\nabla f = 0$. Each critical point has an *index* that equals the number of independent directions in which f decreases., Peaks, passes, and basins of a landscape have indices 2, 1, and 0 respectively.

Chapter 3

Preserving Total Curvature Under Infinitesimal Deformations

3.1 Motivation

In industrial surface generation, it is important to consider surfaces with minimal areas for two main reasons: these surfaces require less material than non-minimal surfaces, and they are therefore cheaper to manufacture. Based on a prototype, a so-called masterpiece, the final product is created by optimising the surface through small deformations, either manually or automatically.

Automated surface optimisation has the main advantage that it is cheap, fast, and it does not require user intervention. However, it requires that optimisation criteria are defined in advance. Many criteria are based on geometric properties. One such criterion is the preservation of total Gauß curvature. A low Gauß curvature indicates low bending energy. Therefore, it is useful to limit, or even minimise, total Gauß curvature in order to minimise the material cost and waste, as well as bending energy [JS07, GU02].

This chapter is based on the work presented in [BHH15]. I present a global bilinear deformation technique which preserves the total curvature of a masterpiece. In particular, I derive sufficient conditions for these bilinear deformations to be total curvature preserving when applied to an analytically defined masterpiece. Furthermore, I demonstrate how linear deformation can be utilised to minimise total curvature.

Total curvature is used as a tool to restrict deformation by deriving a compatible bilinear deformation function from the analytical surface definition. With the help of this function, the surface is deformed in normal direction in a manner that preserves total curvature. As such, this technique requires an analytically defined surface, with analytically defined normals to be carried out in its pure form. However, it is possible to empirically determine a deformation function for surfaces which are defined as discrete meshes. Since every surface point is moved using the same global criterion, this is a global deformation technique.

In Section 3.2, I give an overview of related work. In Section 3.4, I describe and prove my approach. I apply the technique to a helicoid surface in an example in Section 3.5, In Section 3.6, I perform a case study on a more complex model, during which the deformation is more than just infinitesimal. I deform a part of a fandisk model which is composed of Bézier surface patches using a variety of linear deformation functions. Then, I analyse error for each of the functions. Finally, I conclude the chapter in Section 3.7.

3.2 Related Work

Efimov was the first to introduce partial differential equations as a tool to study infinitesimal bending. He gives an overview of the state of the art of infinitesimal bendings in his textbook [Efi57]. Hagen et al. [HH98] visualise the momentarial rotation field that is associated with infinitesimal bending. They then use the structure of this rotation field as a tool to analyse the deformations that were generated by this bending. Hahmann et al. [HH96] investigate numerical aspects of discretising the deformation vector field. Ivanova and Subitov [IKS95] examine infinitesimal bendings of surfaces of revolution and of polyhedra. Meziani [Mez07] studies infinitesimal bending of homogeneous surfaces that have a flat point and positive curvature.

More recent works on infinitesimal bending for curves and non-parametric surfaces have been published by L. Velimirović et al. They study total mean curvature variation on oriented, boundary-free surfaces [VRZ11], and they visualise changes of bent curves as surfaces constructed from different stages of deformation [VC11]. Eigensatz et al. [ESP08] use curvature as a tool to control surface deformation. They extend this work to allow various user-specified local restrictions on deformation [EP09].

Other works have addressed perturbations preserving the topological form of polyhedra [ADPS95], and deformations preserving ambient isotopy of curves [JMM$^+$08, LP12].

To be applicable to discrete surfaces, the technique presented in this chapter requires discrete curvature measures. Dyn et al. [DHKL01] compute *discrete total Gauß curvature* and absolute discrete total mean curvature with the goal of mesh optimisation. Various other scientists have extended this work to include an area-based weighting [LP82, Boi95, KKL02, MDSB03, LLV05, Xu06]. Kerautret et al. [KLN08] compare different discrete curvature estimators for curves with the goal of detecting corners. They compare osculating circle curvature estimators [CMT01], global minimisation curvature estimators [KL08], and binomial convolution curvature estimators with each other. Bousquet [Bou97] introduced discrete mean curvature for an edge. Others have extended this definition to vertices [LBS05], and to neighbourhoods with non-uniform edges [LP82, Boi95, KKL02, MDSB03, LLV05].

In this work, rather than studying total curvature changes after bending, or using curvature as a tool to deform surfaces, I employ total curvature as a tool to restrict bending and avoid large changes. I assume a rigid material which can be bent out of shape through exterior deformations, but which cannot be stretched in tangent direction through interior deformations, as common in engineering [BWFS11, BIA12].

3.3 Background

3.3.1 Weingarten Equations

It can be shown that $\langle \mathbf{n}_i, \mathbf{n} \rangle = 0;\ i = 0,1$, where $\mathbf{n}_0 = \mathbf{n}_u, \mathbf{n}_1 = \mathbf{n}_w$ (cf. Equation 2.4). Thus, $\mathbf{n}_i$ can be represented by the local frame of the tangent plane, and the following relation holds

$$\mathbf{n}_i = -\sum_{k=1}^{2} h_i^k \mathbf{S}_k\,, \tag{3.1}$$

where the following equations

$$h_1^1 = \frac{h_{11}g_{22} - h_{12}g_{12}}{g}\,, \qquad h_1^2 = \frac{h_{12}g_{11} - h_{11}g_{12}}{g}\,, \tag{3.2}$$

$$h_2^1 = \frac{h_{12}g_{22} - h_{22}g_{12}}{g}\,, \qquad h_2^2 = \frac{h_{22}g_{11} - h_{12}g_{12}}{g}\,, \tag{3.3}$$

are called *Weingarten equations* [Do 76]. Further, the normals can be expressed in terms of the Gauß frame, and it can be shown that the following relation holds

$$\mathbf{S}_{ij} = h_{ij}\mathbf{n} + \sum_{k=1}^{2} \Gamma_{ij}^k \mathbf{S}_k\,. \tag{3.4}$$

where $\Gamma_{ij}^k = \langle \mathbf{S}_k, \mathbf{S}_{ij} \rangle$ are called the *Christoffel Symbols*.

Definition 3.1 (Christoffel Symbols)**.** Let $\mathbf{S} : \mathbf{u} \to \mathbb{E}^3$ be a surface with

$$\mathbf{S}_{ij} = \frac{\partial^2 \mathbf{S}}{\partial u \partial w} = \Gamma_{ij}^k \mathbf{S}_k + \mathbf{k}_{ij} \cdot \mathbf{n}\,,$$

where $\mathbf{k}_{ij} = \mathbf{S}_{ij} \cdot \mathbf{n}$, and $\mathbf{t}_m$ is a tangent vector of the surface, i.e. $\mathbf{t}_m \cdot \mathbf{n} = 0$.

$$\mathbf{S}_{ij} \cdot \mathbf{t}_m = \Gamma_{ij}^k \mathbf{S}_k \cdot \mathbf{t}_m + \mathbf{k}_{ij} \underbrace{\mathbf{n} \cdot \mathbf{t}_m}_{=0} \qquad = \Gamma_{ij}^k \mathbf{S}_k \cdot \mathbf{t}_m = \Gamma_{ij}^k g_{km}\,.$$

The *Christoffel symbols* Γ_{ijm} *of the first kind* [Kre68, p. 127] can be defined as

$$\Gamma_{ijm} = \langle \mathbf{S}_{ij}, \mathbf{t}_m \rangle \qquad = g_{km}\Gamma_{ij}^k = \frac{1}{2}\left(\frac{\partial g_{ij}}{\partial u^m} + \frac{\partial g_{jm}}{\partial u^i} + \frac{\partial g_{mi}}{\partial u^j} \right)\,.$$

Thus, the *Christoffel symbols* Γ_{ij}^l *of the second kind* can be defined as

$$\Gamma_{ij}^l = g^{lm}\Gamma_{ijm}\,.$$

Remark 3.1. Christoffel symbols can *not* be interpreted as tensors.

3.4 Deformations

Let $\mathbf{S}(u,w)$ be the masterpiece of an industrial surface. Let it further be a minimal surface (i.e. $H \equiv 0$), such that it covers a minimal area. This masterpiece should be deformed along its normal direction $\mathbf{n}(u,w)$ by applying a deformation function $f(u,w)$

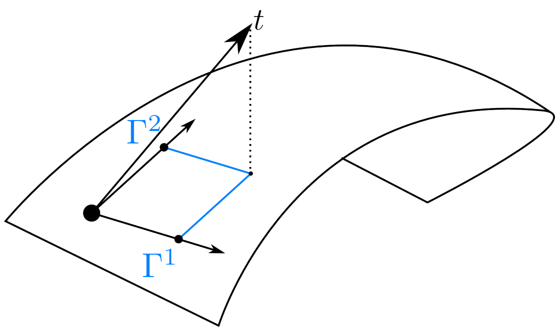


Figure 3.1: Christoffel symbols Γ^i for the derivative of the tangent vector $\mathbf{t}$.

($f : U \to \mathbb{E}$). Deformations along the normal mean that interior deformations of the surface are not permitted, i.e. inner bending is not possible.

I consider bilinear deformations of the form

$$\tilde{\mathbf{S}}(u,w,t) := \mathbf{S}(u,w) + t \cdot f(u,w) \cdot \mathbf{n}(u,w)\,, \tag{3.5}$$

for $t \in (-\varepsilon, \varepsilon)$, $\tilde{g} = g + o(t^2)$, such that $o(t^2)$ constitutes an infinitesimal change. The more general case of bilinear deformations

$$\tilde{\mathbf{S}}(u,w,t) = \mathbf{S}(u,w) + t \cdot Z(u,w)\,, \tag{3.6}$$

where $Z(u,w)$ is a continuous vector field ($Z : U \to \mathbb{E}^3$), is called an infinitesimal bending if $\partial s_t^2 = \partial s^2 + o(t^2)$, i.e. the difference of the squares of the line elements of these surfaces has at least second order [Efi57, HH96, HH98].

Let me first prove two properties of minimal surfaces. These properties will be needed to prove Theorem 3.1.

Lemma 3.1. For a minimal surface $\mathbf{S}(u,w)$, i.e. a surface with $H \equiv 0$, the following statements hold

(a) $[\mathbf{S}_u, \mathbf{n}_w] + [\mathbf{n}_u, \mathbf{S}_w] = 0\,,$

(b) $\langle \mathbf{n}, h_{11}\mathbf{n}_{ww} + h_{22}\mathbf{n}_{uu} - h_{12}\mathbf{n}_{uw} - h_{12}\mathbf{n}_{wu} \rangle = 0\,,$

where $\mathbf{n} = \frac{[\mathbf{S}_u,\mathbf{S}_w]}{g}$ for $g = \det\begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix} = g_{11}g_{22} - g_{12}^2\,.$

I will prove part (a) and part (b) of this Lemma separately.

(a) *Proof.* To prove Lemma 3.1a, I can expand Equation 3.1 to

$$\mathbf{n}_u = \frac{h_{12}g_{12} - h_{11}g_{22}}{g}\mathbf{S}_u + \frac{h_{11}g_{12} - h_{12}g_{11}}{g}\mathbf{S}_w\,, \tag{3.7}$$

$$\mathbf{n}_w = \frac{h_{22}g_{12} - h_{12}g_{22}}{g}\mathbf{S}_u + \frac{h_{12}g_{12} - h_{22}g_{11}}{g}\mathbf{S}_w\,, \tag{3.8}$$

from which I can immediately conclude the assumption:

$$\begin{aligned}
[\mathbf{S}_u,\mathbf{n}_w]+[\mathbf{n}_u,\mathbf{S}_w] &= \frac{h_{12}g_{12}-h_{22}g_{11}}{g}[\mathbf{S}_u,\mathbf{S}_w]+\frac{h_{12}g_{12}-h_{11}g_{22}}{g}[\mathbf{S}_u,\mathbf{S}_w] \\
&= (h_{12}g_{12}-h_{22}g_{11}+h_{12}g_{12}-h_{11}g_{22})\cdot\frac{[\mathbf{S}_u,\mathbf{S}_w]}{g} \\
&= -(h_{11}g_{22}+h_{22}g_{11}-2h_{12}g_{12})\cdot\mathbf{n} \\
&= -\frac{h_{11}g_{22}+h_{22}g_{11}-2h_{12}g_{12}}{g}\cdot g\cdot\mathbf{n} \\
&= -(\kappa_1+\kappa_2)\cdot g\cdot\mathbf{n} \\
&= -2H\cdot g\cdot\mathbf{n}=0\,.
\end{aligned}$$

□

(b) *Proof.* To prove Lemma 3.1b, I first compute the second derivatives of **n**. Then, using the Weingarten equations, I can conclude the following relations:

$$\begin{aligned}
\mathbf{n}_{uu} &= -\frac{\partial h_1^1}{\partial u}\mathbf{S}_u - h_1^1\mathbf{S}_{uu} - \frac{\partial h_1^2}{\partial u}\mathbf{S}_w - h_1^2\mathbf{S}_{wu}\,, \\
\mathbf{n}_{ww} &= -\frac{\partial h_2^1}{\partial w}\mathbf{S}_u - h_2^1\mathbf{S}_{uw} - \frac{\partial h_2^2}{\partial w}\mathbf{S}_w - h_2^2\mathbf{S}_{ww}\,, \\
\mathbf{n}_{uw} &= -\frac{\partial h_1^1}{\partial w}\mathbf{S}_u - h_1^1\mathbf{S}_{uw} - \frac{\partial h_1^2}{\partial w}\mathbf{S}_w - h_1^2\mathbf{S}_{ww}\,, \\
\mathbf{n}_{wu} &= -\frac{\partial h_2^1}{\partial u}\mathbf{S}_u - h_2^1\mathbf{S}_{uu} - \frac{\partial h_2^2}{\partial u}\mathbf{S}_w - h_2^2\mathbf{S}_{wu}\,.
\end{aligned}$$

Next, I look at the scalar product of the normal vector and its second partial derivatives. From this computation, I receive all basic components needed to express part of the formula given in Lemma (3.1)(b):

$$\begin{aligned}
\langle\mathbf{n},\mathbf{n}_{uu}\rangle &= -h_1^1\langle\mathbf{n},\mathbf{S}_{uu}\rangle - h_1^2\langle\mathbf{n},\mathbf{S}_{wu}\rangle &&= -h_1^1h_{11}-h_1^2h_{12} \\
\langle\mathbf{n},\mathbf{n}_{ww}\rangle &= -h_2^1\langle\mathbf{n},\mathbf{S}_{uw}\rangle - h_2^2\langle\mathbf{n},\mathbf{S}_{ww}\rangle &&= -h_2^1h_{12}-h_2^2h_{22} \\
\langle\mathbf{n},\mathbf{n}_{uw}\rangle &= -h_1^1\langle\mathbf{n},\mathbf{S}_{uw}\rangle - h_1^2\langle\mathbf{n},\mathbf{S}_{ww}\rangle &&= -h_1^1h_{12}-h_1^2h_{22} \\
\langle\mathbf{n},\mathbf{n}_{wu}\rangle &= -h_2^1\langle\mathbf{n},\mathbf{S}_{uu}\rangle - h_2^2\langle\mathbf{n},\mathbf{S}_{wu}\rangle &&= -h_2^1h_{11}-h_2^2h_{12}\,.
\end{aligned}$$

I want to show that $\langle \mathbf{n}, h_{11}\mathbf{n}_{ww} + h_{22}\mathbf{n}_{uu} - h_{12}\mathbf{n}_{uw} - h_{12}\mathbf{n}_{wu}\rangle = 0$. Taking the above results, combined with Equation 3.2, I arrive at

$$
\begin{aligned}
&\langle \mathbf{n}, h_{11}\mathbf{n}_{ww} + h_{22}\mathbf{n}_{uu} - h_{12}\mathbf{n}_{uw} - h_{12}\mathbf{n}_{wu}\rangle \\
&= \langle \mathbf{n}, h_{11}\mathbf{n}_{ww}\rangle + \langle \mathbf{n}, h_{22}\mathbf{n}_{uu}\rangle - \langle \mathbf{n}, h_{12}\mathbf{n}_{uw}\rangle - \langle \mathbf{n}, h_{12}\mathbf{n}_{wu}\rangle \\
&= +h_{12}h_1^1\langle \mathbf{n}, \mathbf{S}_{uw}\rangle + h_{12}h_1^2\langle \mathbf{n}, \mathbf{S}_{ww}\rangle + h_{12}h_2^1\langle \mathbf{n}, \mathbf{S}_{uu}\rangle + h_{12}h_2^2\langle \mathbf{n}, \mathbf{S}_{wu}\rangle \\
&\quad - h_{11}h_2^1\langle \mathbf{n}, \mathbf{S}_{uw}\rangle - h_{11}h_2^2\langle \mathbf{n}, \mathbf{S}_{ww}\rangle - h_{22}h_1^1\langle \mathbf{n}, \mathbf{S}_{uu}\rangle - h_{22}h_1^2\langle \mathbf{n}, \mathbf{S}_{wu}\rangle \\
&= -h_{11}h_2^1h_{12} - h_{11}h_2^2h_{22} - h_{22}h_1^1h_{11} - h_{22}h_1^2h_{12} \\
&\quad + h_{12}h_1^1h_{12} + h_{12}h_1^2h_{22} + h_{12}h_2^1h_{11} + h_{12}h_2^2h_{12} \\
&= -h_{11}h_{22}(h_1^1 + h_2^2) + (h_{12})^2(h_1^1 + h_2^2) \\
&= (h_{11}h_{22} - (h_{12})^2)(-h_2^2 - h_1^1) \\
&= \frac{h}{g}(h_{12}g_{12} - h_{11}g_{22} + h_{12}g_{12} - h_{22}g_{11}) \\
&= \frac{h}{g}(-2Hg) \\
&= -2hH = 0\,.
\end{aligned}
$$

□

I am interested in shape-preserving modification of the masterpiece. I consider infinitesimal deformations which do not change the Gauß curvature, and which therefore preserve the total curvature of the surface. A further advantage of infinitesimal deformations is that the genus of a model will be preserved. Large deformations can introduce self-intersections.

I restrict myself to exterior deformations, i.e. deformations in normal direction. Interior deformations, such as perturbations in the tangent plane, are not permitted. Restricting deformation serves the purpose of exaggerating or reducing features that are present in the masterpiece but refraining from introducing additional perturbations. This leads to the main theorem of this chapter:

Theorem 3.1. A linear deformation

$$\tilde{\mathbf{S}}(u,w,t) = \mathbf{S}(u,w) + t\cdot f(u,w)\cdot \mathbf{n}(u,w) \tag{3.9}$$

of a minimal surface $\mathbf{S}(u,w)$ with $t \in (-\varepsilon, \varepsilon)$, and $\tilde{g} = g + o(t^2)$ preserves Gauß curvature if

$$
\begin{aligned}
\mathrm{D}f :&= h_{11}f_{ww} + h_{22}f_{uu} - 2h_{12}f_{uw} - f_u(h_{11}\Gamma_{22}^1 - h_{12}\Gamma_{12}^1 + h_{22}\Gamma_{11}^1)\sqrt{g} \\
&\quad + f_w(h_{11}\Gamma_{22}^2 - h_{12}\Gamma_{12}^2 + h_{22}\Gamma_{11}^2)\sqrt{g} \\
&= 0\,,
\end{aligned}
$$

and therefore preserves the total curvature $\iint_{\mathbf{S}} K\partial s$ of a minimal surface $\mathbf{S}(u,w)$.

$$\tilde{\mathbf{S}}(u,w,t) = \mathbf{S}(u,w) + t\cdot f(u,w)\cdot \mathbf{n}(u,w)\,.$$

Proof. To examine the impact of linear deformation as given in Equation 3.9, I need to observe changes in some surface properties. I start with normal vectors along which I perturb the surface. Normal vectors deform as follows:

$$\tilde{\mathbf{n}}(u,w,t) = \frac{[\mathbf{S}_u, \mathbf{S}_w] + t\cdot[f_w\mathbf{S}_u - f_u\mathbf{S}_w, \mathbf{n}]}{\|[\mathbf{S}_u, \mathbf{S}_w] + t\cdot[f_w\mathbf{S}_u - f_u\mathbf{S}_w, \mathbf{n}]\|} + \mathrm{o}(t^2)\,.$$

The deformed second fundamental form $\widetilde{II}$, defined as

$$\begin{pmatrix} \tilde{h}_{11} & \tilde{h}_{12} \\ \tilde{h}_{12} & \tilde{h}_{22} \end{pmatrix} = \begin{pmatrix} \langle \tilde{\mathbf{n}}, \tilde{\mathbf{S}}_{uu} \rangle & \langle \tilde{\mathbf{n}}, \tilde{\mathbf{S}}_{uw} \rangle \\ \langle \tilde{\mathbf{n}}, \tilde{\mathbf{S}}_{wu} \rangle & \langle \tilde{\mathbf{n}}, \tilde{\mathbf{S}}_{ww} \rangle \end{pmatrix},$$

can be written as

$$\begin{aligned}
\tilde{h}_{11} &= \langle \mathbf{n} + t\cdot[f_w\mathbf{S}_u - f_u\mathbf{S}_w, \mathbf{n}], \mathbf{S}_{uu} + tf_{uu}\mathbf{n} + 2tf_u\mathbf{n}_u + tf\mathbf{n}_{uu}\rangle + o(t^2) \\
&= \langle \mathbf{n}, \mathbf{S}_{uu}\rangle + t\cdot\langle [f_w\mathbf{S}_u + f_u\mathbf{S}_w, \mathbf{n}], \mathbf{S}_{uu}\rangle + tf_{uu}\langle \mathbf{n}, \mathbf{n}\rangle + tf\langle \mathbf{n}, \mathbf{n}_{uu}\rangle + o(t^2) \\
&= h_{11} + t\cdot\{f_{uu} + \det(f_w\mathbf{S}_u - f_u\mathbf{S}_w, \mathbf{n}, \mathbf{S}_{uu}) + f\langle \mathbf{n}, \mathbf{n}_{uu}\rangle\} + o(t^2)\,, \\
\tilde{h}_{22} &= h_{22} + t\cdot\{f_{ww} + \det(f_w\mathbf{S}_u - f_u\mathbf{S}_w, \mathbf{n}, \mathbf{S}_{ww}) + f\langle \mathbf{n}, \mathbf{n}_{ww}\rangle\} + o(t^2)\,, \\
\tilde{h}_{12} &= h_{12} + t\cdot\{f_{uw} + \det(f_w\mathbf{S}_u - f_u\mathbf{S}_w, \mathbf{n}, \mathbf{S}_{uw}) + f\langle \mathbf{n}, \mathbf{n}_{uw}\rangle\} + o(t^2)\,, \\
\tilde{h}_{21} &= h_{21} + t\cdot\{f_{wu} + \det(f_w\mathbf{S}_u - f_u\mathbf{S}_w, \mathbf{n}, \mathbf{S}_{wu}) + f\langle \mathbf{n}, \mathbf{n}_{wu}\rangle\} + o(t^2)\,.
\end{aligned}$$

I know that $\tilde{K} = \frac{\det(\widetilde{II})}{\det(\tilde{I})}$ holds, and the determinant $\det(\tilde{I}) = g$ of the first fundamental form is given. This determinant remains identical to $\det(I)$ up to an infinitesimal change under deformation. To compute $\tilde{K}$, I have to compute the determinant of the second fundamental form, $\det(\widetilde{II}) = \tilde{h}_{11}\cdot\tilde{h}_{22} - \tilde{h}_{12}\cdot\tilde{h}_{12}$.

$$\begin{aligned}
\det(\widetilde{II}) &= \tilde{h}_{11}\cdot\tilde{h}_{22} - \tilde{h}_{12}\cdot\tilde{h}_{12} \\
&= h_{11}h_{22} - h_{12}h_{12} + o(t^2) \\
&\quad + t\cdot\{h_{11}f_{ww} + h_{11}\det(f_w\mathbf{S}_u - f_u\mathbf{S}_w, \mathbf{n}, \mathbf{S}_{ww}) + h_{11}f\langle \mathbf{n}, \mathbf{n}_{ww}\rangle\} \\
&\quad + t\cdot\{h_{22}f_{uu} + h_{22}\det(f_w\mathbf{S}_u - f_u\mathbf{S}_w, \mathbf{n}, \mathbf{S}_{uu}) + h_{22}f\langle \mathbf{n}, \mathbf{n}_{uu}\rangle\} \\
&\quad - t\cdot\{h_{12}f_{wu} + h_{12}\det(f_w\mathbf{S}_u - f_u\mathbf{S}_w, \mathbf{n}, \mathbf{n}_{wu}) + h_{12}f\langle \mathbf{n}, \mathbf{n}_{wu}\rangle\} \\
&\quad - t\cdot\{h_{12}f_{uw} + h_{12}\det(f_w\mathbf{S}_u - f_u\mathbf{S}_w, \mathbf{n}, \mathbf{n}_{uw}) + h_{12}f\langle \mathbf{n}, \mathbf{n}_{uw}\rangle\} \\
&= h_{11}h_{22} - h_{12}^2 + o(t^2) \\
&\quad + t\{h_{11}f_{ww} + h_{11}\det(f_w\mathbf{S}_u - f_u\mathbf{S}_w, \mathbf{n}, \mathbf{S}_{ww}) \\
&\qquad + h_{22}f_{uu} + h_{22}\det(f_w\mathbf{S}_u - f_u\mathbf{S}_w, \mathbf{n}, \mathbf{S}_{uu}) \\
&\qquad - 2h_{12}f_{uw} - 2h_{12}\det(f_w\mathbf{S}_u - f_u\mathbf{S}_w, \mathbf{n}, \mathbf{n}_{uw})\}\,.
\end{aligned}$$

As proven in Lemma 3.1b, $\langle \mathbf{n}, h_{11}\mathbf{n}_{ww} + h_{22}\mathbf{n}_{uu} - h_{12}\mathbf{n}_{uw} - h_{12}\mathbf{n}_{wu}\rangle = 0$ holds. Assuming $f_{uw} = f_{wu}$, one can conclude that

$$\begin{aligned}
\tilde{K} &= \frac{\tilde{h}_{11}\tilde{h}_{22} - \tilde{h}_{12}^2}{g} + o(t^2) \\
&= K + t\cdot\{h_{11}f_{ww} + h_{22}f_{uu} - 2h_{12}f_{uw} \\
&\qquad + h_{11}(f_w\Gamma_{22}^2 - f_u\Gamma_{22}^1)\det(\mathbf{S}_u, \mathbf{n}, \mathbf{S}_w) \\
&\qquad + h_{22}(f_w\Gamma_{11}^2 - f_u\Gamma_{11}^1)\det(\mathbf{S}_u, \mathbf{n}, \mathbf{S}_w) \\
&\qquad + 2h_{12}(f_w\Gamma_{12}^2 - f_u\Gamma_{12}^1)\det(\mathbf{S}_u, \mathbf{n}, \mathbf{S}_w)\}\,.
\end{aligned}$$

Since $\det(\mathbf{S}_u, \mathbf{n}, \mathbf{S}_w) = -\det(\mathbf{n}, \mathbf{S}_u, \mathbf{S}_w) = -\langle \mathbf{n}, [\mathbf{S}_u, \mathbf{S}_w]\rangle = -\sqrt{g}$, the Gauß curvature changes als follows under deformation:

$$\begin{aligned}
\tilde{K} = K + t\cdot\{&h_{11}f_{ww} + h_{22}f_{uu} - 2h_{12}f_{uw} + h_{11}(f_w\Gamma_{22}^2 - f_u\Gamma_{22}^1)\sqrt{g} \\
&+ h_{22}(f_w\Gamma_{11}^2 - f_u\Gamma_{11}^1)\sqrt{g} - 2h_{12}(f_w\Gamma_{12}^2 - f_u\Gamma_{12}^1)\sqrt{g}\}\,.
\end{aligned}$$

□

This concludes the proof of the main theorem of this chapter.

3.5 Example

In the following examples, I consider linear deformations, assuming bilinear deformation function

$$f(u,w) = au + bw + c$$

which has the derivatives $f_u = a$, $f_w = b$, and $f_{uw} = f_{wu} = 0$.

Example 3.1 (Helicoid)**.** I deform a helicoid $\mathbf{S}$ which is a minimal surface of the form

$$\mathbf{S}(u,w) = \begin{pmatrix} u\cos w \\ u\sin w \\ d\cdot w \end{pmatrix},$$

with $d = \frac{d_0}{2\pi}$, where d_0 is the number of windings.

This gives us the following derivatives and normal vector:

$$\mathbf{S}_u = \begin{pmatrix} \cos w \\ \sin w \\ 0 \end{pmatrix} \quad \mathbf{S}_w = \begin{pmatrix} -u\sin w \\ u\cos w \\ d \end{pmatrix} \quad \mathbf{S}_{uw} = \begin{pmatrix} -\sin w \\ \cos w \\ 0 \end{pmatrix},$$

$$\mathbf{n} = \frac{[\mathbf{S}_u, \mathbf{S}_w]}{||[\mathbf{S}_u, \mathbf{S}_w]||} = \frac{\left[\begin{pmatrix} \cos w \\ \sin w \\ 0 \end{pmatrix}, \begin{pmatrix} -u\sin w \\ u\cos w \\ d \end{pmatrix}\right]}{\left\|\left[\begin{pmatrix} \cos w \\ \sin w \\ 0 \end{pmatrix}, \begin{pmatrix} -u\sin w \\ u\cos w \\ d \end{pmatrix}\right]\right\|} = \frac{1}{\sqrt{u^2+d^2}} \begin{pmatrix} d\sin w \\ -d\cos w \\ u \end{pmatrix}.$$

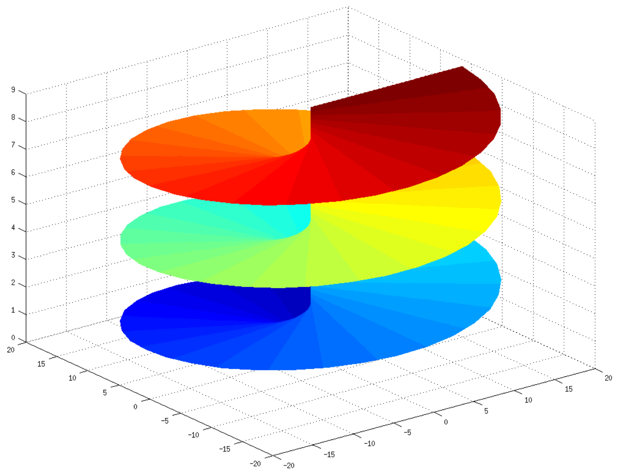


Figure 3.2: A helicoid with $u, w \in [0, 6\pi], d = \frac{3}{2\pi}$.

Next, I compute the elements g_{ij} of the first fundamental form, and the elements h_{ij} of the second fundamental form.

$$\begin{aligned}
g_{11} &= \langle \mathbf{S}_u, \mathbf{S}_u \rangle = \cos^2 w + \sin^2 w + 0 && = 1 \\
g_{12} &= \langle \mathbf{S}_u, \mathbf{S}_w \rangle = -u \sin w \cos w + u \sin w \cos w + 0 && = 0 \\
g_{22} &= \langle \mathbf{S}_w, \mathbf{S}_w \rangle = u^2 \sin^2 w + u^2 \cos^2 w + d^2 && = u^2 + d^2 \\
h_{11} &= \langle \mathbf{n}, \mathbf{S}_{uu} \rangle = \frac{d \sin w \cos w - d \cos w \sin w}{\sqrt{u^2+d^2}} && = 0 \\
h_{12} &= \langle \mathbf{n}, \mathbf{S}_{uw} \rangle = \frac{-d \sin^2 w - d \cos^2 w}{\sqrt{u^2+d^2}} && = \frac{-d}{\sqrt{u^2+d^2}} \\
h_{22} &= \langle \mathbf{n}, \mathbf{S}_{ww} \rangle = \frac{-du \cos w \sin w + du \cos w \sin w}{\sqrt{u^2+d^2}} && = 0\,.
\end{aligned}$$

Computing Df for surface **S** and a deformation function f, and setting $\mathrm{D}f = 0$ (Theorem 3.1) yields

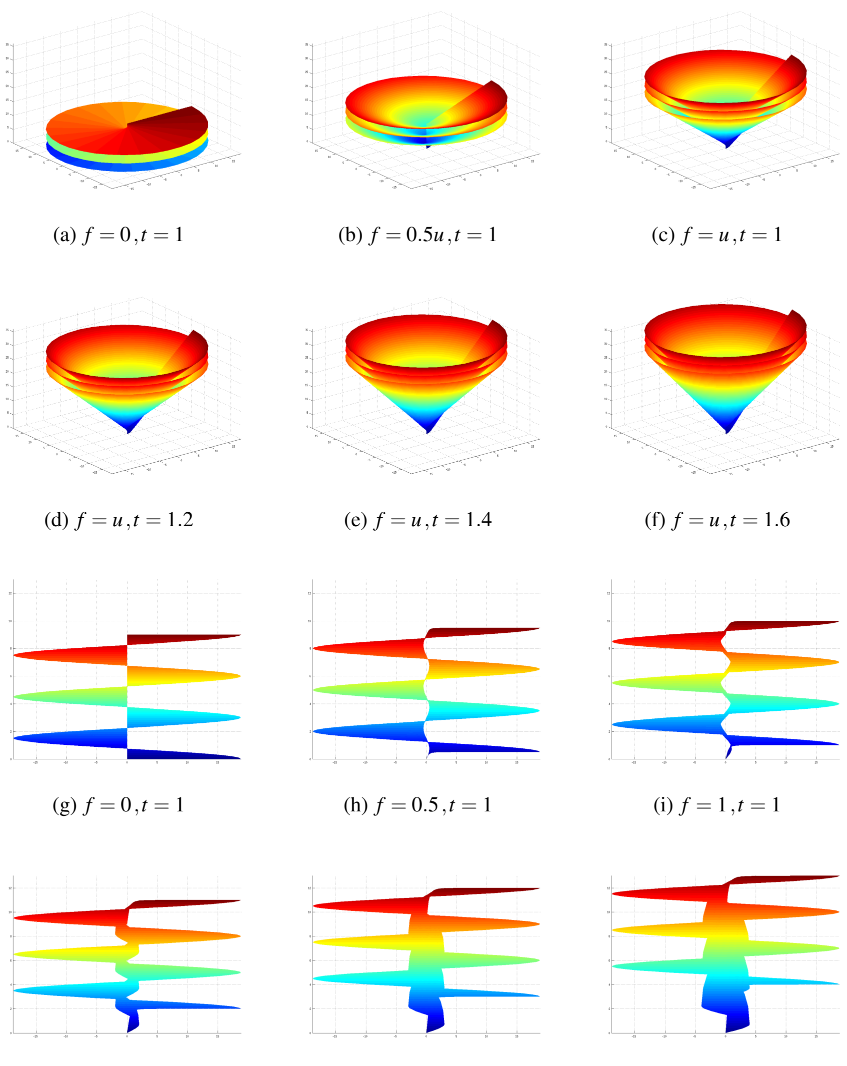
$$\begin{aligned}
\mathrm{D}f &= h_{11} f_{ww} + h_{22} f_{uu} - 2h_{12} f_{uw} - (h_{11}\Gamma^1_{22} - h_{12}\Gamma^1_{12} + h_{22}\Gamma^1_{11}) \cdot f_u \sqrt{g} \\
&\quad + (h_{11}\Gamma^2_{22} - h_{12}\Gamma^2_{12} + h_{22}\Gamma^2_{11}) \cdot f_w \sqrt{g} \\
&= 0 \cdot f_{ww} + 0 \cdot f_{uu} - 2\frac{-d}{\sqrt{u^2+d^2}} \cdot f_{uw} \\
&\quad - \left(0 \cdot \Gamma^1_{22} - \frac{-d}{\sqrt{u^2+d^2}} \Gamma^1_{12} + 0 \cdot \Gamma^1_{11} \right) \cdot f_u \sqrt{u^2+d^2} \\
&\quad + \left(0 \cdot \Gamma^2_{22} - \frac{-d}{\sqrt{u^2+d^2}} \Gamma^2_{12} + 0 \cdot \Gamma^2_{11} \right) \cdot f_w \sqrt{u^2+d^2} \\
&= \frac{2d}{\sqrt{u^2+d^2}} \cdot f_{uw} + d \cdot 0 \cdot f_u + d \cdot u \cdot f_w \\
&= \frac{2d}{\sqrt{u^2+d^2}} \cdot 0 + du \cdot f_w \\
&= du \cdot f_w \\
&= du \cdot b \overset{!}{=} 0 \\
&\Leftrightarrow b = 0\,,
\end{aligned}$$

since $\Gamma^1_{12} = \langle \mathbf{S}_{uw}, \mathbf{S}_u \rangle = 0$ and $\Gamma^2_{12} = \langle \mathbf{S}_{uw}, \mathbf{S}_w \rangle = u$.

Therefore, to make the deformation (total) curvature-preserving, the bilinear distribution function has to be simplified to the form $au + c$.

The resulting deformation is shown in Figure 3.3. The influence of the linear coefficient $a \in \{0, 0.5, 1\}$ is given in the first row: in the beginning, the surface is a helicoid, but with increasing a, it deforms to a funnel. This effect is amplified by the scaling parameter $t \in \{1.2, 1.4, 1.6\}$, since both are multipliers for the normal, as seen in the second row. The influence of the constant coefficient $c \in \{0, 0.5, 1\}$ is given in the third row: in the beginning, the helicoid's centre curve is a straight line, but with increasing c, it deforms into a helix, dragging along the adjacent portions of the surface. The last row demonstrates the effect of $t \in \{2, 3, 4\}$ on this additive portion: the almost vertical surface parts are stretched from little more than a line to long sheets hanging down.

In real-world examples, parameters have to be chosen carefully (and small) to avoid such drastic deformations. I used extremely large parameters for the example in Figure 3.3 to exaggerate the effect of each parameter on the deformation. Realistically,



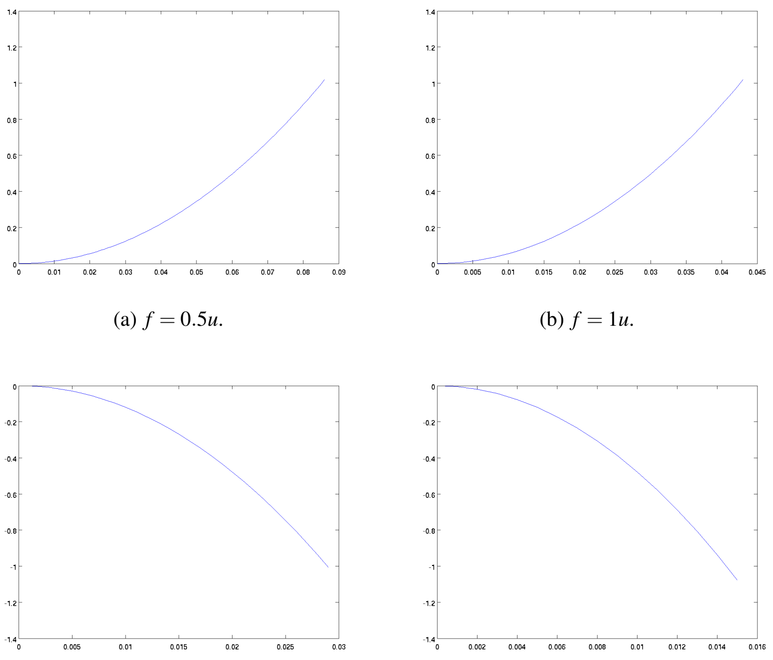
(a) $f = 0, t = 1$ (b) $f = 0.5u, t = 1$ (c) $f = u, t = 1$

(d) $f = u, t = 1.2$ (e) $f = u, t = 1.4$ (f) $f = u, t = 1.6$

(g) $f = 0, t = 1$ (h) $f = 0.5, t = 1$ (i) $f = 1, t = 1$

(j) $f = 1, t = 2$ (k) $f = 1, t = 3$ (l) $f = 1, t = 4$

Figure 3.3: The impact of deformation on the helicoid, shown separately for a, c, t. In the first and third row, only $f = au + c$ is varied. The first row shows a varying coefficient a with a fixed coefficient $c = 0$, while the third row shows a varying coefficient c with a fixed coefficient $a = 0$. For the second and fourth row, I keep f fixed, while varying the scaling parameter t in order to demonstrate the influence of scaling on the linear and constant coefficients. Figures (3.3a) and (3.3g) display the same surface from different perspectives.



(a) $f = 0.5u$.

(b) $f = 1u$.

(c) $f = 0.5$.

(d) $f = 1$.

Figure 3.4: Plot of ΔK_{tot} (vertical) over increasing t (horizontal) up to $|\Delta K_{\text{tot}}| = 1$.

one has to choose a much smaller t since I assume $o(t^2)$ to be negligible in my proof. Thus $t < 1$ arises as a necessary requirement.

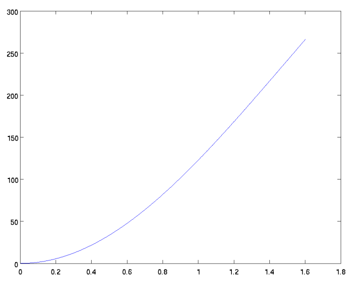
With an initial $K_{\text{tot}} = 0.0011$, I examine changes of up to a total of $\Delta K_{\text{tot}} = 1$. The discretised helicoid consists of 10201 points, so this results an average change of 0.000098 in Gauss curvature per point. This threshold is first reached for $t = 0.015$ with $f = 1$, and it is last reached for $t = 0.086$ with $f = u$ in the presented example.

In Figures 3.4 and Figures 3.5, I use the same deformation function parameters as in Figure 3.3. Both Figures illustrate how ΔK_{tot} changes with increasing t. In Figure 3.4, I show the change until the threshold of $\Delta K_{\text{tot}} = 1$ is reached. In Figure 3.5, I continue deforming until $t = 1.6$, the maximum deformation used for the upper half of Figure 3.3, to demonstrate the instabilties occuring for large t. In these cases, the prototype of a model has to be adapted before applying further infinitesimal bendings.

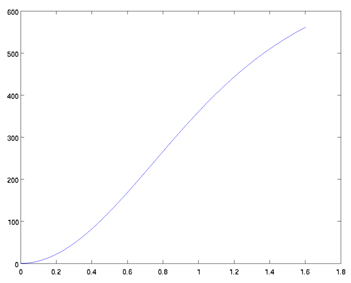
For this particular example, the signs of a and c do not affect ΔK_{tot} when varied individually since the helicoid is symmetric and applying the deformation with the opposite sign results in a similar deformation in opposite direction.

3.6 Case Study

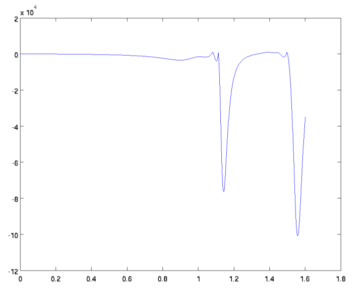
Large industrial surface models are typically composed of smaller parts. E.g. consider a turbine: it is composed of fan blades, fandisks, and many other components. It would not necessarily make sense to deform the entire model at once, but it is relatively easy to modify a single part like a fan blade or a fandisk.



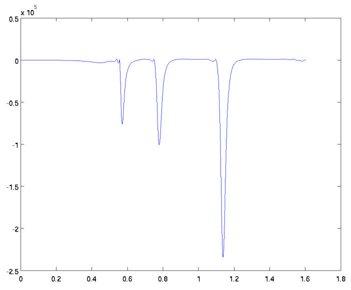
(a) $f = 0.5u$.



(b) $f = 1u$.



(c) $f = 0.5$.



(d) $f = 1$.

Figure 3.5: Plot of ΔK_{tot} (vertical) over increasing t (horizontal) up to $t = 1.6$.

Example 3.2 (Fandisk)**.** In this example, I present deformations on Hoppe's fandisk model [Hop96]. I have recreated the part marked in the rendering of the original model (Figure 3.6a) from Bézier surface patches (Figure 3.6b). As most real-world examples, this model has hard edges. I preserve them as surface patch boundaries between adjacent patches. To deform the entire model rather than a single patch at a time, I take the average of adjacent surface normals to perturb edges. Note, that this is only possible for oriented manifolds.

I now take the technique I developed for minimal surfaces and adapt it to general surfaces. My goal remains to keep the surface area as low as possible.

Now, I deform all surface patches with

$$\tilde{\mathbf{S}}(u,w,t) = \mathbf{S}(u,w) + t \cdot f(u,w) \cdot \mathbf{n}(u,w)\,,$$

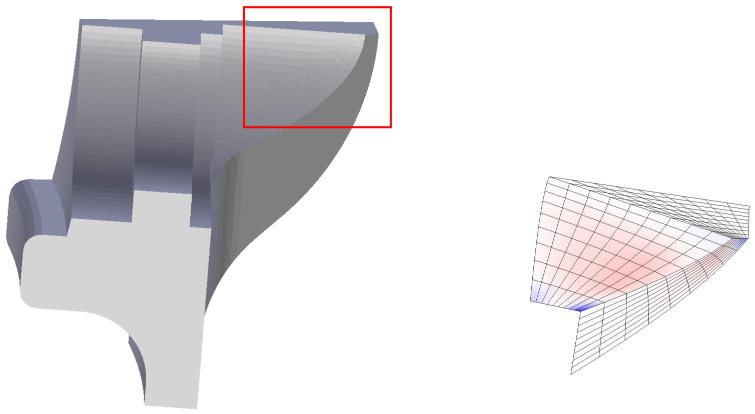
where

$$f(u,w) = au + bw + c$$

is my deformation function.

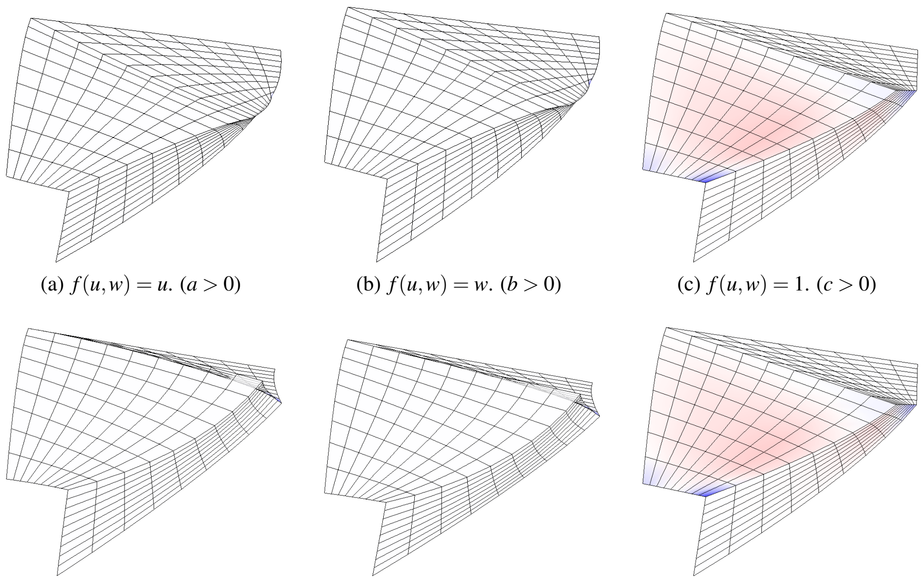
In Figure 3.7, I demonstrate the effect of isolated changes of a,b,c on the deformation. Figure 3.8 extends this by illustrating deformations for a range of different parameter combinations. Note that the colour map in Figures 3.6b, 3.7, and 3.8 depends on the Gauß curvature at each point. This means that blue areas are minima of Gauß curvature, red areas are maxima of Gauß curvature relative to the rest of the model, and white areas are close to the median Gauß curvature.

In Figure 3.9, I show changes in ΔK_{tot} over a deformation with $t \in [0,10]$. For relatively small values of t, the deformation-induced change is stable. However, as



(a) Model shown in Blender. (b) Recreated as surface patches.

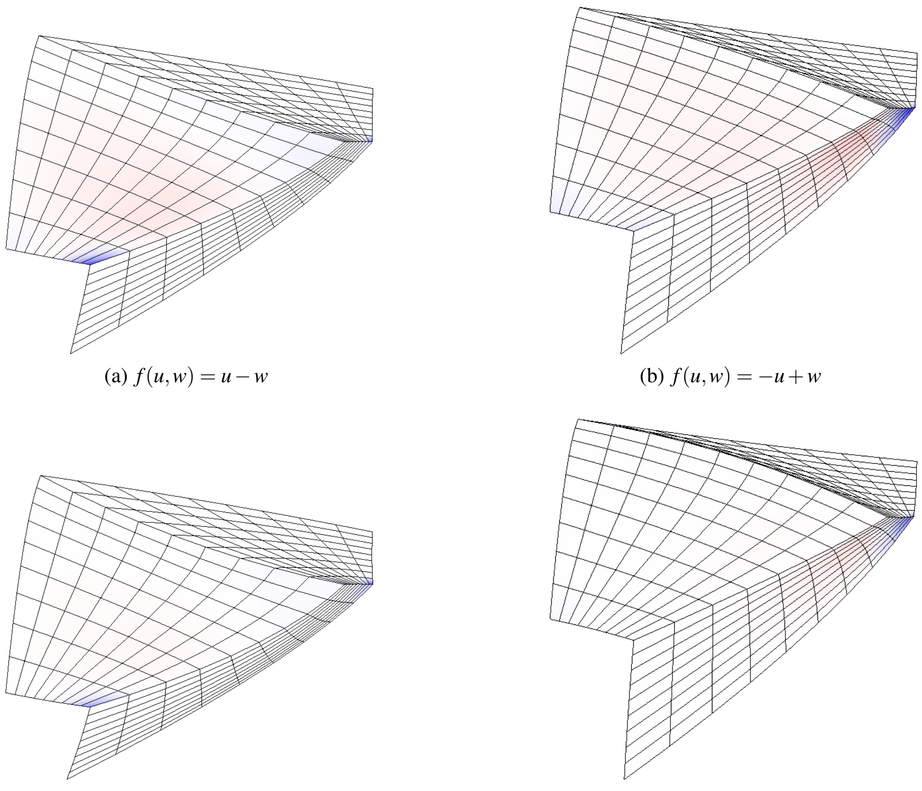
Figure 3.6: Fandisk model by Hoppe [Hop96] and a portion of it recreated from Bézier surface patches.



(a) $f(u,w) = u$. $(a > 0)$ (b) $f(u,w) = w$. $(b > 0)$ (c) $f(u,w) = 1$. $(c > 0)$

(d) $f(u,w) = -u$. $(a < 0)$ (e) $f(u,w) = -w$. $(b < 0)$ (f) $f(u,w) = -1$. $(c < 0)$

Figure 3.7: Fandisk model under deformation with $t = 0.1$.



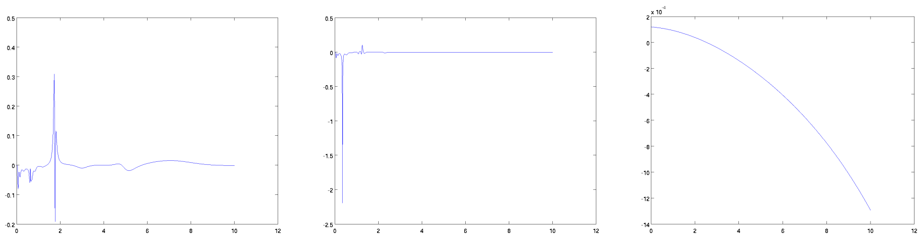
(a) $f(u,w) = u - w$

(b) $f(u,w) = -u + w$

(c) $f(u,w) = 2u - 2w$

(d) $f(u,w) = -2u + 2w$

Figure 3.8: Isolated change of one parameter at a time with $t = 0.1$.



(a) $f = u$. (b) $f = w$. (c) $f = 1$.

Figure 3.9: Change in ΔK_{tot} for $t \in [0, 10]$.

the deformation grows, instabilities begin to occur for t approximately between 0.5 and 2.0. For extremely large values of t, the deformation is stable again, however the deformed surface no longer looks similar to the original one.

3.7 Conclusion

The goal of this work is to deform the masterpiece in a meaningful way, i.e. enhance or decrease features. This can be done by perturbing in normal direction. Since there is a direct connection between total curvature and bending energy, the restriction of total curvature serves to restrict the bending energy. This maintains a surface area that is as minimal as possible and therefore reduces material cost.

I have presented a method to perturb surfaces without altering their total curvature, thereby keeping their bending energy low. This results in surfaces with a small surface area which can be manufactured at lower cost than surfaces, which have a higher total curvature or a higher bending energy. The surface and the deformation function given in Example 3.1 have simple analytic descriptions, so it is possible to make all computations manually. For the surface in Example 3.2, this is not possible since it requires the definition of normals on sharp edges.

I can deform a surface in normal direction, both outward ($f(u,w) > 0$) to increase features, and inward ($f(u,w) < 0$) to decrease features. While the examples in Figure 3.3 only present the results for deformation with a positive f, the results look very similar (but upside down) for a negative f.

In real-world examples, the surface description is a lot more complicated, making it more difficult to comprehend, what exactly happens to the surface during deformation. A lot of such complex models possess sharp edges on which tangents and normals are not clearly defined. In these cases, they have to be estimated from the neighbourhood of an edge, which can be achieved using the methods presented in the next chapter.

My method is subjected to the same numerical limitations as partial differential equations. Correctness has been proven for objects with an analytic description. However, total curvature preserving deformation can also be used for discrete surfaces. The biggest challenge with discrete surfaces is finding a suitable deformation function. It requires expert knowledge to empirically determine it, so user intervention is required. However, given such a function, the technique is applicable to discrete meshes at the sacrifice of accuracy. In my first example, I computed normals, tangents, and the deformation function analytically, but the actual deformation is performed on a discretised version of the model. Given an arbitrary mesh, the technique is limited by the availability of tangents and normals. Solutions to this are presented by

[MDSB03, TDR99, TW97], as well as in Chapter 4. If the surface has a boundary, I am, again, limited by the availability of tangents and normals. However, given this information, the deformation procedure does not discriminate between boundary points and interior points. For a single surface patch, the normal vectors and tangent vectors on the boundary can be assumed to be identical to those in their neighbourhood.

In this chapter, I have presented a global deformation method which uses curvature as a tool to restrict deformation. While normals are computed analytically for this method, I present a variety of semi-local methods to determine normals from a discrete surface structure in Chapter 4. In Chapter 5, I present a fully local technique which uses curvature as a tool to drive deformation rather than restricting it. Once the surface model of a virtual prototype has been constructed, engineers can investigate physical properties, or interaction of the environment (e.g. air) with the surface model by using simulation. Chapter 5 provides solutions to some of the challenges arising during simulation.

One challenge for possible future work is to apply my deformations in a way that preserves the index sum of all singularities of a vector field defined on this surface, and that also leaves the indices of each singularity unchanged.

Chapter 4

Discrete Normal Fields for Bézier Polygons and Bézier Meshes

4.1 Motivation

Surface prototyping is a prevalent method in engineering, which is used during the manufacturing process. As a first step, a masterpiece is created, which constitutes a draft that already provides a general idea of surface shape, but which allows for small additional changes. This first draft already contains all desired features butit still requires optimisation by an industrial surface designer to make it suitable for its task. In order to manipulate the model, e.g. by deforming the surface in normal direction, the engineer has to interact with the model. Moving each surface point individually is inefficient, and it requires tedious work. Often, models are given as collection of Bézier surface patches to facilitate interaction. Instead of manipulating the surface directly, an industrial surface designer can work with the control mesh, a much coarser representation of the surface. Complex models can be obtained by stitching together surface patches that share boundary vertices.

In the previous chapter, I presented a deformation technique that deforms the entire surface in normal direction. This requires an analytic surface description, even for discretised surfaces, since one has to define a normal vector for each point of the surface. If a normal field is provided, it is possible to minimise total Gauß curvature of a discrete surface (e.g. a surface mesh). One of the main issues with analytically given surfaces is that in most applications, engineers think in terms of shapes and surface properties (e.g. aerodynamics) rather than equations.

To an industrial surface designer, the shape and features of a surface are more relevant than its exact geometric properties. During the design process, features are enhanced or diminished. In this process, support through intuitive and efficient means of interaction is essential. This is especially true for interaction with large and complex models.

For surfaces that have a simple analytical description, such as the helicoid, the normal field along which a surface is deformed can be computed easily. If I want to apply the same kind of deformation to a Bézier curve or surface, determining the normal for every point on the surface is technically possible, however it is computationally expensive. Instead, one can exploit the properties of Bézier curves and surfaces: rather than moving each surface point in normal direction, it is possible to move the control points. If a mesh point is moved, the entire surface patch is adapted automatically, and the surface remains continuous. If normals are provided, moving control points in normal direction is a fast, easy, and efficient interaction device for industrial surface designers. However, this requires the presence of normals for the control polygon or control mesh.

In this chapter, I propose the construction of a discrete normal field on the polygon to achieve this goal. This saves a lot of computation time, but it also poses the question of how the normal of a control point should be defined.

My goal is to preserve the end points as given and only modify the inner control points of a polygon or mesh. This means that several patches can be combined to represent a surface without creating holes during deformation[1]. If I wanted to move the end points, the normal would be very easy to compute since the tangent vector or tangent plane is given immediately by the Bézier structure.

After an overview of related work, and a short introduction to the notation used in this paper,I present four different possibilities to compute normals at control points for curves and surfaces (Section 4.4), as well as two steps to considerably improve the results and to ensure a higher degree of robustness (Section 4.5). In Section 4.6, I give a detailed comparison of the different methods, their advantages and limitations. Finally, Section 4.7 concludes my work and highlights some special cases which are unlikely to occur in industrial surface design, yet may be of value to other applications.

4.2 Related Work

On an analytic surface, normal vectors can be defined analytically. For discrete surfaces this is not possible. Instead, vertex normals have to be defined based on the neighbourhood of each point. There have been various works focusing on discrete normal definitions.

Grinspun et al. [GGRZ06] use a normal map to determine the normal at each point. They approximate the mid-edge normal by fitting a quadratic curve interpolating the edges' end point.

Vertex normals are assumed to be provided for Phong Shading [BB06], which uses normals to produce a visually smooth rendering of discrete surfaces. Intermediate normals along an edge are interpolated linearly, and normalised.

Langer et al. [LBS05] approximate vertex normals by using a weighted average of the surrounding one-ring of triangles.

Mitra et al. [MNG04] present a normal estimation for point cloud data in $\mathbb{R}^2$ and $\mathbb{R}^3$, which attempts to bound error through a least square approach. They prove the existence of bounds of errors which are introduced through curvature and noise.

Max [Max99] derive weights for vertex normals computed from face normals by assuming that the surface locally approximates a sphere. These weights are therefore exact if the object is a (even irregular) tessellation of a sphere. However, it is unclear how this approximation behaves on more complex meshes, since no error bounds are defined [MDSB03].

[1]Preserving higher order continuity requires more constraints than this, but at least I keep the topological classification.

Nehab et al. [NRDR05] improve point positions and normals from 3D scans by optimising position error first and then using surface tangents to correct the normal error.

Thürmer and Wüthrich [TW97, TW98] estimate normals on the surface of a voxelised object by looking at the three-ring of neighbours and combining the values at these points with weighting factors that take distance on the (regular) grid into account. This results in a definition of the normal vector which is independent of the shape or length of the adjacent faces. Therefore, it is not suitable for irregular meshes.

Meyer et al. [MDSB03] define local vertex areas A_M which are used based on the work of Pinkall and Polthier [PP93]. Vertex normals are computed as the weighted sum of all edges .

Dyn et al. [DHKL01] present a method for 3D triangulation optimisation through discrete curvature. They employ a local cost function based on discrete total Gauß curvature and discrete total mean curvature and minimise it. An optimal triangulation is obtained through sequential edge flipping. Subsequently, they employ butterfly subdivision. The resulting meshes seem to be smoothed.

4.3 Notation

Bézier surfaces are a prevalent tool for surface modelling in the automotive industry since they are intuitive and easy to use. They build on the same concept as Bézier curves but they possess slightly different properties

Definition 4.1 (Bézier curves)**.** A *Bézier curve* $\mathbf{C}(t)$ of degree n for control points $\mathbf{p}_i$ is defined as

$$\mathbf{C}(t) = \sum_{i=0}^{n} B_i^n(t)\mathbf{p}_i\,,$$

where

$$B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i$$

are called the *Bernstein polynomials*.

Its derivative is given as

$$\mathbf{C}'(t) = n\sum_{i=0}^{n-1} B_i^{n-1}(t)(\mathbf{p}_{i+1} - \mathbf{p}_i)\,.$$

Bézier curves possess a couple of important properties [Far02b]:

Affine Invariance The curve is invariant under affine maps. This allows us to deform the control polygon instead of changing the curve directly.

Convex Hull The curve always lies inside the convex hull of all control points. This makes its shape intuitive to approximate from its polygon.

Endpoint Interpolation The curve interpolates the polygon at the end points. In design, it is usually required that the end points of a curve are in specific positions.

Pseudo-Local Control Each Bernstein polynomial $B_i^n(t)$ has exactly one maximum which lies at $t_k = \frac{i}{n}$. This means that control point $\mathbf{p}_i$ has the biggest influence near the curve point $\mathbf{C}(t_k)$. This gives a pseudo-local control over the curves [Far02b, p. 62].

Variation-Diminishing The curve mimics the shape of the polygon. If it is intersected with any line (or plane in 3D), there are at most as many curve intersections as there are polygon intersections [Far02b, p. 84]. This means that a convex polygon will always produce a convex curve, but a non-convex polygon may not necessarily produce a non-convex curve.

Definition 4.2 (Bézier surfaces)**.** A *Tensor Product Bézier patch* (in the following: *Bézier surface*) $\mathbf{S}(u,w)$ of degree $m \times n$ for control points $\mathbf{p}_{ij}$ is defined as

$$\mathbf{S}(u,w) = \sum_{i=0}^{m}\sum_{j=0}^{n} B_i^m(u)B_j^n(w)\mathbf{p}_{ij}\,.$$

Its derivatives in u– and w–direction are given as

$$\mathbf{S}_u(u,w) = n\sum_{i=0}^{m-1}\sum_{j=0}^{n} B_i^{m-1}(u)B_j^n(w)(\mathbf{p}_{i+1,j} - \mathbf{p}_{ij})\,,$$

$$\mathbf{S}_w(u,w) = m\sum_{i=0}^{m}\sum_{j=0}^{n-1} B_i^{m}(u)B_j^{n-1}(w)(\mathbf{p}_{i,j+1} - \mathbf{p}_{ij})\,.$$

Bézier surfaces possess a couple of important properties:

Affine Invariance and Convex Hull The surface is invariant under affine maps, thus allowing deformation of the control mesh instead of each surface point. In addition, the surface always lies inside the convex hull of all control points.

Boundary Curves The surface interpolates the control mesh at the end points. Since the mesh has Bézier polygons along its border, the surface borders are comprised of Bézier curves. This is very useful in design since you can achieve C^0-continuity simply by re-using border control points.

No Variation-Diminishing This property is not well-defined for Bézier surfaces. E.g. it is easy to imagine a line that intersects the surface but not its polygon.

Definition 4.3 (Hessian Normal Form)**.** A plane can be defined by its normal and its distance from the origin:

$$ax + by + cz + d = 0 \qquad \Leftrightarrow \qquad \frac{n_x x + n_y y + n_z z + d_0}{\sqrt{a^2+b^2+c^2}} = 0\,,$$

where $\mathbf{n} = (n_x, n_y, n_z)^T$. I get $\mathbf{n}\cdot\mathbf{x} = -d_0$, where $\mathbf{x}$ is a random point on the plane and d_0 can be calculated using scalar projection. This is called the *Hessian Normal Form* [GGH+89].

Then the point-plane distance for a given point $\mathbf{x}_0$ is given by

$$D = \frac{\mathbf{n}}{\|\mathbf{n}\|}\mathbf{x}_0 + d_0\,.$$

An illustration of this relationship can be found in Figure 4.1.

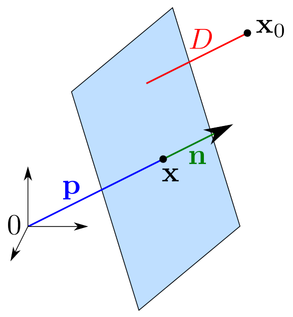


Figure 4.1: The Hessian Normal Form can be used to determine the distance D of point $\mathbf{x}_0$ to the plane defined by $\mathbf{x}$ and $\mathbf{n}$.

4.4 Normal Approximation Methods

When deforming a Bézier curve or a Bézier surface in normal direction, it is very inefficient to perform this for every single surface point since each point's normal would have to be computed, and then every single point would have to be moved. Instead, one can deform the control polygon and simply recompute the curve or surface afterwards. Hence, it is sufficient to determine normals for the control polygon. The elongated normals of control points should be orthogonal to a curve or surface, while avoiding unnecessary intersections with other parts of the control polygon or curve between the given curve and control as demonstrated in Figure 4.2.

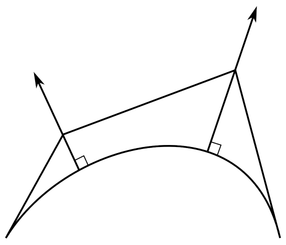


Figure 4.2: Ideal normals for a curve's control polygon.

Some of the methods I present in this section assume the presence of normal vectors for the curve or surface. For the purpose of this section, I assume that these normal vectors can be computed upon request, and that all normal vectors point towards the same side of the surface.

I compare four different normal approximation strategies. First of all, the normal can be approximated from the control structure, as presented by Grinspun et al. [GGRZ06]. While the results are not very exact, polygon-based (or mesh-based) normals are very easy to compute and provide a good first intuition of the effect a deformation will have, as summarised in Section 4.4.1. Second, I consider the curve point that has the same relative position on the curve as the control point has on the polygon. These parameter-based normals are presented in Section 4.4.2. Inspired by the concept shown in Figure 4.2, I determine direction-based normals. This is achieved by finding the curve point for which a straight line in normal direction runs through the control point, as outlined in Section 4.4.3. Finally, distance-based normals can be obtained by choosing the normal of the curve point that has the smallest distance from a control point. I present this in Section 4.4.4.

4.4.1 Control Structure Based Normal Approximation

Discrete normals can be computed for any discrete curve or surface by averaging the normals of adjacent edges or faces, as covered in literature [GGRZ06, LBS05].

Polygon-Based Normals for Curves

Assume Bézier curve is defined by control points $\mathbf{p}_i$ as seen in Figure 4.3a. Each edge $\overline{\mathbf{p}_{i-1}\mathbf{p}_i}$ has a normal $\mathbf{n}_{i-1}$ from which the control point normals are approximated.

These edge normals are computed as follows:

Let $\mathbf{c}_{i-1} = \mathbf{p}_i - \mathbf{p}_{i-1}, \mathbf{c}_i = \mathbf{p}_{i+1} - \mathbf{p}_i$ be edge vectors pointing from one control point to the next, and let $\mathbf{v} = \mathbf{p}_{i+1} - \mathbf{p}_{i-1}$ be the vector completing a triangle between these points, as illustrated in Figure 4.3. Then, I can define normal of the plane spanned by points $\mathbf{p}_{i-1}, \mathbf{p}_i, \mathbf{p}_{i+1}$:

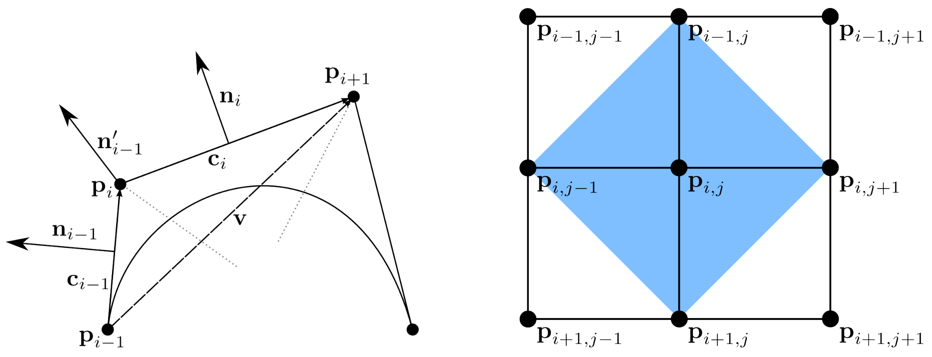
$$\mathbf{n}_{\text{span}} = \mathbf{v} \times \mathbf{c}_{i-1} = \mathbf{c}_i \times \mathbf{v}\,,$$

which can be used to define the desired normals

$$\mathbf{n}_{i-1} = \mathbf{n}_{\text{span}} \times \mathbf{c}_{i-1}\,,$$

$$\mathbf{n}_i = \mathbf{n}_{\text{span}} \times \mathbf{c}_i\,.$$

If the control polygon is not planar, the edge normals for inner edges (i.e. edges which do not contain an end point of the polygon) are not unique as both end points contribute a normal calculation. As inner edges are only used to determine control point normals, this does not impact the method at hand. However, if they are needed independently, an average of the two normals can be used.



(a) For curves, the control point's normal is computed as the weighted average of the adjacent edges' normals.

(b) For surfaces, the it is computed as the weighted average of the spanned triangles' normals.

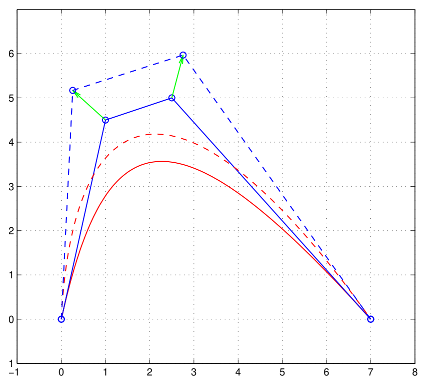
Figure 4.3: Approximation of normals from the control structure.

By taking the unweighted average of the two normals $\mathbf{n}_{i-1}, \mathbf{n}_i$, the control point's normal $\mathbf{n}'_i$ corresponds to the bisecting line of the angle between the neighbouring polygon edges. Long edges have a bigger influence on curve shape than short ones. Hence, I adapt the method and compute the normal direction $\mathbf{n}'_{i-1}$ at control point $\mathbf{p}_i$ as the weighted arithmetic average of the adjacent edges' normals

$$\mathbf{n}'_{i-1} = \frac{1}{\|\mathbf{p}_i - \mathbf{p}_{i-1}\|}\mathbf{n}_{i-1} + \frac{1}{\|\mathbf{p}_{i+1} - \mathbf{p}_i\|}\mathbf{n}_i\,.$$

For the end points, the edge normal of the edge starting from the end point is used.

A comparison of these two approaches is presented in Figure 4.4. The difference between these two approximations is noticeable, as the distance between the original curve and the deformed curve changes much more when deforming with non-weighted normals, and it stays more uniform for weighted normals.


(a) Arithmetic average.

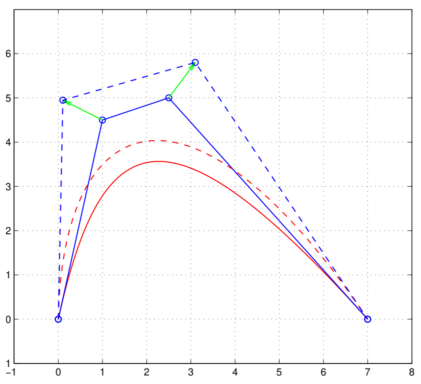

(b) Weighted average.

Figure 4.4: Comparison of unweighted and weighted polygon-based normals for control points.

Polygon-Based Normals for Surfaces

For a Bézier surface with a given control mesh as sketched in Figure 4.3b, $\mathbf{p}_{ij}$ denote the points, A_{ij} are the areas of adjacent triangles that are spanned by corresponding edges, and $\mathbf{n}_{ij}$ are the normals at control points.

The normal is now computed as the weighted average of neighbouring face normals. This neighbourhood consists of the star set. A star set is defined as the set of all triangles which are spanned by adjacent edges that fan out from a point. As weights, I use the relative area of each triangle A_{ij} in relation to the area of the entire star set $\bar{A}$.

$$\mathbf{n}_{ij} = \sum_{k=0}^{n} \frac{A_k}{\sum_{l=0}^{n} A_l} \left(\mathbf{c}_k \times \mathbf{c}_{k+1 \mod n}\right), \qquad 0 \leq k \leq n,$$

where $\mathbf{c}_k$ are the edges fanning out from $\mathbf{p}_{ij}$, n is the number of triangles spanned by the edges, and A_k are the areas of these triangles.

Along the control mesh boundary, each control point normal is computed from only two triangle normals, and for corner points, only a single triangle normal is used.

4.4.2 Parameter-Based Normal Approximation

In order to approximate a normal based on parameters, I compute the relative position of a control point in the polygon. Then, I obtain the normal of the curve or surface point with corresponding relative position on the curve or surface.

Parameter-Based Normals for Curves

A simple solution would be to set the relative position parameter t_r to $t_r = \frac{i}{n+1}$, mimicking the pseudolocal control property provided by Bézier curves. However, this only produces sensible results if all control polygon edges have similar lengths. Therefore, the edge length should be considered for the relative parameter. A comparison of parameters with and without this consideration is given in Figure 4.5. If the edges of a control polygon have different lengths, this should be considered during the computation. Therefore, I compute the relative position t_r of a control point $\mathbf{p}_r$ as the sum of edge lengths from the first control point to the current control point divided by the edge length of the entire polygon:

$$t_r = \frac{\sum_{i=0}^{r-1} \|\mathbf{p}_{i+1} - \mathbf{p}_i\|}{\sum_{i=0}^{n-1} \|\mathbf{p}_{i+1} - \mathbf{p}_i\|}.$$

This $t_r \in [0,1]$ can be used as a parameter for $\mathbf{C}$ to get the point $\mathbf{C}(t_r)$. Then, the normal of curve point $\mathbf{p}_r$ is the normal of $\mathbf{C}(t_r)$, i.e. $\mathbf{n}(t_r)$. If a control polygon's edges have uniform length, this corresponds exactly to $t_r = \frac{i}{n+1}$.

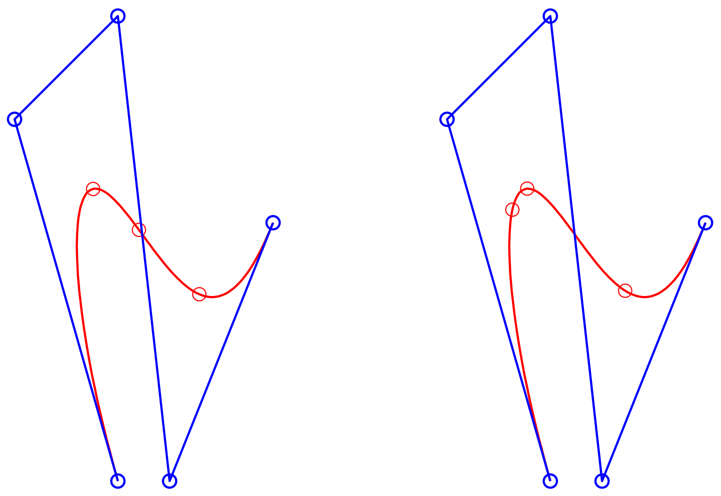


Figure 4.5: The relative parameters t_r, independent of edge length (left) and considering edge length (right). The very short second edge and the very long third edge lead to a distortion of the relative parameter.

Parameter-Based Normals for Surfaces

Similarly, the relative positions u_{rs}, w_{rs} with respect to a row or a column of control points $\mathbf{p}_{rs}$ of the mesh are computed as

$$u_{rs} = \frac{\sum_{i=0}^{r-1} \|\mathbf{p}_{i+1,s} - \mathbf{p}_{i,s}\|}{\sum_{i=0}^{n-1} \|\mathbf{p}_{i+1,s} - \mathbf{p}_{i,s}\|},$$

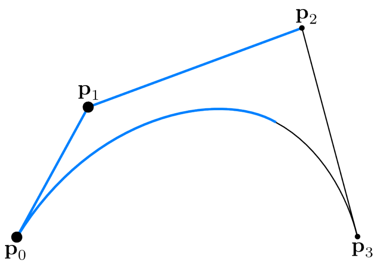
$$w_{rs} = \frac{\sum_{j=0}^{s-1} \|\mathbf{p}_{r,j+1} - \mathbf{p}_{r,j}\|}{\sum_{j=0}^{m-1} \|\mathbf{p}_{r,j+1} - \mathbf{p}_{r,j}\|}.$$

Using these parameters, one can obtain the surface point $\mathbf{S}(u_{rs}, w_{rs})$ which provides its normal $\mathbf{n}(u_{rs}, w_{rs})$ for the control point $\mathbf{p}_{rs}$.

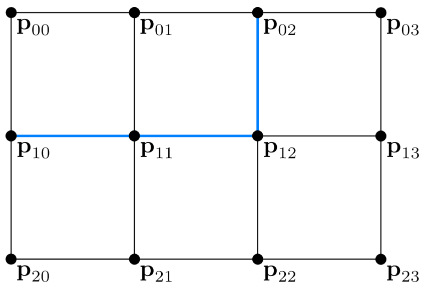
First, the relative parameters u_{0j}, u_{mj} along the boundary columns, and w_{i0}, w_{in} along the boundary rows have to be determined. Then, all u_{ij} can be computed by marching along the polygon column from w_{i0} to w_{in}. Similarly, all w_{ij} can be computed by marching along the polygon row from u_{0j} to u_{mj}.

For example, u_{rs}, w_{rs} can be determined by computing $u_{0s}, u_{ms}, w_{r0}, w_{rn}$ and then finding the appropriate point between each pair.

Figure 4.6 sketches how the relative parameter is computed.



(a) Parameter-based normals for curves ($r = 2$).



(b) Parameter-based normals for surfaces ($r = 1, s = 2$).

Figure 4.6: Approximation of normals from parameter values.

4.4.3 Direction-Based Normal Approximation

My goal is to determine a normal for each control point such that a straight line in normal direction through the control point and the curve itself are orthogonal at the intersection point, as demonstrated in Figure 4.2. It is possible to determine this normal exactly by finding the curve parameters for which this is the case.

Direction-Based Normals for Curves

I want to find a normal $\mathbf{n}(t_k)$ such that the following holds for a parameter t_k and a sufficiently large constant $s > 0$.

$$\mathbf{C}(t_k) + s \cdot \mathbf{n}(t_k) = \mathbf{p}_i \qquad \text{for } i \in \{1, \ldots, n-1\}.$$

For each parameter t, the tangent $\mathbf{C}'(t)$ is orthogonal to the normal plane of the curve. Using the Hessian Normal Form, the normal plane for a point on the curve can be defined using the corresponding tangent vector. This is the plane to which the tangent vector is orthogonal:

$$\frac{\mathbf{C}'(t_k)}{\|\mathbf{C}'(t_k)\|}\mathbf{x} = -d_0\,.$$

The distance d_0 of this plane from the origin is the scalar projection of $\mathbf{C}(t_k)$ onto the plane's normal (the curve's tangent vector $\mathbf{C}'(t_k)$):

$$d_0 = \frac{\mathbf{C}(t_k) \cdot \mathbf{C}'(t_k)}{\|\mathbf{C}'(t_k)\|}\,.$$

Therefore, I only need to test if a control point lies in the normal plane. This is the case when its distance to the plane equals zero:

$$D = \frac{\mathbf{C}'(t_k)}{\|\mathbf{C}'(t_k)\|}\mathbf{p}_i + d_0 \overset{!}{=} 0\,.$$

Figure 4.7 sketches this construction: if $\mathbf{p}_i$ lies inside the plane, then $D = 0$. In case of discrete curves, $D = 0$ may never occur. Then, the parameter t_k leading to the minimal D is chosen:

$$\min_{t_k}|D| = \min_{t_k}\left|\frac{\mathbf{C}'(t_k)}{\|\mathbf{C}'(t_k)\|}\mathbf{p}_i + d_0\right|\,.$$

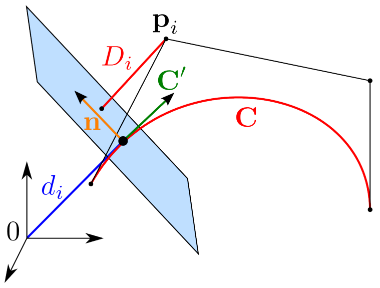


Figure 4.7: Determining the distance D_i between a normal plane and a control point $\mathbf{p}_i$ using the Hessian Normal Form.

Direction-Based Normals for Surfaces

I want to find a normal $\mathbf{n}(u_k, w_l)$ such that the following holds for parameters u_k, w_l and a sufficiently large constant $s > 0$.

$$\mathbf{S}(u_k, w_l) + s \cdot \mathbf{n}(u_k, w_l) = \mathbf{p}_{ij} \qquad \text{for } i \in \{1, \dots, n-1\}, j \in \{1, \dots, m-1\}\,.$$

For each parameter set u, w, the tangents $\mathbf{S}_u(u,w), \mathbf{S}_w(u,w)$ are orthogonal to the normal plane of the surface. Using the Hessian Normal Form, the normal planes for a point on the surface can be defined using the corresponding tangent vectors.

$$\frac{\mathbf{S}_u(u,w)}{\|\mathbf{S}_u(u_k,w_l)\|}\mathbf{x} = -d_{u0}\,, \qquad \frac{\mathbf{S}_w(u,w)}{\|\mathbf{S}_w(u_k,w_l)\|}\mathbf{x} = -d_{w0}\,.$$

The distances d_{u0}, d_{w0} of these planes from the origin are the scalar projections of $\mathbf{S}(u_k, w_l)$ onto the planes' normals:

$$d_{u0} = \frac{\mathbf{S}(u_k,w_l)\cdot\mathbf{S}_u(u_k,w_l)}{\|\mathbf{S}_u(u_k,w_l)\|} \qquad d_{w0} = \frac{\mathbf{S}(u_k,w_l)\cdot\mathbf{S}_w(u_k,w_l)}{\|\mathbf{S}_w(u_k,w_l)\|}\,.$$

Therefore, I only need to test if a control point lies in the normal plane. This is the case when its distance to a plane equals zero:

$$D_u = \frac{\mathbf{S}_u(u_k,w_l)}{\|\mathbf{S}_u(u_k,w_l)\|}\mathbf{p}_{ij} + d_{u0} \overset{!}{=} 0\,,$$

$$D_w = \frac{\mathbf{S}_w(u_k,w_l)}{\|\mathbf{S}_w(u_k,w_l)\|}\mathbf{p}_{ij} + d_{w0} \overset{!}{=} 0\,.$$

In case of discrete surfaces, $D_u \neq 0$ and $D_w \neq 0$ can occur. Then, we use parameters u_k, w_l which lead to minimal D_u, D_w:

$$\min_{u_k} |D_u| = \min_{u_k} \left| \frac{\mathbf{S}_u(u_k, w_l)}{\|\mathbf{S}_u(u_k, w_l)\|} \mathbf{p}_{ij} + d_{u0} \right| ,$$

$$\min_{w_l} |D_w| = \min_{w_l} \left| \frac{\mathbf{S}_w(u_k, w_l)}{\|\mathbf{S}_w(u_k, w_l)\|} \mathbf{p}_{ij} + d_{w0} \right| .$$

The normal $\mathbf{n}$ is then given by $\mathbf{n}(u_k, w_l)$.

4.4.4 Distance-Based Normal Approximation

The normal of a curve or surface point closest to a control point often runs through the control point. Therefore, I can compute this normal in an alternative way. For direction-based normals, I searched for the curve or surface point which has the smallest distance to the control point. Now, I minimise the distance between a control point and the curve or surface points directly.

Distance-Based Normals for Curves

The closest curve point can be found by computing the parameter t_k with minimal distance

$$\min_{t_k} \|\mathbf{p}_i - \mathbf{C}(t_k)\| .$$

The normal at point $\mathbf{C}(t_k)$ is $\mathbf{n}(t_k)$, so the control point $\mathbf{p}_i$ is assigned $\mathbf{n}(t_k)$ as its normal.

Since I have to go over the whole parameter vector t for $n-2$ of the n control points, I end up with a complexity of $O(n \cdot t)$.

Distance-Based Normals for Surfaces

The corresponding surface point can be found very easily by computing the pair of parameters (u_k, w_l) with minimal distance

$$\min_{(u_k, w_l)} \|\mathbf{p}_{ij} - \mathbf{S}(u_k, w_l)\| .$$

The normal at point $\mathbf{C}(u_k, w_l)$ is $\mathbf{n}(u_k, w_l)$, so the control point $\mathbf{p}_{ij}$ is assigned $\mathbf{n}(u_k, w_l)$ as its normal.

Since I have to go over the whole parameter vectors u, w for $(m-2) \times (n-2)$ of the $m \times n$ control points, I end up with a complexity of $O(mn \cdot uw)$.

4.5 Adjustments

In Figure 4.8, the results of all previously introduced strategies are depicted for curves. As one can see, deforming the control polygon in normal direction yields results which no longer resemble the original curve. There are two main improvements to be made

Normal Orientation I adapt normal orientation to improve feature preservation. If all normals point towards the same side of a non-convex object, deforming in this direction eventually results in a convex shape, and therefore in a loss of features. Instead, normals should always point from the curve or surface towards the control structure. E.g. curve normals should point left during a right turn, and right during a left turn.

Local Influence I restrict the search for a normal to the neighbourhood around the control point. This increases efficiency as fewer points have to be considered, and the resulting normals lead to a more faithful representation of the curve or surface.

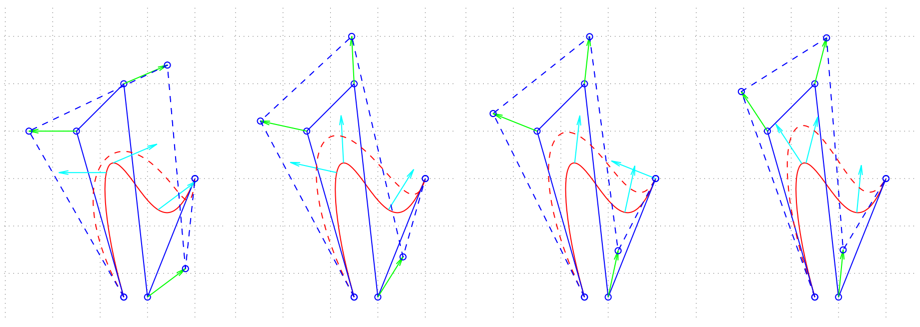

(a) Polygon. (b) Parameter. (c) Direction. (d) Distance.

Figure 4.8: Comparison of the different normal computations prior to adjustments. Figures (a) through (d) display the normals at the control point and at the curve point they belong to.

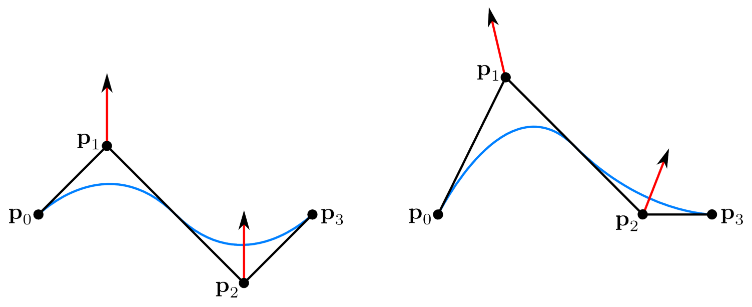
4.5.1 Normal Orientation

I want normal vectors which provide the basis for feature-preserving deformation if the control points are moved along them. If all normals point to the same side of the curve, this is only possible for convex curves. A curve with an inflection point, i.e. a curve that has left turns and right turns, will suffer a loss of features if such normals are used.

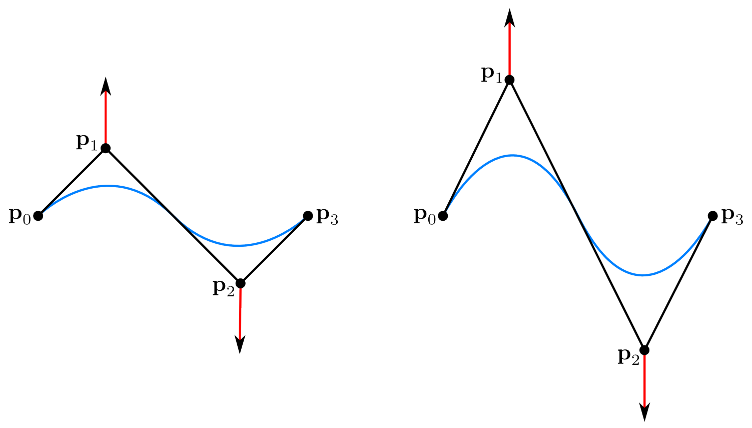
As Figure 4.9 demonstrates, there is a large difference between deformation resulting from the original, uniformly oriented normals, and the deformation resulting from flipped normals that change direction if the curve is non-convex, i.e. it has at least one inflection point. For the original normals, the right turn is enhanced whereas the left turn vanishes. If the two points were moved by another normal length, the previously non-convex polygon would become convex. This can be avoided if normals are flipped depending on the turning direction. For curves in a plane, one can compute the turning direction, however, in three-dimensional space, turning directions are not properly defined. However, at the inflection point, the relative position of the curve with respect to the polygon changes (e.g. left/right).

This relationship can be exploited to flip normals easily by applying the signs of the distance vector from curve point to control point to the normal:

$$\mathbf{n}(\mathbf{p}_i) = \operatorname{sign}(\mathbf{p}_i - \mathbf{C}(t_k)) \cdot (|n_x|, |n_y|, |n_z|)^T ,$$



(a) Original, uniformly oriented normals.



(b) Flipped normals.

Figure 4.9: Control points $\mathbf{p}_1$ and $\mathbf{p}_2$ are moved in normal direction: changing the normal orientation when the polygon changes its turning direction preserves features.

where t_k is the curve parameter for the respective normal computation strategy. This assumes that normals point from the curve towards the control point, as described in the motivation of Section 4.4. Normals approximated from the control polygon do not have any corresponding curve points. In this case, I use the same point that is used for parameter-based normals (cf. Section 4.4.2) to compute the sign.

For surfaces, normals are flipped in a similar manner:

$$\mathbf{n}(\mathbf{p}_{ij}) = \operatorname{sign}(\mathbf{p}_{ij} - \mathbf{S}(u_k, w_l)) \cdot (|n_x|, |n_y|, |n_z|)^T ,$$

where u_k, w_l are the surface parameters for the respective normal computation strategy. Normals approximated from the control mesh do not have any corresponding surface points, so we use the same point that is used for parameter-based normals to compute the sign.

At curve end points and surface corners, the sign is not defined as the distance between control structure and curve/surface is zero. As I only move inner points, curve end points and the entire surface border are left facing in the original direction.

Note, that the possibility to adapt normal orientation for feature preservation is a major strength of control structure-based curve and surface definitions. At inflection points, the sign of a normal switches, so the sign function has a singularity, i.e. it is not defined for these curve points. Similarly, the sign changes at saddle points, so again, the sign function has a singularity, i.e. it is not defined for these surface points.

So far, I have assumed that normals are provided along with the curve or surface. In the following section, I present a technique to define normal vectors for curves if the tangent vector $\mathbf{C}'(t_k)$ is known.

Computing a Normal Vector Given a Tangent Vector

All normal approximation methods I present are based on a tangent vector. A 3D vector and its starting point define a plane according to the Hessian Normal Form. The normal vector I am looking for has to lie in this plane. However, its direction on the plane can be chosen.

I want the normal vector to point towards the control polygon as I want the normal direction to be a good representative for both control point and curve. Therefore, I want the normal vector to lie in the plane spanned by the tangent vector and the vector pointing from the curve point to the control point. Given the restriction to both this spanned plane and the normal plane, the normal vector can be uniquely determined.

Let $\mathbf{t} = \mathbf{C}'(t_k)$ be the tangent vector at parameter t_k, and $\mathbf{v} = \mathbf{p}_i - \mathbf{C}(t_k)$ the vector pointing from $\mathbf{C}(t_k)$ to $\mathbf{p}_i$. Then I can define the spanned plane's normal

$$\mathbf{n}_{\text{span}} = \mathbf{t} \times \mathbf{v}$$

which can be used to define the desired normal

$$\mathbf{n} = \mathbf{n}_{\text{span}} \times \mathbf{t}\,.$$

This is illustrated in Figure 4.10.

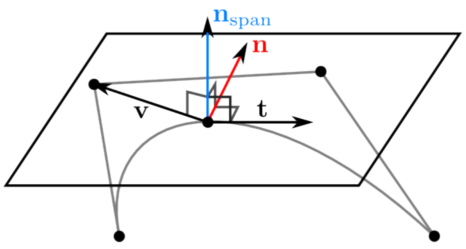


Figure 4.10: Computing a normal vector, given a tangent vector.

4.5.2 Local Influence

The methods I have presented so far use the entire curve or surface as a basis, so they have global influence of the curve on the normals. One of the main drawbacks of direction-based normals and distance-based normals as defined in Section 4.4 is the fact that they are not always uniquely defined. When there are several optimal solutions, it is not clear which solution is the most desirable. Matlab, for instance, automatically chooses the first optimum if there are several identical values. This leads to a bias towards the values that appear earlier in arrays.

As seen in Figure 4.8c, a non-convex curve can have several points with normals which run exactly through a control point. The last (right-most) curve point's normal runs exactly through the second control point, thus it was chosen as a normal although it is not a desirable result. In this particular case, the issue lies in discretisation. The resulting normal was a slightly better match (i.e. passed closer to the control point) than the expected curve point's normal.

A non-convex curve (or a self-containing curve, e.g. a spiral) can have curve points that are far away from a control point in parameter space, but very close to it in Euclidean space. As seen in Figure 4.8d, the first (bottom left) curve point is much closer to the fourth control point than any other curve point, including all points which would yield a suitable result.

I want to restrict the search space for a suitable normal vector to the neighbourhood of each control point. The question is: how do you define a neighbourhood?

Bézier curves are constructed using Bernstein polynomials as basis functions. Figure 4.11 shows the Bernstein polynomials $B_i^n(t)$ of degree $n = 5$. Each basis function has its maximum at $t = \frac{i}{n}$ [Far02b].

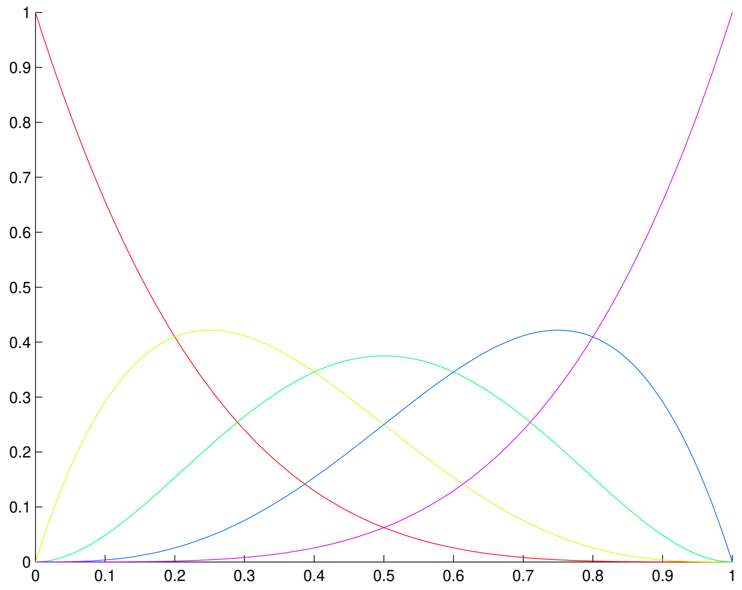


Figure 4.11: Bernstein polynomials.

I propose the introduction of local influence, similar to the concept of local control as provided by B-Splines. Instead of using a knot vector, each control point $\mathbf{p}_k$ can be assigned a normal $\mathbf{n}(t_k)$ for which t_k lies in the range of values between the maxima of Bernstein polynomials of the two adjacent control points:

$$\frac{k-1}{n} \leq t_k \leq \frac{k+1}{n}.$$

These regions of interest are illustrated in Figure 4.12.

To determine the normal $\mathbf{n}(t_k)$, only points $\mathbf{C}(t) \in [\mathbf{C}(\frac{k-1}{n}), \mathbf{C}(\frac{k+1}{n})]$ are considered.

4.6 Comparison

In the previous section, I introduced four different strategies to compute normal vectors for control structure-based curves and surfaces. In this section, I compare how well they fulfil the requirement of feature preservation given in Figure 4.2, and examine the differences in computational complexity.

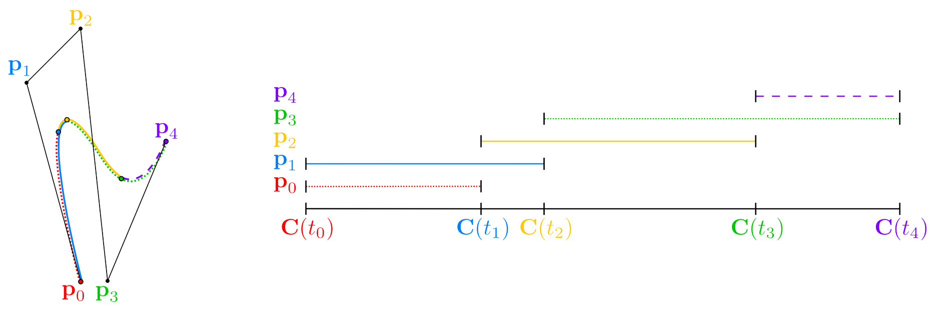


Figure 4.12: Each point only considers a part of the curve for its normal.

Figure 4.13 shows a comparison of all different normal computation strategies at the example of three different Bézier curves which present a variety of challenges. These examples are the basis on which I evaluate each method.

The top row presents the results for the previously used wave-shaped curve. This example exhibits typical properties of industrial surfaces: it bends in different directions but it does not self-intersect and it is not self-containing.

In addition, I examine some special cases which, although they are not desirable in industrial modelling, may be of interest to other applications. The middle row presents the results for a loop-shaped curve: this is a self-intersecting convex curve with a self-intersecting polygon. The bottom row presents the results for a non-intersecting, self-containing spiral control polygon with a self-intersecting curve.

Figures 4.15 and 4.16 show a comparison of each of the four stategies for surfaces. In both figures, I compare the results for global influence and local influence side by side. While Figure 4.15 shows a convex surface for which uniform normal orientation yields correct results, Figure 4.16 shows a non-convex surface, and it highlights the effect of adjusting normal orientation.

4.6.1 Feature Preservation

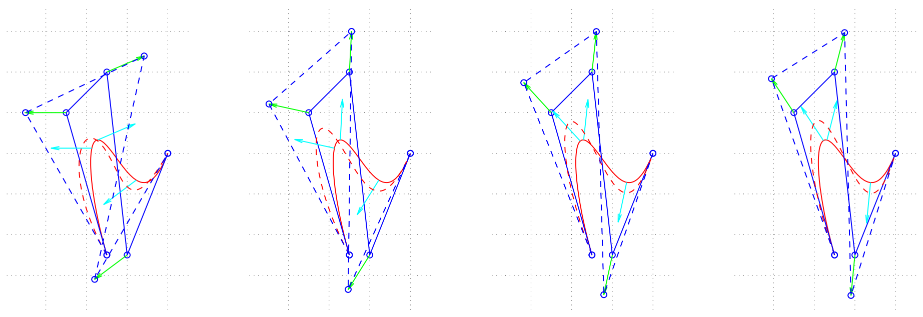
Let me first look into the feature-preserving qualities of all methods. I start with curves, and I move on to surfaces.

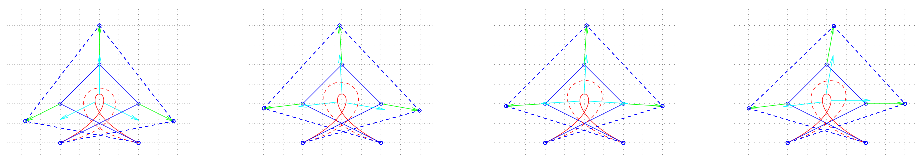
For the wave polygon (Figure 4.13a–4.13d), the most significant difference between the four strategies is that polygon-based normals and parameter-based normals introduce self-intersections in the polygon, whereas direction-based normals and distance-based normals preserve polygon topology. A comparison between direction-based normals and distance-based normals shows that the inflection point before and after deformation stays the same for distance-based normals but not for direction-based normals. Therefore, distance-based normals produce the most faithful representation for this example.

For the loop polygon (Figure 4.13e–4.13h), the results of 4.13f–4.13h look almost identical. In all three cases, the loop grows uniformly in all directions such that the original loop is centred in the deformed loop curve. One aspect that is noteworthy is the slight variation in the top normal of Figure 4.13h, which does not point up vertically. This is a discretisation issue: at a low resolution (100 points in this example), it is less likely that the optimal point is contained in the dicretised curve than for a higher-resolution curve. The same effect is present for parameter-based normals and direction-based normals, but it has less impact. Polygon-based normals result in a slightly less convincing representation than the other three strategies as the centre of the loop curve changes noticably through deformation, as seen in Figure 4.13e.

For the spiral polygon (Figure 4.13i–4.13l), the first thing of note is that for three methods (polygon-based normals, direction-based normals, and distance-based normals), the edges of the deformed polygon are almost parallel to those of the original polygon. However, for parameter-based normals, the results differ significantly. This leads to a shift in clockwise direction for the polygon, and a shift to the right for the curve, both of which are not present for the other methods. Polygon-based normals have a similar issue but to a lesser extent. Here, the different edge lengths at two control points lead to slightly skewed normals. Direction-based normals and distance-based normals yield very similar results which are faithful representations of the original polygon and curve.

Overall, there are two equivalence classes: as a whole, direction-based normals and distance-based normals provide better results than polygon-based normals and parameter-based normals. Within each equivalence class, performance depends on the specific example.


(a) Polygon. (b) Parameter. (c) Direction. (d) Distance.


(e) Polygon. (f) Parameter. (g) Direction. (h) Distance.

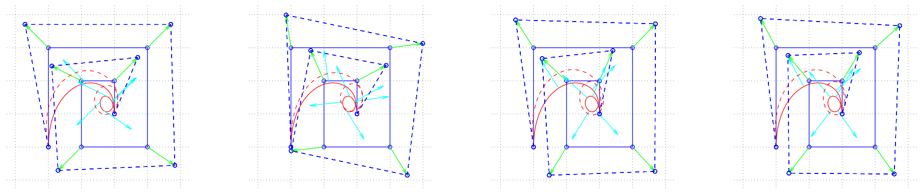

(i) Polygon. (j) Parameter. (k) Direction. (l) Distance.

Figure 4.13: Comparison of the different normal computations using local influence with changed normal orientations for three different polygons.

For surfaces, one can see particularly well how much difference it makes to reduce search space to the neighbourhood of a control point . In Figures 4.15 and 4.16, each column contains all four normal approximation strategies. In each row, I compare the normals based on global influence to those based on local influence.

One noteable difference between strategies is that direction-based normals and distance-based normals perform much better using local influence than using global influence. In comparison, mesh-based normals and parameter-based normals stay the same. The main reason for this lies in their local nature: mesh-based normals do not depend on the surface at all, and parameter-based normals naturally end up with a normal from the neighbourhood unless the control mesh has very different edge lengths.

One major drawback of mesh-based normals is the difficulty to adjust normal orientation as this strategy relies solely on the control structure. A possible solution would be to use surface points determined using the strategy described in Section 4.4.2 to determine orientation. Alternatively, one could determine local minima, local maxima, and saddles. This can be achieved by using discrete curvature definitions to determine these structures.

This drawback of parameter-based normals is demonstrated in Figure 4.14. In this example, the surface is planar in order to emphasise the connection between parametrisation and control mesh. The control polygon is plotted in colours, whereas the surface and the distribution of its parameters is plotted in black (for $u, w \in \left\{0, \frac{1}{3}, \frac{2}{3}, 1\right\}$). As one can see, the parameter values corresponding to the control points do not line up: the control point at $(2,0)$ lies at $u = \frac{1}{3}$ of the length of the mesh border. The value on the surface for $u = \frac{1}{3}$ lies at about $(1,0)$, however the value of the surface at about $u = 0.5$ would be a more appropriate choice.

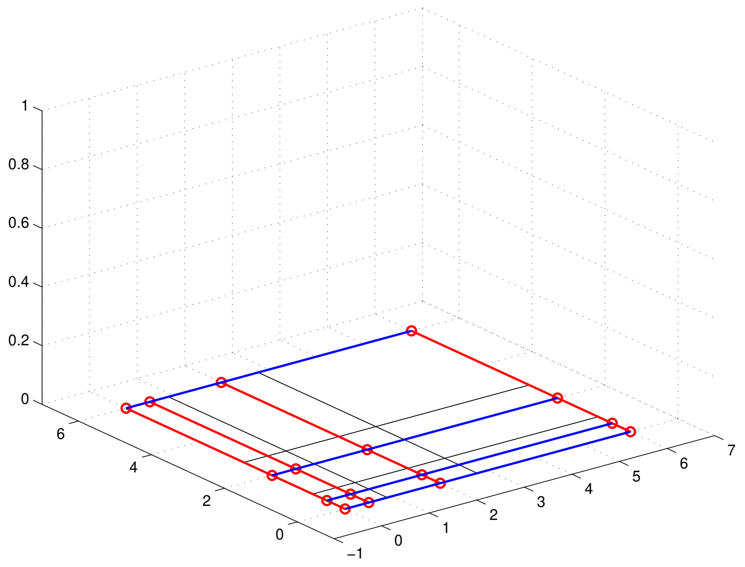


Figure 4.14: Distortion in parameter distribution resulting from large differences in edge lengths.

As pointed out before, when there are several optimal solutions, it is not clear how to choose the correct one. Both direction-based normals and distance-based normals exhibit a noticeable shift of normal vectors in one direction as the first optimal match is chosen. This can be seen in Figures 4.15e, 4.15g, 4.16e, and 4.16g. These two strategies are the strategies which benefit most in runtime from a reduction of search space.

Overall, direction-based normals and distance-based normals always provide good results if combined with the suggested adjustments, whereas control structure-based normals and parameter-based normals depend heavily on the specific example at hand.

4.6.2 Computational Complexity

For the complexities given here, I assume that n is the number of control points, and $\bar{t}$ is the number of curve points.

Control structure based normals are one of the fastest methods at an efficiency in $O(n)$. Each edge normal has to be computed [$n-1$ edges, since computing a normal is assumed to be constant]. Then, each pair of neighbouring edges is averaged using the edge lengths as a weight [$n-2$ pairs].

Parameter-based normals also have an efficiency in $O(n)$. Each edge's length has to be computed [$n-1$]. There are $r \leq n$ edges which are visited twice [n].

Both direction-based normals and distance-based normals require the search for a minimum over the whole parameter space. Depending on the desired normal accuracy, the number of parameter values $\bar{t}$ can be chosen.

Direction-based normals further require calculating the distances D_i between normal planes and each control point [n] for each parameter [t], so the efficiency lies in $O(n^2\bar{t})$. Computing D has quadratic complexity since two polynomials are multiplied.

Similarly, distance-based normals require computing the distances between $n-2$ control points and each parameter [$\bar{t}$]. Their complexity therefore lies in $O(n\bar{t})$.

As $n \ll \bar{t}$, both methods have complexities in $\mathcal{O}(\bar{t})$.

Overall, the complexity rises with the expected result quality.

4.7 Conclusion

The main objective of this chapter was to develop a method to define normals on a discrete mesh, wish special focus on Bézier surface patches.

I have presented four different methods to compute normals and evaluated their impact during deformation. Polygon-based normals can be used for arbitrary meshes, whereas the remaining three techniques are more specifically aimed at Bézier surface patches. However, the same concepts can be applied to other techniques which define curves or surfaces through control structures. Some examples include Bézier triangle patches, or Coons patches.

One major problem of direct manipulation of a curve (or surface) is, that it only provides continuous results for convex curves (or surfaces). A non-convex curve's normals should flip at inflection points to preserve characteristic features. Similarly, a non convex surface's normals should flip at saddle points to preserve characteristic features.

Control structure-based normals provide a very fast normal approximation. They work well on convex structures but have difficulties with non-convex structures. The major advantage of this technique is its independence of curve or surface resolution. Overall, the results are not as good as those produced by other strategies.

Parameter-based normals provide a fast normal approximation. They work well on control structures with edges of uniform length, however, large differences in edge length can result in significant errors. This can be seen in Figures 4.13j and 4.14. Resolution plays a noticeable role in quality, as a curve or surface point of similar parameter value as the control point are required. For applications which value speed over accuracy, this method may be a profitable choice.

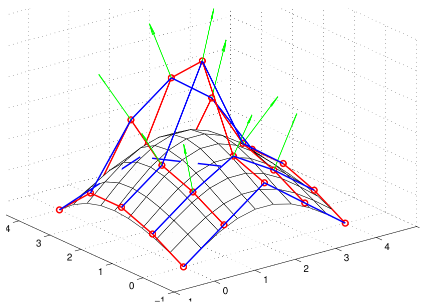
Direction-based normals provide accurate normal approximations if combined with local influence. Accuracy depends on curve or surface resolution as a higher number of points increases the likelihood of an exactly fitting normal. They are slightly slower than the first two strategies but still perform very well.

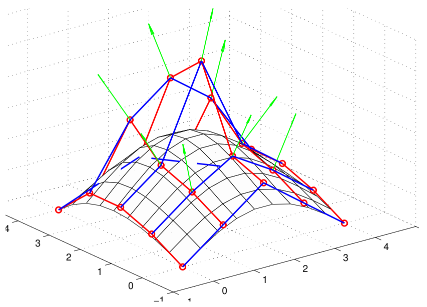
Distance-based normals provide a very accurate approximation if combined with local influence. Like direction-based normals, they are slightly slower than the first two strategies as the efficiency depends on resolution. However, they balance the additional effort with very good results. In most examples, distance-based normals performed slightly better than direction-based normals.

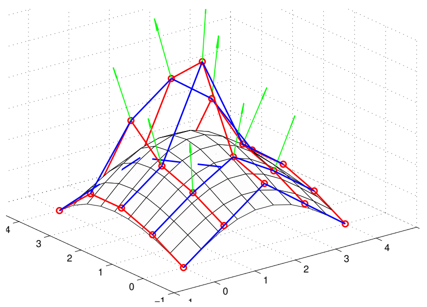
In addition to these normal definitions, I have introduced two additional techniques which can be combined with a variety of methods.

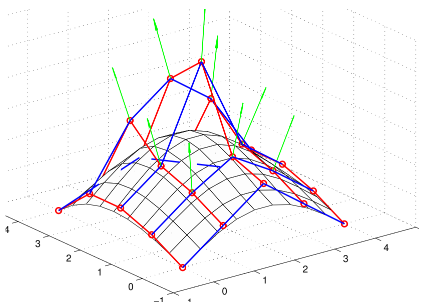
First of all, I have adjusted normal orientation to preserve surface features throughout the deformation process. This is not possible for analytically defined curves or surfaces. Re-orienting normals and directly deforming the curve in normal direction introduces discontinuities to the curve. This problem can be solved by deforming the control polygon or control mesh, as presented in this chapter. Therefore, control structure-based methods are at a distinct advantage for feature preservation. If adequately oriented normals are provided, interaction with the surface model becomes much more intuitive for industrial surface designers: pulling on a surface in normal direction makes a feature bigger, and pushing on it makes it smaller, no matter where on the surface the feature is located, and in which direction it bends.

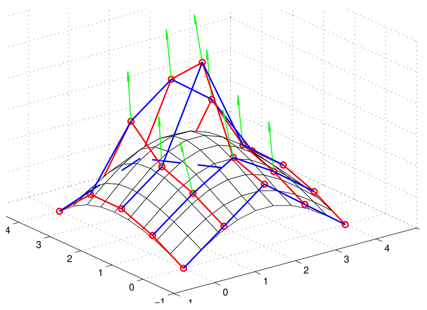
The second technique introduces the concept of local influence to all techniques except polygon-based normals which are local by nature. Instead of basing the normal computation on the entire surface (e.g. searching through all points when trying to find the closest one), only points that are reasonably close in parameter space are considered. This has the advantage that the three-dimensional location in space can be arbitrary. While this technique is aimed at normal computation, it can serve as a (slightly more dynamic) neighbourhood definition for any neighbourhood-based technique.

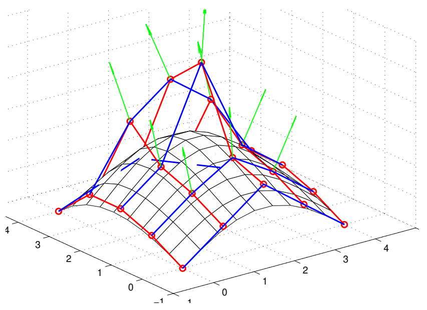

(a) Global influence mesh-based normals.

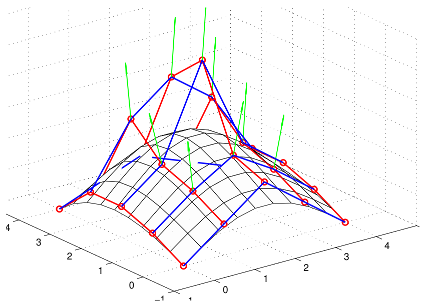

(b) Local influence mesh-based normals.


(c) Global influence parameter-based normals.


(d) Local influence parameter-based normals.


(e) Global influence direction-based normals.


(f) Local influence direction-based normals.


(g) Global influence distance-based normals.

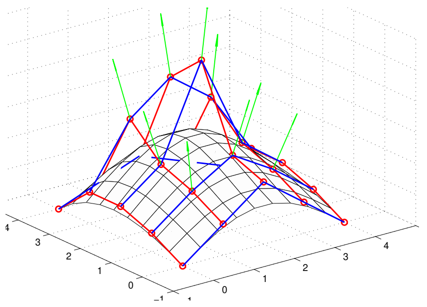
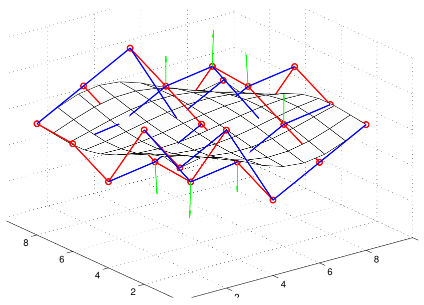
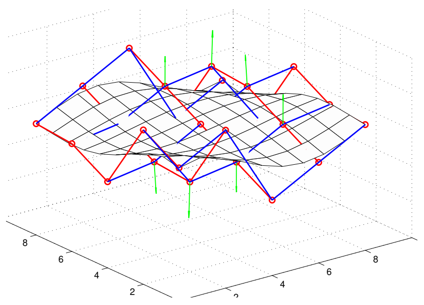

(h) Local influence distance-based normals.

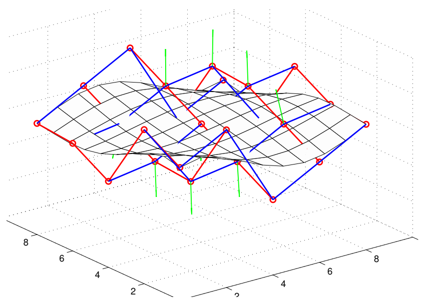
Figure 4.15: Surface mesh normals for a convex mesh using global influence and local influence. As all normals point to the same side of the surface, normal reorientation is not demonstrated in this figure.



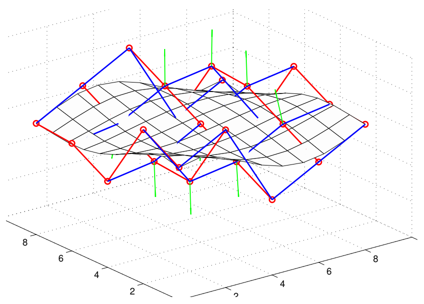
(a) Global influence mesh-based normals.



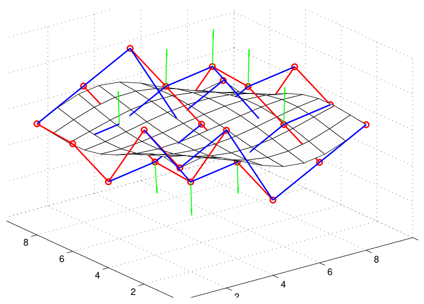
(b) Local influence mesh-based normals.



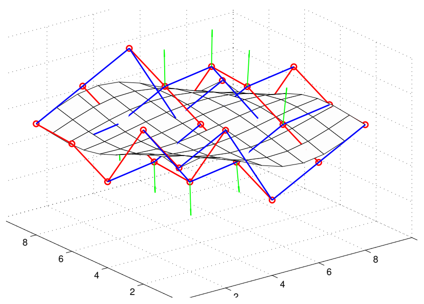
(c) Global influence parameter-based normals.



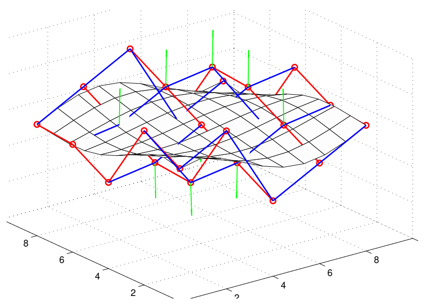
(d) Local influence parameter-based normals.



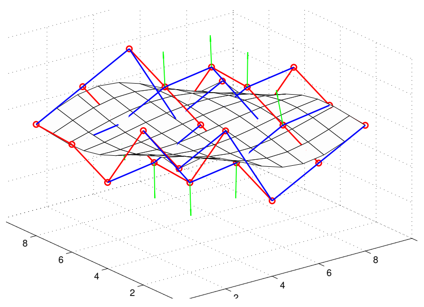
(e) Global influence direction-based normals.



(f) Local influence direction-based normals.



(g) Global influence distance-based normals.



(h) Local influence distance-based normals.

Figure 4.16: Surface mesh normals for a non-convex mesh using global influence and local influence. The normals along the boundary all point in the same direction, the normals at inner points are reoriented as described in Section 4.5.1.

Chapter 5

Adaptive Particle Relaxation for Time Surfaces

5.1 Motivation

Computer simulation has become a staple of virtual prototyping. After a masterpiece has been modeled, engineers can run simulations on the model. This serves to evaluate physical properties and behaviour of the model and its environment under different conditions, i.e. different parameter settings. In a further step the model can be tweaked to optimise properties or behaviour. This simulation-and-optimisation process is iterated until a satisfactory result has been achieved.

Simulations can be run to evaluate the model's material, or to evaluate what happens in the model's environment. One typical application is to analyse flow behaviour (e.g. air or water) around a model. First, a simulation is run to determine how a flow evolves in the proximity of an object. Then, the resulting flow field can be analysed to identify potential problems in flow behaviour. In case of undesirable behaviour, the model can be adapted to avoid the problems. To analyse flow behaviour, it is very helpful to visualise it. This can be achieved by inserting particles into the flow field and tracing their movements individually or as a group. There are a variety of different particle-based flow visualisation techniques.

Integral surfaces are a versatile tool for flow visualisation. They build on the strengths of particle-based approaches for the illustration of fluid transport, while at the same time providing a more continuous display of flow manifolds than particles or line-based techniques. Recent improvements in extraction and visualisation methodology have made flow surfaces a key analysis and visualisation tool. They are used in application areas such as the design and optimisation of aerodynamic vehicles or industrial mixing devices.

Extracting flow surfaces is a highly non-trivial task. The complexity of surface extraction and visualisation procedures is heavily influenced by properties of the flow field at hand. Complex and turbulent, time-varying flow fields, for example, frequently cause parts of the flow surface to undergo strong stretching, folding, and shearing. Since flow surfaces are typically represented as triangular meshes of flow particles, this excessive stretching and deformation of the mesh may result in degenerate or highly distorted triangles and a generally bad sampling of the actual surface. In order to avoid numerical and visual artifacts, such degenerate surface meshes are to be avoided.

A solution to this problem is given in the form of *adaptive* surface extraction techniques. These techniques dynamically insert and remove flow particles from the surface representation to maintain an appropriate mesh quality and resolution. However, such refinement strategies are not without disadvantages: they generally lead to unpredictable increases in computational complexity (advection times and load-imbalance) and memory requirements (mesh complexity).

In this chapter, I propose a novel strategy for the adaptive extraction of time surfaces in large time-varying datasets, which is partially based on [BOJH15]. Instead of performing dynamic mesh refinement during integration, I employ a relaxation method that distributes a fixed-size, large set of flow particles evenly across an accurate, bicubic representation of the time-surface. I achieve such a particle redistribution efficiently by performing a highly parallel energy-based optimisation of particle locations in 2D surface parameter space. This two-dimensional optimisation is enabled through the use of CAGD fundamentals. Metric tensors allow for the encoding of three-dimensional surface stretching in a two-dimensional surface parameter space. They can therefore be used to counteract the negative effects of surface stretching and shearing on the mesh. Gauß curvature encodes the local bending of a surface. It can therefore be used to adapt surface resolution for a better sampling of smaller features.

I demonstrate how this novel form of time surface adaptivity can contribute to a more balanced and concurrent surface extraction, and I evaluate accuracy and performance benefits of the proposed approach.

In summary, this chapter makes the following contributions to the field of flow visualisation:

- An adaptive particle system for flow surface representation.
- A technique for metric-tensor based time surface relaxation.
- A technique for curvature-based time surface relaxation.
- A method for accurate bicubic time-surface approximation.
- Techniques to optimise mesh quality.

After establishing a scientific and mathematical background in Sections 5.2 and 5.3, I detail these contributions in Section 5.4 and 5.5. Section 5.6 discusses the value of the proposed techniques in the context of multiple benchmark datasets.

5.2 Related Work

Flow surfaces are an integration-based geometric flow visualisation tool [MLP$^+$10] whose use in various application domains has increased steadily since Hultquist [Hul92] defined adaptive stream surfaces for steady flows in the 1990s.

Advection of flow surfaces through a velocity field may cause strong shape changes, including stretching and folding, requiring the implementation of surface maintenance techniques. Modern adaptive flow surface extraction techniques [GKTJ08, KGJ09, OHBKH11] perform mesh maintenance throughout the surface generation process, to balance computational costs due to integration with low surface accuracy due to insufficient resolution. Mesh adaptivity measures include edge flipping, splitting and merging and typically result in strongly increased flow surface particle numbers. Especially in regions with strong flow divergence, i.e., strong surface stretching, the resulting number of particles in the refined mesh is high and extremely hard to predict.

In typical compute environments, the involved increases in particle numbers take a significant toll on space and time complexity of adaptive surface extraction algorithms. In settings, where multi-core computing environments are used [CCG$^+$12, CKP$^+$13], such adaptive surface extraction techniques lead to increased communication between compute units and create spatially unbalanced data reads – two of the typical bottlenecks of parallel computing. The computational efficiency of such approaches, including surface extraction techniques based on Graphics Processing Units (GPU) [BFTW09], may therefore benefit from keeping the number of particles constant and their positions evenly distributed.

In this work, I propose the use of relaxation techniques to allow for this type of surface mesh adaptation. The incremental nature of time surface advection together with the availability of an underlying vector field make it possible to maintain a well-formed mesh by distributing a fixed number of flow particles in an 2D anisotropic metric tensor field. General relaxation techniques are not uncommon in visualisation, and have, for example, been extensively used for implicit surface meshing [MWK$^+$08, Mey08] or other scalar field feature extraction techniques such as crease sampling [KESW09].

Additionally, anisotropic sampling of (static) parametric surfaces is a well-researched topic in CAGD. Examples of surface refinement through particle insertion is given in work by Szelinski et al. [ST92]. Others [BH96] use metric tensors to remesh 2D domains. Both types of techniques rely on the insertion or deletion of mesh nodes, whereas I favor the redistribution of an existing set of particles. Shimada et al. [SYI$^+$97] solve the anisotropic meshing problem by packing ellipsoidal bubbles in meshes of static surfaces.

More recently, such techniques have been adapted to evolving meshes [JCNH10]. Jiao et al. propose an algorithmic technique that makes use of non-physically motivated vertex redistribution similar to Laplacian smoothing, and mesh maintenance. The latter involves edge splits and merges, effectively changing the number of overall mesh nodes/surface particles. Recently, a general metric-based particle remeshing strategy for surfaces was proposed by Zhong et al. [ZGW$^+$13].

I refer to Owen [Owe98] for a survey of mesh generation techniques that includes particle-based relaxation strategies. In visualisation, Obermaier et al. [OJ12] pack ellipses in surface parameter-space to enhance the visualisation of flow surface properties. Similar mathematical concepts may be used to improve the quality of reconstructed surfaces. Schneider et al. [SWS09], for example, use these velocity gradient tensors to produce more accurate stream surface representations by means of providing tangents for higher-order reconstruction, a goal that is also shared by the present work. I extend these previous works to efficient, adaptive time-surface generation in time-varying vector fields.

It is notable that behaviours of my surface generation technique are closely interconnected with divergence measures well-known in the flow visualisation community. Haller's [HY00] notion of Lagrangian Coherent Structures and related divergence measures have been used in one form or the other in several works focused on the visualisation of volume or surface deformations [OHBKH09, AOJ13].

5.3 Flow Surfaces

5.3.1 Basic Definition

A *flow surface* $\mathbf{S}(u,w)$, $\mathbf{S} : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^3$ is a two-manifold with *parameterisation* (u,w), whose shape is governed by advection properties of a flow field V.

In the context of this paper, I focus on a subset of flow surfaces, so called *time surfaces*. A time surface $\mathbf{S}_t$ is a flow surface that tracks the evolution of a set of flow particles $\mathbf{p}(u,w)$. Given a time-varying 3D flow field $V : \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}^3$, the shape of the surface over time is defined by advection of these flow particles. Specifically, individual positions $\mathbf{p}(u,w)$ on the surface over time evolve along path-lines

$$\mathbf{p}_t(u,w;t_0) = \mathbf{p}_{t_0}(u,w;t_0) + \int_{t_0}^{t} \mathbf{v}(\mathbf{p}_\tau(u,w;t_0),\tau)\mathrm{d}\tau\,, \tag{5.1}$$

with $\mathbf{p}_{t_0}(u,w;t_0)$ being the original position on the seed surface. In practice, this evolution of particle positions – resulting in a change of surface shape and location – are computed through numerical advection. Since continuous representations of flow surfaces are generally not available, a time surface $\mathbf{S}_t$ typically consists of a discrete set of flow particles $\mathbf{p}(u_i,w_i)$ which are advected and tracked over time. Specific continuous instances of the time-surface are represented by a triangulation of this set, the *surface mesh*. The most practicable solution to the surface meshing problem is to define an initial surface mesh for $\mathbf{S}_{t_0}$, which is then adjusted and maintained as the surface evolves over time.

5.3.2 Adaptivity Measures

The representation of time surfaces as sets of discrete flow particles leads to several challenges during surface generation. First and foremost, unsteady flow often induces significant relative stretching and shearing motions, resulting in the separation of neighbouring flow particles. Together with a mesh-based surface representation, this frequently leads to ill-formed, degenerated, or very thin or large triangles. Additionally, it is not clear how to define the desired number of discrete flow particles in a way such that the surface is optimally sampled. These observations have lead to the development of a number of adaptive surface generation techniques [GKTJ08, Hul92, KGJ09, OHBKH11]. These techniques control surface mesh quality by inserting new flow particles into the surface representation. Several heuristics may govern this insertion process. Typically, flow particles are inserted along stretched edges of the surface mesh, or on overly large triangles. Together with Delaunay-based edge flipping, this ensures that the quality of triangles in the surface mesh does not degenerate. However, these adaptivity measures have several drawbacks. First, purely mesh-based refinement leads to the introduction of errors during particle insertion, since piece-wise linear surface representations tend to be inaccurate. Second, the insertion of new triangles makes surface size, memory usage, and extraction times highly unpredictable, and – in the worst case – may lead to strongly unbalanced surfaces and particle densities. I aim at resolving these issues by keeping flow particle numbers constant and counteracting stretching by particle redistribution. This redistribution happens in $u-$direction and $w-$direction on the surface. However, points that lie on the surface border are restrained to movement parallel to the border to preserve parameter space. To further improve accuracy, particles are not moved by linearly interpolating between end points. Instead, I fit a cubic Coons patch that represents positions as well as local curvature.

5.3.3 Metric Tensors

Stretching of a parameterised surface can be represented mathematically in the form of *metric tensors*. Given two 3D surface tangents in parameter-space, $\mathbf{t}_u, \mathbf{t}_w$, the corresponding 2D metric tensor in the surface is

$$M = \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix} , \tag{5.2}$$

where $g_{ij} = \mathbf{t}_i \cdot \mathbf{t}_j$ are the components of the *first fundamental form*.

This metric tensor describes the local stretching of space in 3D with respect to a given parameterisation. As indicated in previous work [OJ12], these metric tensors are useful representations of surface stretching, deformation, and mesh quality. The ability to quantify the amount of stretching undergone by a surface allows us to develop a relaxation strategy to balance the surface sampling and improve overall surface sampling and mesh quality.

5.3.4 Surface Properties

For the remainder of the paper, I assume the availability of the following data. An initial time surface $\mathbf{S}_{t_0}$ together with a 2D parameterisation and a time-varying vector field V. The time surface consists of a set of m flow particles $\mathbf{p}$ with initial positions $\mathbf{p}_{t_0}(u_i, w_i)$, velocities $\mathbf{v}$, tangent vectors $\mathbf{t}_u, \mathbf{t}_w \in \mathbb{R}^3$ and normals $\mathbf{n}$. Additionally, the initial surface is triangulated with a surface mesh consisting of triangles, vertices (the flow particle positions) and edges. Each triangle knows all triangles that it shares an edge with, while every vertex keeps a list of neighbouring vertices and triangles.

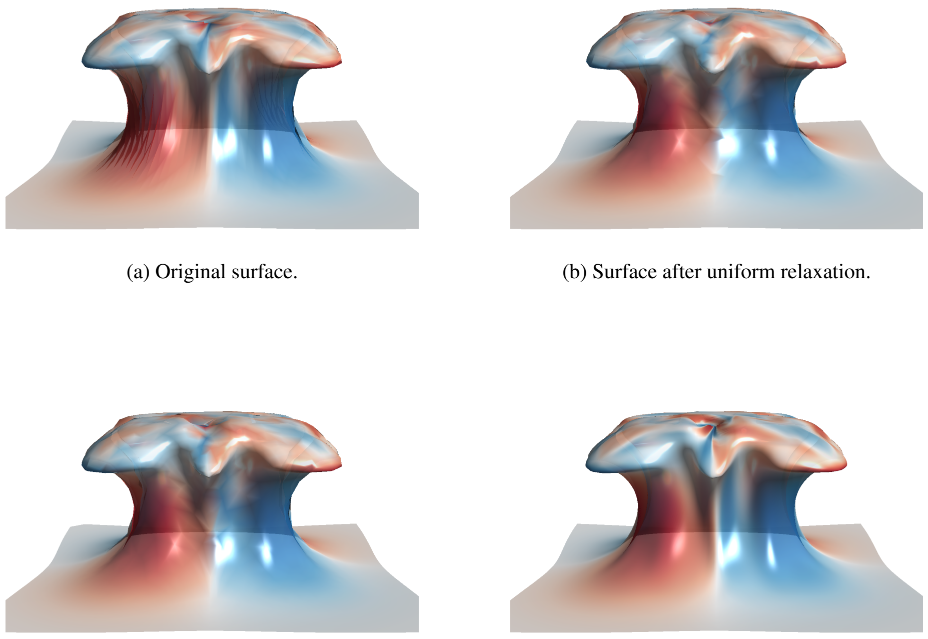
In reference to the notion to doubly-linked edge lists and to avoid confusion with vertices' neighbour triangles, I call the triangle neighbours *twins*.

5.4 Adaptive Particle Relaxation

My examples start as planar surfaces on a uniform grid, however, my method works for any triangulated and parametrised 2-manifold.

For a uniform relaxation, each particle has an energy function that is highest at its position and decreases with distance. This gives us a scalar field of energies. For a more even particle distribution, one can move the particle opposite to the gradient towards lower energy. For a feature-dependent particle distribution, the energy is based on local curvature. Instead of moving away from the gradient, I move the particle in gradient direction towards higher energy, I start with particles moving in parameter space and determine what happens to the particle in 3D.

One of the advantages of computing the change of a metric tensor is that I am able to use it to improve particle placement. Time surfaces are usually created by integrating a set of pathlines and connecting the particle positions at specific instances in time to form a surface. Since pathlines (particle traces) can diverge strongly, the surface resolution becomes unbalanced after a few advection steps.



(a) Original surface.

(b) Surface after uniform relaxation.

(c) Surface after feature-dependent relaxation.

(d) High-resolution surface.

Figure 5.1: Comparison of lower-resolution surfaces (50x50) prior to relaxation and after each relaxation technique, to a higher-resolution surface (300x300).

The advantage of a curvature-based energy function is that one can adapt the sampling to feature size, as small surface features tend to have higher curvature, and large surface features tend to have lower curvature.

Figure 5.2 shows a schematic overview of my method.

5.4.1 Particle Advection

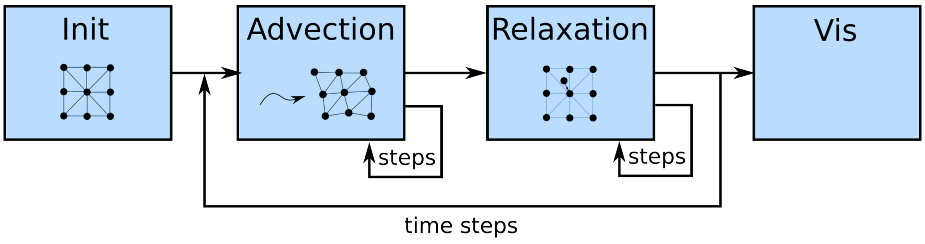
A time surface $\mathbf{S}$ in t_0 is represented by a set of m particles $\mathbf{p}_i$. In order to trace the surface through the flow field V, these particles are advected by numerical integration. For mathematical simplicty, I limit my description to simple Euler integration. Higher-order schemes may be used to increase the accuracy of particle traces.

At time t the motion of a particle (vertex of the surface mesh) with parameters $u_i, w_i \in \Omega$ and position in space $\mathbf{p}_t(u_i, w_i)$ is governed by the value of the velocity field at $\mathbf{p}$. Following Euler integration the particle position at time $t + \Delta t$ is computed as

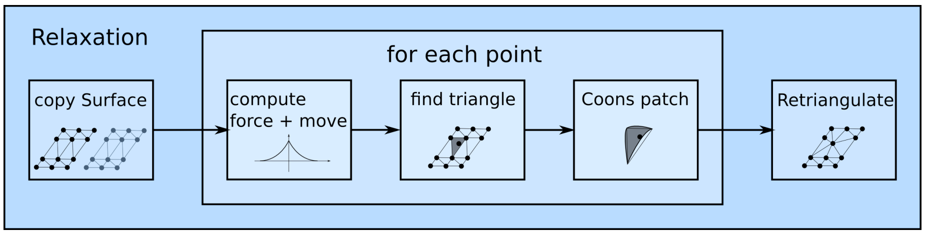
$$\mathbf{p}_{t+\Delta t}(u_i, w_i) = \mathbf{p}_t(u_i, w_i) + \Delta t \cdot \mathbf{v}(\mathbf{p}_t(u_i, w_i), t)\,.$$

Higher-order schemes may be used to increase the accuracy of particle traces. Velocity vectors at particle positions are obtained through interpolation.

In addition, there are some properties that can be derived from the surface. Tangent



(a) Workflow illustration of the proposed relaxation-based time-surface extraction process.



(b) The surface relaxation step consists of multiple tasks.

Figure 5.2: Workflow illustration of the proposed relaxation-based time-surface extraction process.

vectors of flow particles $\mathbf{t}_u, \mathbf{t}_w$ may also be integrated alongside the velocity field. They define the normal $\mathbf{n} = \mathbf{t}_u \times \mathbf{t}_w$.

To integrate these tangents, I make use of the velocity gradient tensor $\nabla \mathbf{v}(\mathbf{p}_i)$, where $\mathbf{p}_x^+ = \mathbf{p}_i + (\varepsilon, 0, 0)^T$, $\mathbf{p}_x^- = \mathbf{p}_i - (\varepsilon, 0, 0)^T$ etc. the velocity gradient tensor is obtained through finite differencing as

$$\nabla \mathbf{v}(\mathbf{p}_i) = \left(\frac{\mathbf{v}(\mathbf{p}_x^+) - \mathbf{v}(\mathbf{p}_x^-)}{2\varepsilon}, \quad \frac{\mathbf{v}(\mathbf{p}_y^+) - \mathbf{v}(\mathbf{p}_y^-)}{2\varepsilon}, \quad \frac{\mathbf{v}(\mathbf{p}_z^+) - \mathbf{v}(\mathbf{p}_z^-)}{2\varepsilon} \right)^T .$$

For points at the boundary of the dataset, I use forward/backward differencing. The evolution of tangent vectors during advection may now be computed as (c.f., [OJ12])

$$\mathbf{t}_{u;t+\Delta t} = \mathbf{t}_{u;t} + \Delta t \cdot \langle \nabla \mathbf{v}(\mathbf{p}_i), \mathbf{t}_{u;t} \rangle \tag{5.3}$$

$$\mathbf{t}_{w;t+\Delta t} = \mathbf{t}_{w;t} + \Delta t \cdot \langle \nabla \mathbf{v}(\mathbf{p}_i), \mathbf{t}_{w;t} \rangle . \tag{5.4}$$

The availability of accurate flow particle positions and tangent vectors allows for precise modeling of the extracted time-surfaces. While positions are generally enough to construct piece-wise constant surface meshes, the availability of correct tangent information facilitates the computation of tangent-space stretching as well as the construction of higher-order surface representations. The particle information obtained during advection is the basis for the proposed relaxation procedure.

5.4.2 Particle Relaxation

During advection, neighbouring flow particles may have separated, or moved closer together, resulting in spatially varying particle densities across the surface. Often, this leads to a less balanced surface and flow field sampling. I aim to improve surface sampling using two different methods to redistribute flow particles across the surface in procedures which are termed *particle relaxation* in the following.

Uniform Particle Relaxation achieves a more uniform surface sampling by employing a density-based relaxation function. This function employs a deformation tensor to counteract surface shearing and stretching resulting from particle advection.

Feature-Dependent Particle Relaxation achieves a feature-dependent surface sampling by employing a curvature-based relaxation function. This function redistributes particles from low-curvature areas with few (and large) features to high-curvature areas with smaller features.

Since particles move in surface tangent space during relaxation, I am able to perform the involved relaxation procedures in a 2D space, followed by a projection onto a local, accurate 3D representation of the surface. Both methods compute a local scalar field in the particle neighbourhood, and they perform relaxation in 2D surface parameter space based on its gradient (see Figure 5.2(b)). This scalar field encodes the energy which can be based on the neighbourhood of a particle. In energy-based particle relaxation procedures, particle positions are optimised based on a globally nonlinear energy minimisation scheme. Mathematically, I estimate this energy function locally as follows.

Given a particle in parameter-space $\mathbf{p}_i = (u_i, w_i)$ with n neighbours $\mathbf{p}_j = (u_j, w_j)$, the *energy* e at $\mathbf{p}_i$ is given as a summation of 2D energy kernels E:

$$e(\mathbf{p}_i) = \sum_{\mathbf{p}_j \in N_2(\mathbf{p}_i)} E(\mathbf{p}_i, \mathbf{p}_j), \tag{5.5}$$

where N_2 is the two-ring of neighbours as given by Definition 2.4.

Based on these energy kernels, a direction of force $\mathbf{f}_i$ along which the particle is moved can be computed. This direction is equal to the negative gradient direction of e:

$$\mathbf{f}_i = -\bigtriangledown e = -\begin{pmatrix} \frac{\partial e(\mathbf{p}_i)}{\partial u} \\ \frac{\partial e(\mathbf{p}_i)}{\partial w} \end{pmatrix}. \tag{5.6}$$

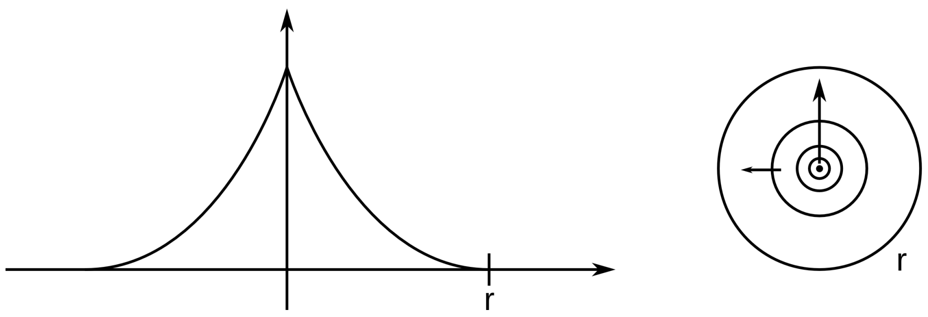


Figure 5.3: An illustration of the force function $\mathbf{f}_i$ in 1D and 2D. The 2D representation includes isocontours and examples of inverse gradient directions.

An explicit form of this gradient can be derived directly from Equation (5.5). A schematic representation is presented in Figure 5.3.

$$\frac{\partial e(\mathbf{p}_i)}{\partial u} = \frac{\partial}{\partial u} \sum_{\mathbf{p}_j \in N_2(\mathbf{p}_i)} E(\mathbf{p}_i, \mathbf{p}_j) = \sum_{\mathbf{p}_j \in N_2(\mathbf{p}_i)} \frac{\partial}{\partial u} E(\mathbf{p}_i, \mathbf{p}_j), \tag{5.7}$$

$$\frac{\partial e(\mathbf{p}_i)}{\partial w} = \frac{\partial}{\partial w} \sum_{\mathbf{p}_j \in N_2(\mathbf{p}_i)} E(\mathbf{p}_i, \mathbf{p}_j) = \sum_{\mathbf{p}_j \in N_2(\mathbf{p}_i)} \frac{\partial}{\partial w} E(\mathbf{p}_i, \mathbf{p}_j). \tag{5.8}$$

Computing this gradient allows me to move particles in parameter-space. Each iteration of particle relaxation accelerates the particle according to $\mathbf{f}_i$

$$\mathbf{v}_{i+1} = \mathbf{v}_i + \mathbf{f}_i \cdot \text{step}_{\text{relax}}, \tag{5.9}$$

and causes immediate displacement in parameter-space as

$$\mathbf{p}_{i+1} = \mathbf{p}_i + \mathbf{v}_{i+1} \cdot \text{step}_{\text{relax}}. \tag{5.10}$$

The relaxation step width $\text{step}_{\text{relax}}$ can be varied to adapt the amount of change in each iteration of the relaxation.

$$\text{step}_{\text{relax}} = \frac{\max\text{edge} + \min\text{edge}}{2} \cdot c,$$

where $c = 0.08 \cdot \mathbf{f}_{\max}$.

Note that, because values of e itself are not required for the proposed relaxation techniques, it is sufficient to give a definition of ∇E.

To constrain particle movement to surface parameter-space Ω, I impose the following restrictions on particle movement:

Corners Corner particles remain fixed in parameter-space

Borders The movement of particles located on the boundary of surface parameter-space is constrained to displacement parallel to the boundary.

These two restrictions ensure an accurate representation of $\partial\Omega$, and at the same time they constrain particle movements to inner regions of parameter space (i.e., avoid an explosion of the particle system). For some datasets, border particles require additional weights to avoid accumulating particles along borders.

Uniform Particle Relaxation

The goal of uniform particle relaxation is to counteract the deteriorating sampling caused by particle advection, and to achieve a uniform sampling of the surface. As pointed out in Section 5.3.3, metric tensors encode surface stretching and shearing. Since I want to counteract this stretching and keep the particles well-distributed on the surface, I use the conjugate metric tensor T as a deformation tensor to adapt spherical energy kernels:

$$T = M^{-1} = \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix}^{-1} = \frac{1}{\det(g_{jk})} \begin{pmatrix} g_{22} & -g_{12} \\ -g_{21} & g_{11} \end{pmatrix}. \tag{5.11}$$

Spherical energy kernels E, e.g., isotropic Gaussian functions, directly represent local particle density as energy. A higher value in energy indicates a denser particle distribution, and a lower value in energy indicates a sparse particle distribution. As a consequence, moving particles away from regions with a relatively high density automatically results in more even particle distributions on the surface.

Let $\mathbf{p}_{ij} = \mathbf{p}_j - \mathbf{p}_i$ be the directional vector of $\mathbf{p}_i$ to its neighbour, then

$$\nabla E(\mathbf{p}_i, \mathbf{p}_j) = d(\mathbf{p}_i, \mathbf{p}_j)^2 \cdot \frac{\mathbf{p}_{ij}}{\|\mathbf{p}_{ij}\|} \tag{5.12}$$

is the energy kernel of $\mathbf{p}_j$ influencing $\mathbf{p}_i$, and the function

$$d(\mathbf{p}_i, \mathbf{p}_j) = r \cdot c \cdot \sqrt{\frac{\mathbf{p}_{ij}}{\|\mathbf{p}_{ij}\|}^T \cdot T_i \cdot \frac{\mathbf{p}_{ij}}{\|\mathbf{p}_{ij}\|}} - \|\mathbf{p}_{ij}\|, \tag{5.13}$$

where $c = \frac{45}{\pi r^6}$ is constant, gives a distance-based energy which depends on local surface distortion.

The scaling radius r is determined by relative scaling of surface to parameter-space. It depends on the number of points in u- and w-direction: $r = a \cdot (\frac{1}{\#u} + \frac{1}{\#w})$, where a depends on the dataset.

This kernel function is conceptually similar to quadratic (spiky) gradient functions found in previous work (e.g., Clavet et al. [CBP05]) and is illustrated in Figure 5.3. However, introducing the conjugate metric tensor generally creates anisotropic density kernels. Including this tensor has the following effect: particles that separate in 3D cause a stretching of parameter-space. This stretching is directly represented by the metric tensor. Its conjugate represents the inverse of this stretching. As a consequence, the radius of the kernel function is compressed in directions of stretching (causing particles to move closer to one another) and extended in directions of contraction (causing particles to move further apart). This situation is depicted in Figure 5.4: particles move away from high-density areas. Through the deformation of the radial energy function, density increases further in regions particles were advected to, and it decreases in the direction they came from. For more intense stretching through advection, the density is decreased more. This leads to particle movement in the opposite direction of previous stretching, and therefore, to a more uniform particle distribution.

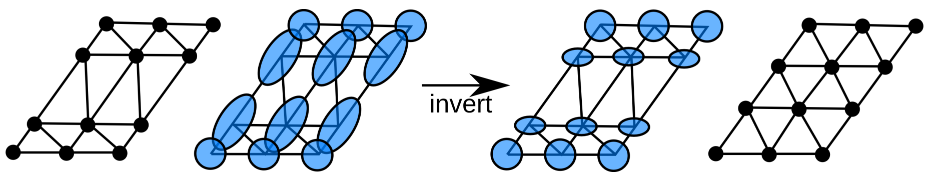


Figure 5.4: Role of the metric tensor (shown as ellipses) and its inverse during particle relaxation.

Figure 5.5 illustrates what the density function looks like. Some particles are highlighted for illustrative purposes. For each of its neighbours, the density function is represented as a fuzzy ellipsoid. Particles are represented by glyphs.

Feature-Dependent Particle Relaxation

The goal of feature-dependent particle relaxation is to achieve a feature-dependent surface sampling. Fine features should be sampled with a finer resolution than coarse features. Surface features occur where the surface bends. This bending results in an increased local curvature.

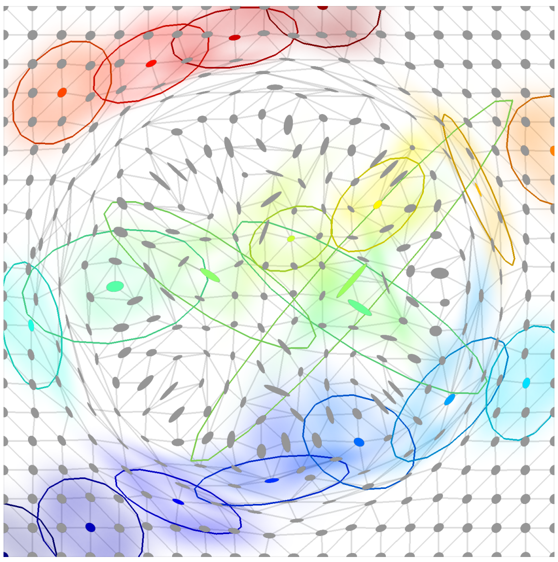


Figure 5.5: Glyph visualisation of particles in parameter space. Some particles are surrounded by an ellipsoidal representation of their force function.

Gauß curvature is a useful tool to measure and compare curvatures of different surface points. It is defined as

$$K(\mathbf{p}) = \kappa_1(\mathbf{p}) \cdot \kappa_2(\mathbf{p}),$$

where $\kappa_1(\mathbf{p}), \kappa_2(\mathbf{p})$ are the principal curvatures of a surface at point $\mathbf{p}$.

For local minima and maxima, Gauß curvature is greater than zero, whereas for saddle points, it is less than zero. If it is exactly zero, the surface point is either parabolic (i.e. it bends in one direction but is flat in the other direction), or it is flat.

A large Gauß curvature signifies large principal curvatures which occur in tight folds where there are small osculating ellipsoids. A small Gauß curvature signifies at least one small principal curvature which occurs in slight bends with large osculating ellipsoids.

As the algorithm operates on a mesh-based surface, a discrete version of Gauß curvature is needed. There are a variety of different definitions. The most common is given as follows [LP82, KKL02, MDSB03, LLV05, Xu06]:

$$K(\mathbf{p}) = \frac{2\pi - \sum_{i=1}^{n} \alpha_i}{\frac{1}{3}\sum_{i=1}^{n} A_i},$$

where n edges span the one-ring of neighbours around $\mathbf{p}$, α_i is the angle between two neighbouring edges, and A_i are the areas of spanned triangles.

Based on this definition, one can build a curvature-based energy Kernel:

$$E(\mathbf{p}_i, \mathbf{p}_j) = -\frac{2 \cdot K_{|\max|}}{|K(\mathbf{p}_i) - K(\mathbf{p}_j)|} \cdot (\mathbf{p}_j - \mathbf{p}_i),$$

where $K(\mathbf{p}_i), K(\mathbf{p}_j)$ are the Gauß curvatures of $\mathbf{p}_i, \mathbf{p}_j$, and

$$K_{|\max|} = \max(\max_i |K_(\mathbf{p}_i)|, 1)$$

is the maximal absolute Gauß curvature of the surface.

As for uniform relaxation, the energy kernel creates a spiky gradient function, as sketched in Figure 5.3. The scaling factor $\frac{2 \cdot K_{|\max|}}{|K(\mathbf{p}_i) - K(\mathbf{p}_j)|}$ depends on curvature. Since $K_{|\max|} > 1$ and $s := |K(\mathbf{p}_i) - K(\mathbf{p}_j)| \leq 2 \cdot K_{|\max|}$, it is always greater than one.

The avid reader will wonder if the inverse, $s^{-1} = \frac{|K(\mathbf{p}_i) - K(\mathbf{p}_j)|}{2 \cdot K_{|\max|}}$, would produce good results. However, $|K(\mathbf{p}_i) - K(\mathbf{p}_j)|$ can take values greater as well as less than 1. If $|K(\mathbf{p}_i) - K(\mathbf{p}_j)| > 1$, it produces strong oscillations in the energy function, which makes them numerically unstable. Using s rather than s^{-1} results in a damped oscillation [PM88], and therefore, in a numerically stable energy function.

As a result of this method, there are many small triangles in regions of high curvature in order to preserve the features present in these regions. In exchange, there are fewer and larger triangles in regions of low curvature in order to save storage and to preserve the total particle count. As the features are coarse, they do not require the same sampling as finer features.

5.4.3 Particle Placement

In order to determine the relaxed 3D position of a particle, I need to identify the containing surface mesh triangle.

I am looking for the triangle that contains the new parametric coordinates of a particle as determined in Equation (5.10). To locate this triangle, I examine the neighbour triangles of the particle and perform a 2D point-in-triangle test for the new parameters $(u'_i, w'_i)^T$. If successful, I estimate the new 3D position of the particle by performing a local fitting of a smooth Coons patch as described in the following section. If the point is not contained in any triangle of this neighbourhood, it is not moved. This ensures locally constrained evolution of the particle set during each iteration of the relaxation procedure.

Local Coons Patch Fitting

Once the correct triangle has been located, I fit a cubic triangular Coons Patch as defined by Farin [Far02b, pp. 414 – 416] into the triangle. Construction of the Coons patch requires the 3D coordinates of triangle vertices as well as tangent information along the edges. I further assume knowledge of the barycentric coordinates $\mathbf{b} = (b_0, b_1, b_2)$ of $\mathbf{p}_k$ in relation to the triangle defined by $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$ in parameter space.

The tangents of a Hermite curve along the edge from point $\mathbf{p}_i$ to point $\mathbf{p}_j$ should lie in the tangent plane of their respective end points – the corner points of the surface patch. For each edge, I can express 2D tangents using a linear combination $(u_{ij}, w_{ij})^T$ which I apply to the 3D tangents at the end points to obtain tangent vectors $\mathbf{t}_u, \mathbf{t}_w$ in edge direction.

With these new tangent vectors $\mathbf{t}_u, \mathbf{t}_w$, I am able to define cubic Hermite curves on each edge between end points $\mathbf{p}_i, \mathbf{p}_j$. These curves interpolate end point positions and tangents, leading to an accurate, locally C^1 representation of the surface.

I make use of the standard cubic Hermite base functions:

$$\begin{aligned}
H_0^3(r) &= B_0^3(r) + B_1^3(r) &&= 2r^3 - 3r^2 + 1\,,\\
H_1^3(r) &= \frac{1}{3}B_1^3(r) &&= r^3 - 2r^2 + r\,,\\
H_2^3(r) &= -\frac{1}{3}B_2^3(r) &&= r^3 - r^2\,,\\
H_3^3(r) &= B_2^3(r) + B_3^3(r) &&= -2r^3 + 3r^2\,.
\end{aligned}$$

To interpolate along the edges of the triangle, I interpolate using relative barycentric coordinates:

$$r = \frac{b_1}{b_1 + b_2}\,,\quad s = \frac{b_2}{b_0 + b_2}\,,\quad t = \frac{b_1}{b_0 + b_1}\,.$$

This results in Hermite curves along each patch boundary:

$$\begin{aligned}
\mathbf{C}_0(r) &= \mathbf{p}_2 \cdot H_0^3(r) + \mathbf{t}_2 \cdot H_1^3(r) + \mathbf{t}_1 \cdot H_2^3(r) + \mathbf{p}_1 \cdot H_3^3(r)\,,\\
\mathbf{C}_1(s) &= \mathbf{p}_0 \cdot H_0^3(s) + \mathbf{t}_0 \cdot H_1^3(s) + \mathbf{t}_2 \cdot H_2^3(s) + \mathbf{p}_2 \cdot H_3^3(s)\,,\\
\mathbf{C}_2(t) &= \mathbf{p}_0 \cdot H_0^3(t) + \mathbf{t}_0 \cdot H_1^3(t) + \mathbf{t}_1 \cdot H_2^3(t) + \mathbf{p}_1 \cdot H_3^3(t)\,.
\end{aligned}$$

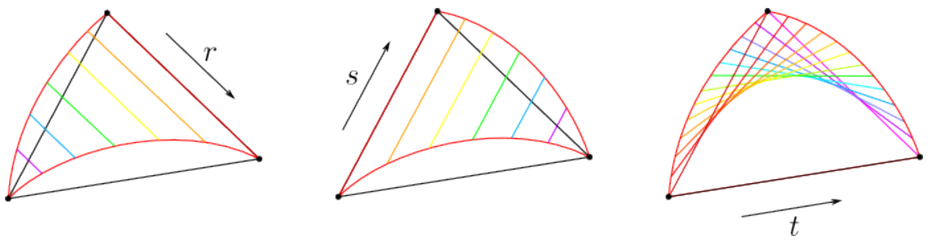


Figure 5.6: Ruled surfaces are constructed between each pair of boundary curves.

Now, I can define ruled surfaces between each pair of boundary curves, as illustrated in Figure 5.6:

$$\begin{aligned}
\mathbf{S}_0(r) &= (1-r) \cdot \mathbf{C}_1(s) + r \cdot \mathbf{C}_2(t)\,,\\
\mathbf{S}_1(s) &= (1-s) \cdot \mathbf{C}_2(t) + s \cdot \mathbf{C}_0(r)\,,\\
\mathbf{S}_2(t) &= (1-t) \cdot \mathbf{C}_1(s) + t \cdot \mathbf{C}_0(r)\,.
\end{aligned}$$

and construct the Coons surface patch (Figure 5.7) as a convex combination of ruled surfaces:

$$\mathbf{S}(\mathbf{b}) = b_0 \cdot \mathbf{S}_0 + b_1 \cdot \mathbf{S}_1 + b_2 \cdot \mathbf{S}_2\,.$$

This local surface approximation makes use of accurate tangent and position information and produces smoother surface representations than standard piece-wise linear meshes. Using this higher-order surface representation ensures that positions of relaxed particles lie close to the true time-surface location. This yields a much more faithful, C^1-continuous representation of the surface to the influence of tangents on surface shape and on the transition between neighbour patches. Note that moving particles on a piece-wise linear representation of the surface would introduce significant errors to relaxed surfaces.

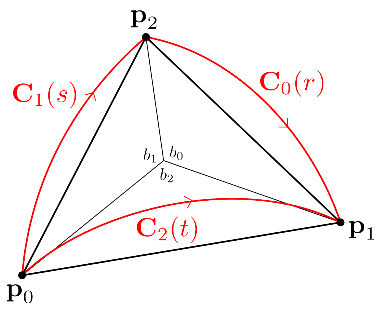


Figure 5.7: The 3D position of a point in parameter space can be determined by using the same barycentric coordinates.

5.4.4 Particle Update

Once the new position of the point on the surface is determined, I interpolate tangents from the available surface information and adapt the particle's parameters. Points on the border of the time surface are moved along the border. They are moved along the Hermite curves $\mathbf{C}_i$ defined in the previous section. This avoids growth or shrinkage during relaxation.

If the new position is on the edge that is shared between the current triangle and one of its neighbours, this shared edge is flipped to maintain a well-formed mesh. This is carried out by the same criteria which are used for mesh quality.

5.5 Mesh Quality Preservation

Mesh quality is an important property of triangulated surfaces. Long and thin triangles are prone to degeneration. This makes it difficult to compute numerically correct normals. As seen in the previous chapters, normals are powerful tools for virtual prototyping. Furthermore, they are used for a variety of other tasks such as producing visually smooth renderings of the virtual prototype. Rendering methods, like Phong shading or raytracing [BB06], rely on normals to determine how light interacts with the surface.

5.5.1 Delaunay Condition

The Delaunay condition is one of the most common criteria to judge mesh quality. It is used for Delaunay triangulation, a method that creates an optimal triangulation for a given set of (planar) points [DBVKOS00]. The Delaunay condition is defined on a pair of triangles which share an edge, as illustrated in Figure 5.8. To assess the quality of a triangulation, the two opposing angles φ_1, φ_2 are examined. The Delaunay condition is fulfilled if

$$\varphi_1 + \varphi_2 \leq 180^\circ$$

holds. If the sum of opposing angles is greater than 180°, triangulation of the four corner points is not optimal as the triangles are narrow. To improve the triangulation, the edge can be flipped, as demonstrated in Figure 5.8.

In order to avoid oscillating edges, i.e. frequent back and forth flipping of edges, I relax the condition as follows:

$$\varphi_1 + \varphi_2 \leq 190^\circ .$$

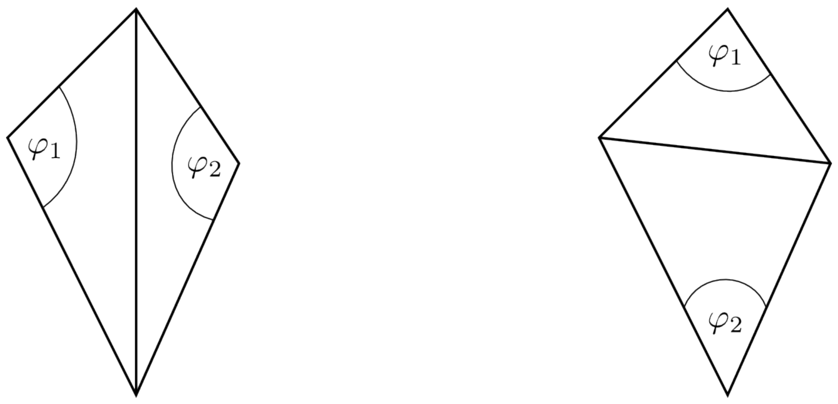


Figure 5.8: Delaunay criterion for neighbouring triangles. If the sum of φ_1 and φ_2 is greater than 180°, the edge is flipped.

After each relaxation step, this criterion is checked for all vertex neighbourhoods. To avoid artefacts, both triangles are marked as flipped and cannot be flipped again during this relaxation iteration. Retriangulation is computationally expensive, and allowing several flips per triangle would further increase computational cost. Over the course of a number of relaxation steps, it is possible for the triangulation to progress towards a globally improved state.

5.5.2 Mean Curvature

The Delaunay condition was developed for triangulation in a plane. For triangulated surfaces which bend in space, this is more challenging. Blindly flipping normals regardless of surface properties leads to a loss of three-dimensional surface features like bulges or cavities. To avoid these issues, I require that convex edges remain convex, and concave edges remain concave.

This can be achieved by considering the mean curvature H at an edge. If it is greater than zero, there is a bulge, and if it is less than zero, there is a cavity. If it is equal to zero, the two triangles lie inside a plane.

I compute discrete mean curvature for edge **e** as defined by [Bou97]:

$$H_{\mathbf{e}} = \frac{1}{2}\vartheta\|\mathbf{e}\|,$$

where ϑ is the angle between the triangle normals, as illustrated in Figure 5.9.

The resulting triangulations can exhibit an increase or decrease in the size of surface features, but convex surface parts do not become concave, and concave surface parts do not become convex.

5.6 Results and Discussion

I present results obtained by applying the proposed relaxation-based time-surface extraction algorithm to the following three benchmark datasets.

Jet Stream The Jet Stream dataset simulates the effects of four high-velocity jet streams. The dataset contains large amounts of turbulence along lateral regions of

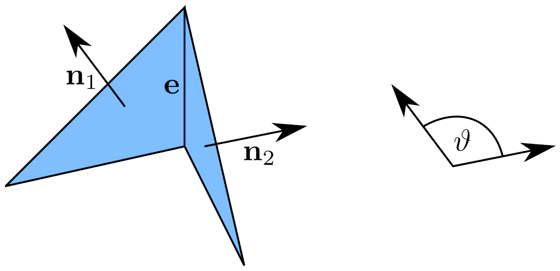


Figure 5.9: Discrete mean curvature of an edge depends on the dihedral angle between adjacent triangles [Bou97].

the streams. The dataset is represented as $128 \times 256 \times 128$ cells in a uniform grid with 5 time steps.

Dam Break The Dambreak dataset is a two-phase flow simulating the collapse of a column of water under gravitational forces. The motion of phases (water and air) are encoded in a joint vector field. The collapse of the water column creates significant distortion effects at the fluid interface, providing an environment for time-surface extraction. The dataset is a uniform grid with $68 \times 68 \times 68$ cells and 91 time steps.

Rayleigh-Taylor Instability The Rayleigh-Taylor Instability simulation models the effects of motion along a fluid interface with a large density discontinuity. The typical chaotic nature of Rayleigh-Taylor Instabilities forms multiple seemingly random Rayleigh-Taylor “fingers” at the fluid boundary, resulting in significant stretching of the interface. The dataset contains $68 \times 68 \times 68$ cells and 61 time-steps.

5.6.1 Performance Statistics

All performances evaluations were carried out on a laptop with an Intel® Core$^{\text{TM}}$ i7-2860QM CPU (8 cores with 2.50GHz) and 16GB RAM, using a 64 bit Linux. Note that these measurements presented here do not reflect the performance of optimised code, but that of a prototype implementation with basic parallelisation.

Table 5.1 presents average run times for advection and relaxation over all time steps of each dataset. Surface advection is consistently fast at only 0.035 seconds of advection time per time step. On average, a single relaxation step takes between twice as long and three times as long as surface advection. Uniform relaxation is slightly faster than feature-dependent relaxation, and both techniques vary depending on the dataset.

5.6.2 Quantitative Analysis

To assess the quality of the resulting surfaces, I use quality metrics based on the minimal angle $\varphi_{\min}(\mathbf{T}_i)$ of each triangle $\mathbf{T}_i$ in the mesh. I compare surfaces prior to relaxation with surfaces after the application of each relaxation method.

As a first quality metric for a surface $\mathbf{S}$ with n triangles, I am interested in the worst case, so I examine the smallest angle of the entire mesh. The smallest angle of a surface $\mathbf{S}$ can be computed as

$$\varphi_{\min}(\mathbf{S}) = \min_i \varphi_{\min}(\mathbf{T}_i)\,.$$

Dataset	**Advection**	**Uniform Relaxation**		**Feature-Dependent Relaxation**	
		1 step	50 steps	1 step	50 steps
Jet	0.037 s	0.097 s	4.85 s	0.099 s	4.95 s
Dam Break	0.041 s	0.084 s	4.20 s	0.091 s	4.56 s
Rayleigh-Taylor	0.033 s	0.084 s	4.18 s	0.103 s	5.14 s
Average	0.035 s	0.088 s	4.41 s	0.098 s	4.89 s

Table 5.1: Run times of advection and both relaxation techniques for all datasets at a resolution of 50x50 points.

As a second quality metric, I am interested in the average case, so I examine the average minimal angle of all triangles. The average minimal angle of all triangles of a surface **S** can be computed as

$$\varphi_{\text{avg}}(\mathbf{S}) = \frac{1}{n}\sum_{i=1}^{n} \varphi_{\min}(\mathbf{T}_i)\,.$$

In Table 5.2, I present the smallest minimal angles and the average minimal angles for each dataset. I compare the angles of unrelaxed surfaces with those of surfaces after relaxation using either of the techniques. As one can see, the overall mesh quality improves significantly: in almost all cases, both the smallest minimal angle and the average minimal angle increase through relaxation. The only exception to this happens for the Dam Break dataset during uniform relaxation: two particles move towards each other independently, resulting in a very narrow triangle. Restricting particle movement to the one-ring of neighbours avoids complete degeneration. Furthermore, mesh optimisation can usually repair the triangulation in such cases through an edge flip. However, the very strong stretching that takes place where the surface folds over makes it difficult to repair the triangulation.

Relaxation	**Jet Stream**		**Dam Break**		**Rayleigh-Taylor**	
	$\varphi_{\min}(\mathbf{S})$	$\varphi_{\text{avg}}(\mathbf{S})$	$\varphi_{\min}(\mathbf{S})$	$\varphi_{\text{avg}}(\mathbf{S})$	$\varphi_{\min}(\mathbf{S})$	$\varphi_{\text{avg}}(\mathbf{S})$
Original	0.59	28.74	0.62	35.43	9.45	35.85
Uniform	2.31	37.92	0.04	38.42	12.22	41.12
Feature-dependent	6.66	38.00	0.89	38.41	12.71	40.71

Table 5.2: Smallest minimal angles and average minimal angles over all triangles of the surface mesh. Each surface mesh has a resolution of 50x50 points.

The biggest improvement can be seen for the Jet Stream dataset, while the highest mesh quality is achieved for the Rayleigh-Taylor dataset. The average minimal angle increased from $\varphi_{\text{avg}}(\mathbf{S}) \in [28.74, 35.85]$ to $\varphi_{\text{avg}}(\mathbf{S}) \in [37.92, 41.12]$ for uniform relaxation and $\varphi_{\text{avg}}(\mathbf{S}) \in [38.00, 40.71]$ for feature-dependent relaxation.

A direct comparison between uniform relaxation and feature-dependent relaxation shows that for the Jet Stream dataset, feature-dependent relaxation yields better results, whereas for the Rayleigh-Taylor Instability dataset, uniform relaxation performs better. For the Dam Break dataset, the average minimal angles are almost identical. The smallest minimal angle improves with respect to the unrelaxed surface in the feature-dependent case, while for the uniform case, it deteriorates. Overall, the results are very similar, and the choice of an optimal relaxation method depends on the dataset.

5.6.3 Qualitative Analysis

Quantitative analysis can give some insight into the quality of results. However, it is difficult to judge how well-balanced or poorly balanced a surface sampling is without considering visuals.

In this section, I give a side-by-side comparison of surface meshes before, during, and after relaxation. Results of both strategies – uniform relaxation and feature-dependent relaxation – are shown in these figures.

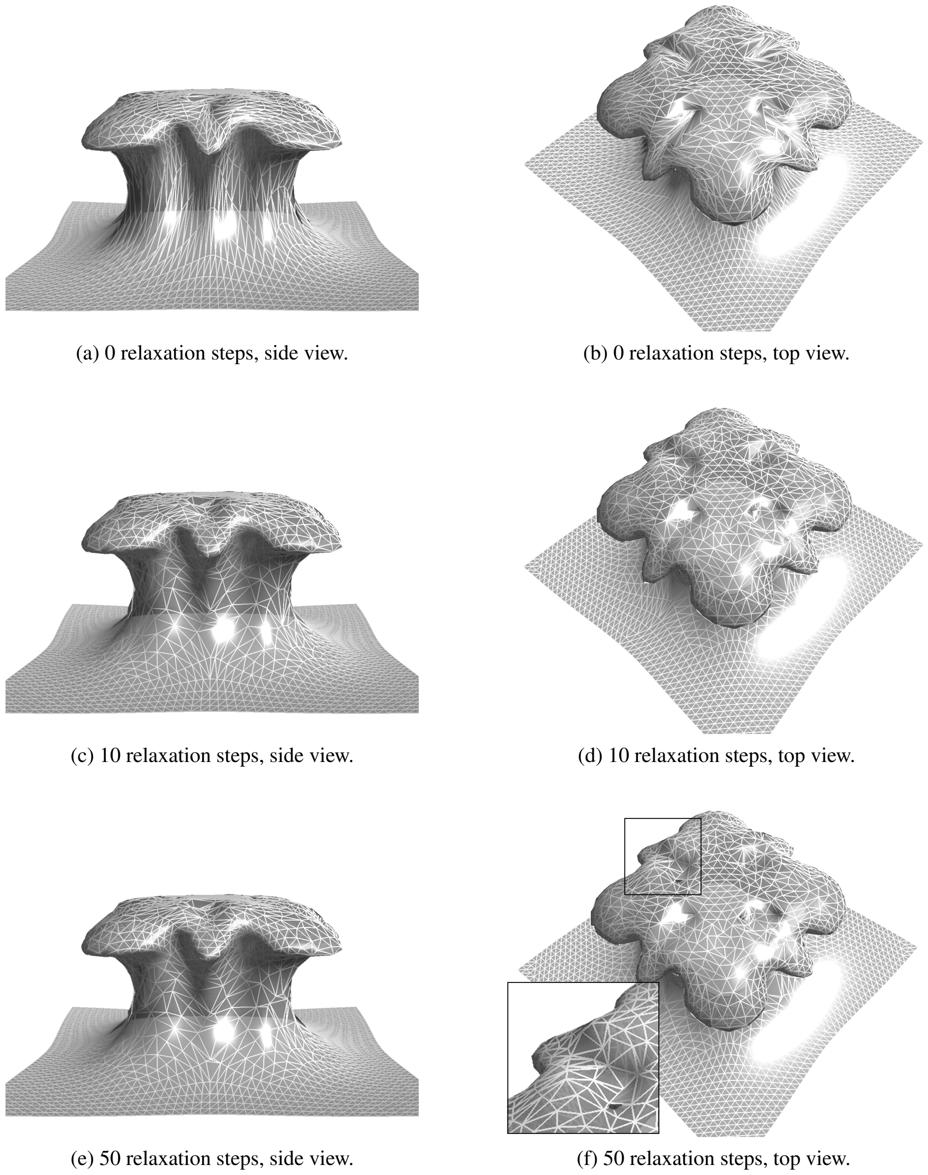

(a) 0 relaxation steps, side view.
(b) 0 relaxation steps, top view.
(c) 10 relaxation steps, side view.
(d) 10 relaxation steps, top view.
(e) 50 relaxation steps, side view.
(f) 50 relaxation steps, top view.

Figure 5.10: Jet during uniform relaxation for a surface of resolution 50x50.

A detailed overview of differences between the methods can be seen in Figures 5.10

and 5.11, in which I applied both relaxation techniques to the Jet Stream dataset. In both figures, the top row shows the surface immediately after advection, the middle row shows results after 10 relaxation steps, and the bottom row shows results after 50 relaxation steps. The most significant change can be observed for the switch from the surface prior to relaxation to the surface after 10 relaxation steps. On the neck of the plume, the triangulation changes tremendously from very long and thin triangles which are stacked next to each other, to a much more balanced triangulation with well-distributed particles. After relaxation, the discretisation gives a much better surface sampling than prior. On top of the plume, turbulence in the vector field has led to the formation of pockets. Both relaxation techniques result in a much better triangulation in these areas.

While the results of uniform relaxation and feature-dependent relaxation are very similar in most areas of the jet stream surface, they differ in the pockets, as shown in Figures 5.10f and 5.11f. Here, uniform relaxation evens out the particle distribution, which leads to a slight reduction in pocket depth. Feature-dependent relaxation, on the other hand, was designed especially for such features. As expected, it preserves pocket depth much better.

In Figure 5.12, I present the results of applying each relaxation technique to the Dambreak dataset. Part of the surface remains upright, and another part bends down (seen in the left column) and folds back to curl further down (seen in the right column) and flow down towards the front (not pictured). There are two regions in which the two methods differ greatly. The first region is located where the surface bends down (pictured in the left column). On this part of the surface, there are a lot of narrow triangles, however, the curvature is rather low. Therefore, uniform relaxation spreads out the triangles over a larger area to counteract the stretching, whereas feature-dependent relaxation mainly focuses on the surface part closer to the fold as the curvature increases in this area. The second region is located where the surface curls up in the back (pictured in the right column), and in the crease between the bent part and the upright part of the surface. Again, uniform relaxation spreads out triangles to counteract the compression taking place during advection. However, feature-dependent relaxation preserves and further reduces triangle size towards the crease where the curvature is highest.

Overall, both methods behave as expected. Uniform relaxation performs better than feature-dependent relaxation in areas with strong stretching but low curvature, and feature-dependent relaxation performs better in areas with high curvature but less stretching.

There are three main differences between the two methods. First of all, uniform relaxation computes energy based on local deformation information, whereas feature-dependent relaxation computes it based on Gauß curvature. Second, the uniform method moves particles away from high energy, whereas the feature-dependent method moves them towards high energy. Finally, uniform relaxation bases its energy function on information from a single point, whereas feature-dependent relaxation bases it on information from the star set of each point.

In Figure 5.13, I present the results obtained by applying both methods to the Rayleigh-Taylor Instability dataset. As there is a lot of turbulence in this dataset, it is difficult to compare it side by side in static images. On the left of each image, there is a close-up of a region in which the differences are easily seen. Feature-dependent relaxation is able to preserve the peak in its full size, wherease it shrinks under uniform relaxation. Uniform relaxation, however, provides a more balanced triangulation on the side of the peak that descends towards the viewer. There are more examples like this throughout the surface, e.g. there is a double peak at about a quarter from the left of the front border. Feature-dependent relaxation preserves the depth of the minimum between the peak, whereas with uniform relaxation it becomes more shallow. As the Rayleigh-Taylor Instability produces high curvature almost everywhere on the surface, feature-dependent relaxation is much better suited for this particular example.

Overall, each technique has its own strengths and weaknesses as they were both developed with a specific goal in mind. One question remains, however: how many relaxation steps are adequate? Throughout this chapter, almost all images and statistics were generated using 50 iterations. However, 10 iterations can produce similar results to those obtained after 50 steps, as seen in Figures 5.10 and 5.11. It is possible to define termination critera such as minimum required change between iterations. This choice depends on the dataset. For the Dambreak dataset, the changes shown in Figure 5.12c have spread significantly starting from the surface border between 10 iterations and 50 iterations (not shown).

5.7 Conclusion

The goal of this chapter is to support simulation through the development of a time-surface extraction technique that performs adaptive particle relaxation to keep the surface well-sampled and balanced.

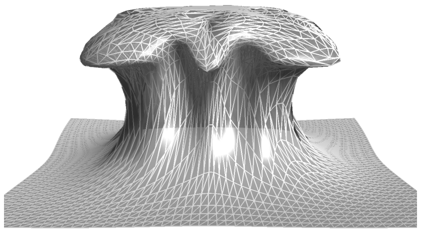
I have presented two different particle relaxation techniques. The first technique performs local energy optimisation based on the metric tensor, such that particles move away from regions with high energy (i.e. high particle density) and towards regions with low energy (i.e. low particle density). The metric tensor is computed from discrete tangents on each particle and it provides a measure for local surface deformation. The second technique performs local curvature optimisation based on discrete Gauß curvature at a particle. Particles move away from low-curvature regions towards high-curvature regions.

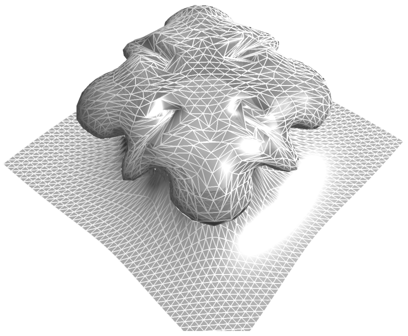
Both approaches operate on a 2D scalar field in parameter space. To transfer point movement from 2D parameter space to 3D Euclidean space, it is necessary to interpolate point positions on the surface. I utilise Coons patches as a higher-order surface representation to enable faithful reconstruction.

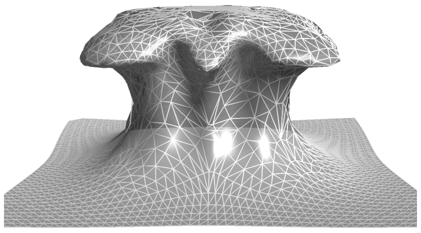
Furthermore, I optimise the mesh in order to achieve accurate surface representations with good mesh quality. Checking the Delaunay criterion for neighbouring triangles to avoid degenerated triangles is the basis for this optimisation. Moreover, I employ discrete mean curvature as a restriction for the Delaunay criterion. This serves to preserve convex and concave surface features.

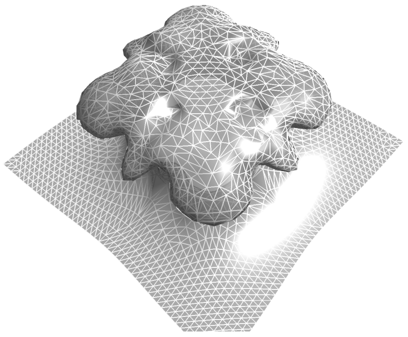
As seen in Section 5.6, both relaxation techniques result in a better surface sampling. Unsurprisingly, uniform relaxation is better at providing a more uniform surface sampling, whereas feature-dependent relaxation is better at preserving surface shape in regions of high curvature.

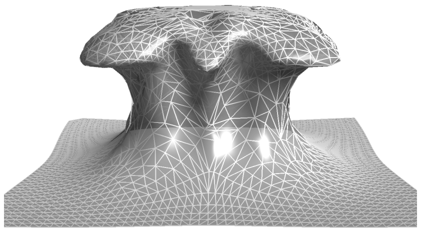
Both relaxation techniques work on a copy of the previous relaxation iteration (or the freshly advected surface). Furthermore, they work locally in the neighbourhood of a point. This means that almost all computations can be executed in parallel. Therefore, particle relaxation could benefit greatly from parallel execution on GPUs, and from executing in a high-performance computing environment.


(a) 0 relaxation steps, side view.


(b) 0 relaxation steps, top view.


(c) 10 relaxation steps, side view.


(d) 10 relaxation steps, top view.


(e) 50 relaxation steps, side view.

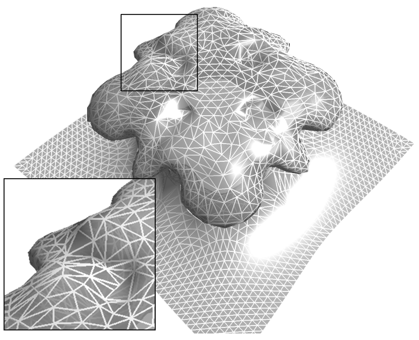
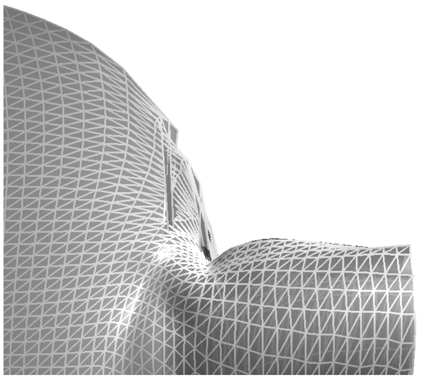
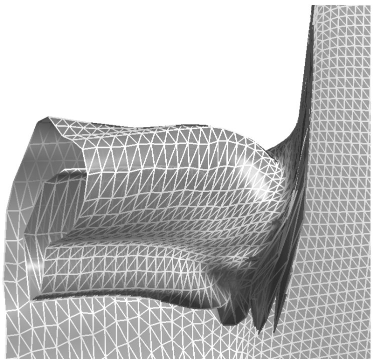
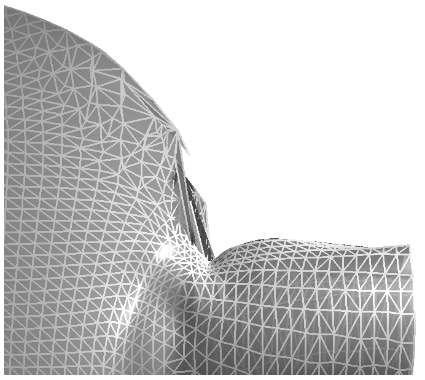
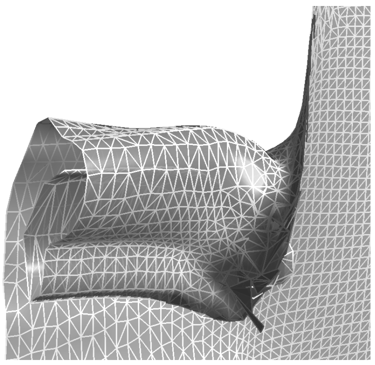

(f) 50 relaxation steps, top view.

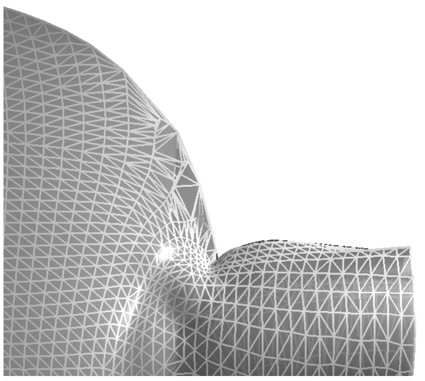
Figure 5.11: Jet during feature-dependent relaxation for a surface of resolution 50x50.


(a) Front view, no relaxation.


(b) Back view, no relaxation.


(c) Front view, uniform relaxation.


(d) Back view, uniform relaxation.


(e) Front view, feature-dependent relaxation.

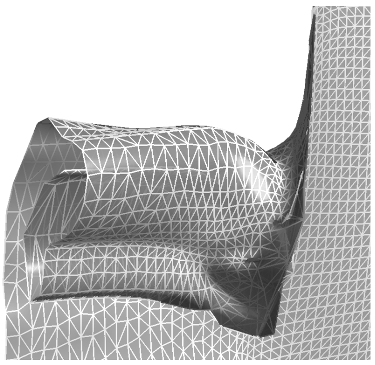
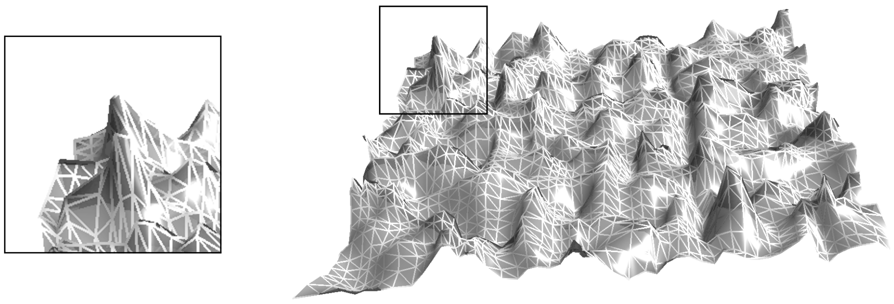
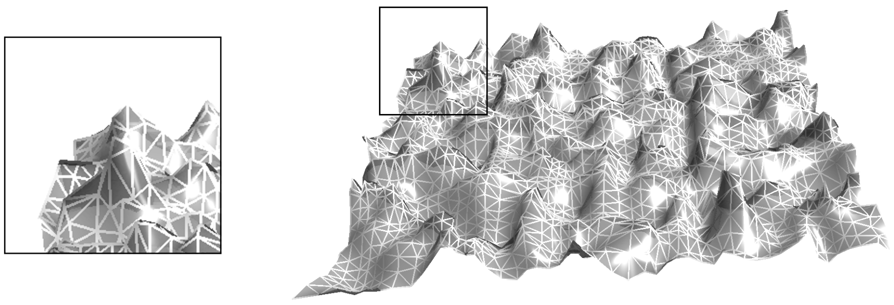

(f) Back view, feature-dependent relaxation.

Figure 5.12: Relaxation of the Dambreak dataset for a surface of resolution 50x50.


(a) No relaxation.


(b) Uniform relaxation.

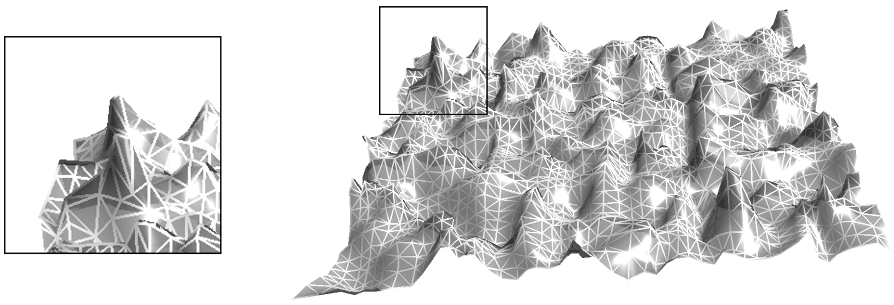

(c) Feature-dependent relaxation.

Figure 5.13: Relaxation of the Rayleigh-Taylor Instability dataset for a surface of resolution 50x50.

Chapter 6

Conclusion

In this thesis, I have presented a range of methods supporting the virtual prototyping workflow of industrial surface design (Figure 6.1). From a masterpiece to a final product, several iterations of refinement are required to improve the prototype. A virtual prototype is built consisting of analytic or discrete surface patches, e.g. Bézier patches. It can be modified interactively by deforming along normal vectors, or one can optimise geometric properties automatically. Alternatively, one can run simulations to identify issues of the model or its interaction with the environment, and one can then reiterate through the virtual design cycle to solve these issues.

After sufficiently many iterations of the virtual design cycle, a physical prototype can be constructed and tested. Depending on the outcome of testing, one can either go back to the virtual design cycle to solve potential real-world issues. Finally, if all tests yield satisfactory results, the final product can be built.

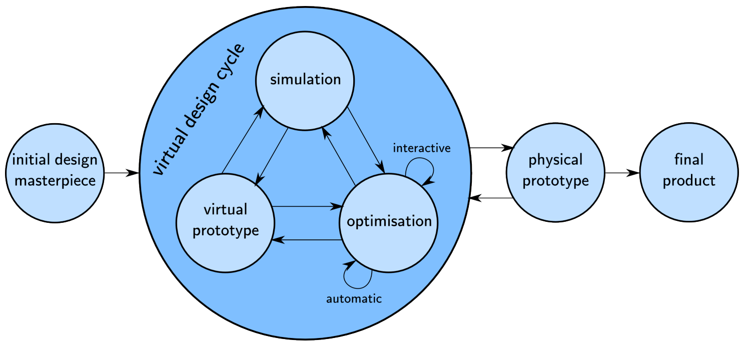


Figure 6.1: Virtual prototyping workflow: from a masterpiece to the final product.

In Chapter 3, the main focus lies on automated surface optimisation. I have presented how to preserve total Gauß curvature during an automatable deformation process that takes place in normal direction. As I have proven in this chapter, infinitesimal changes in normal direction do not affect total Gauß curvature. Minimal total Gauß curvature leads to minimal bending energy and a reduction of material use. Therefore, overall production cost is reduced. This surface optimisation procedure can be employed at the end of the virtual design cycle, after larger changes to the model have already occurred.

In Chapter 4, the main focus lies on interactive surface optimisation. I have provided discrete normals as a tool for interactive surface deformation. Furthermore, I have compared different stategies for discrete normal estimation for both curves and surfaces. The resulting normals provide an intuitive and effective means of manipulation for a virtual prototype consisting of surface patches. Depending on the specific surface model, different types of normals may be the best choice. The industrial surface designer has to judge which set of normals best fits his or her purposes.

In Chapter 5, the main focus is two-fold. Both contributions tie into simulation, but they can also be applied to meshes directly. I have improved surface discretisation through two different relaxation procedures to achieve a better-suiting sampling of the surface. Moreover, I have improved mesh quality in order to better represent and preserve surface shape.

One of the focus areas is the interface between a virtual prototype and simulation, as this step requires discretisation. I have presented two different strategies for surface discretisation, which both use a relaxation procedure to adapt the surface discretisation through redistribution of particles. The first strategy creates a more uniform distribution by using a deformation tensor-based energy function which counterbalances the local distortion. It is well-suited for models which possess features of similar sizes, as the discretisation provides a spatially uniform sampling of the surface. The second strategy creates a non-uniform distribution that has finer sampling for small-scale features, and coarser sampling for large-scale features. This is achieved using curvature-based relaxation which effectively detects surface features. It is well-suited for models which have varying feature sizes, as the sampling is denser in regions with small features than in regions with large features.

The other focus area of this chapter is automated mesh optimisation. Due to the change in sample positions through advection as well as through relaxation, mesh quality suffers while the discretisation becomes better. I have optimised mesh quality by using a combination of the Delaunay criterion and discrete mean curvature as a criterion to steer retriangulation. The resulting meshes have much larger average minimal angles than unrelaxed surfaces, as shown in Section 5.6.2.

Overall, I have contributed the following techniques to support the virtual prototyping workflow:

a discretisation method that can either be applied to the surface model of a virtual prototype, or to integral surfaces during simulation. On the virtual prototype, it supports optimisation of an initial sampling prior to conducting simulations on the model. During the simulation process, it counteracts local deformation of integral surfaces, and it provides a better sampling of differently sized features of the surface.

curvature-aware mesh optimisation which can be used to optimise triangulations of the virtual prototype, or of integral surfaces. In either case, it assists engineers in producing faithful samplings of a surface, which do not sacrifice convex or concave surface features, while keeping the required number of mesh points low.

an interactive modification tool in the form of discrete surface normals for control structure. It can be used to modify the initial design, or to optimise the surface model during the virtual design cycle.

automatic optimisation of total curvature, which can be employed on the virtual prototype initially during the design phase, or before constructing physical prototypes.

Finally, to round off my work, I will conclude this thesis with an exemplary virtual prototyping workflow which incorporates all of these contributions.

6.1 Examplary Workflow

Automobile industry relies heavily on the use of CAGD methods. Some of the geometric foundations of this thesis were, in fact, developed in an automotive engineering context.

When the coachwork of a car is developed, the process starts with a visual idea, e.g. a sketch on paper. This sketch is then constructed as a virtual prototype in a CAGD environment. Next, industrial designers can interactively modify and refine the virtual prototype within this CAGD environment, or possibly even a virtual reality environment. For this modification step, efficient tools for interaction are vital. Discrete normals, as introduced in Chapter 4, are an easy and efficient tool to deform the surface model. They work particularly well if the model is given as Bézier surface patches, as the user can restrict interaction to a small number of control points instead of a large number of individual surface points. After manual modifications are made, the industrial designer can apply automatic optimisation with respect to total curvature in order to minimise material expenses at an early stage, as presented in Chapter 3.

Once the designer is satisfied with the surface model of the virtual prototype, he or she can run simulations. For this purpose, the prototype surface has to be discretised. Initially, this can be achieved by using an equidistant sampling in parameter space. However, as demonstrated in Figure 4.14, equidistant parameters do not always provide a good sampling of a surface. Therefore, it makes sense to optimise the initial sampling using one of the relaxation methods presented in Chapter 5 (and using the vertices as particles). On a single surface patch, uniform relaxation is a suitable choice as large curvature changes are unlikely to occur. If the surface model is constructed from a collection of Bézier patches, large curvature changes within a patch are still unlikely, however, they can occur across surface patch boundaries. When relaxing such a surface, points can travel across patch boundaries after discretisation. In this case, either technique (uniform relaxation or feature-dependent relaxation) could be a good choice, depending on the surface model. For example, the side mirror of a can like the one displayed in Figure 6.2 has much finer features than the roof of a car. Therefore, it makes sense to have a higher resolution sampling of the mirror than of the roof, so feature-dependent relaxation would be a good choice.

When the virtual prototype has been discretised, simulations can be conducted. Wind tunnel simulation is a typical simulation used in automotive engineering. An object, such as a vehicle, is placed in a virtual wind tunnel. The flow behaviour of the air is represented by a three-dimensional vector field. It serves to evaluate the aerodynamic resistance of the vehicle (e.g. to reduce gas consumption), and to identify potential stability issues (e.g. to reduce potential breakage). Turbulence in the air flow is one of the features that engineers are interested in. To aid flow analysis, the flow field can be visualised using integral surfaces. These integral surfaces can be optimised using the relaxation techniques presented in Chapter 5. Figure 6.2 shows an example of visualisation by computing and rendering integral surfaces (without relaxation). As one can see in this rendering, there is a lot of turbulence behind the side mirror. If the engineer is not satisfied with the aerodynamic properties of the surface model, he or she can recycle to the optimisation stage to modify it. After a discretisation of the modified surface model, the simulation can be run again.

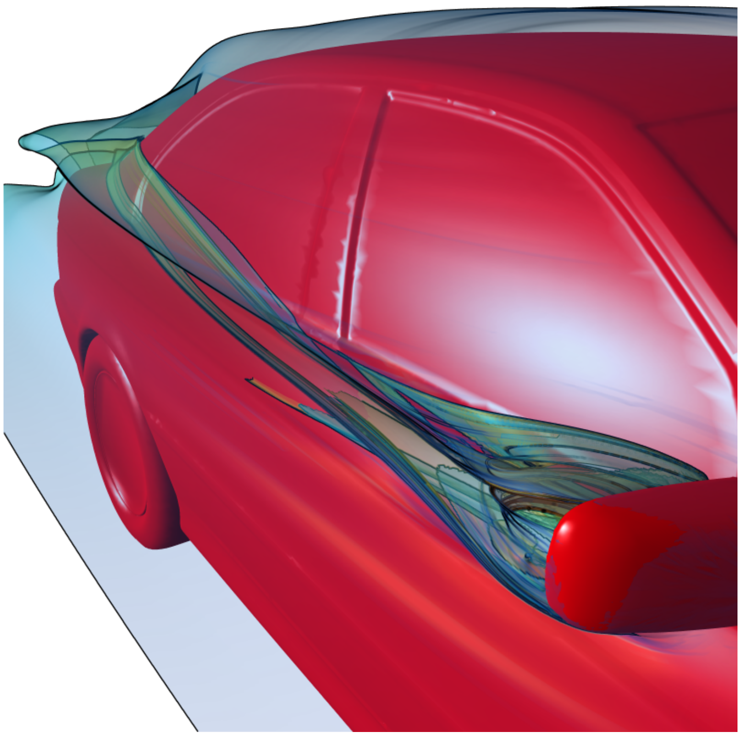


Figure 6.2: Flow visualisation of simulated flow around the surface model of a car. This image is included with kind permission from Mathias Hummel. It was generated using illustrative rendering as presented in [HGH+10].

When sufficiently many iterations of modifying and analysing the virtual prototype have been conducted, the results meet all criteria that the engineer has defined. At this point, the virtual prototype can be built as a physical prototype for further testing, and eventually, it can go into production.

Appendix A

Index of Notations

Lower Case Greek Alphabet

φ, ϑ	angles
κ	curvature
κ_1, κ_2	minimal/maximal curvature
λ^i	choice operator (chooses i^{th} element of a vector
τ	torsion
ω	weight

Upper Case Greek Alphabet

Γ_{ijk}	Christoffel symbols of the first kind
Γ_{ij}^{k}	Christoffel symbols of the second kind
Ω	parameter space

Lower Case Standard Letters

$d(P,Q)$	distance between P,Q
$d_H(P,Q)$	Hausdorff distance between P,Q
f,g,h	function
g_{11}, g_{12}, g_{22}	first fundamental form
g_{jk}	metric tensor
g^{lm}	conjugate metric tensor
h_{11}, h_{12}, h_{22}	second fundamental form
i,j,k,l,m	indices
n	$n \in \mathbb{N}$
r	radius
s	arc length
t	time parameter
u,v,w	spatial parameters
x,y,z	3D coordinates

Upper Case Standard Letters

C^k	k-times continuously differentiable
E	energy kernel
H	mean curvature
K	Gauß curvature
M	metric tensor
$N(\mathbf{x})$	neighbourhood of $\mathbf{x}$
T	(deformation) tensor
S	submanifold
$T_{\mathbf{p}}\mathbf{M}$	collection of all tangent spaces at all points $\mathbf{p}$ of n-dimensional manifold $\mathbf{M}$
$T\mathbf{M}$	tangent bundle ($2n$-dimensional manifold)
U, U_i, U_a	subspace
X, Y	set

Lower Case Altered Letters

$\mathbf{a}, \mathbf{c}$	edges
$\mathbf{b}$	binormal vector
$\mathbf{e}$	eigen vector
$\mathbf{f}$	force vector
$\mathbf{m}$	main normal vector
$\mathbf{n}$	normal vector
$\mathbf{p}, \mathbf{q}$	points
$\mathbf{t}, \mathbf{t}_u, \mathbf{t}_w, \mathbf{t}_{uw}$	tangent vector
$\mathbf{v}$	velocity vector
$\mathbf{x}$	vector
$\dot{\mathbf{x}}, \ddot{\mathbf{x}}, \dddot{\mathbf{x}}$	derivatives w.r.t. t
$\mathbf{x}', \mathbf{x}'', \mathbf{x}'''$	derivatives w.r.t. a different variable
$\mathbf{x}_u, \mathbf{x}_w, \mathbf{x}_{uw}$	directional derivatives

Upper Case Altered Letters

$\mathscr{A}(\mathbf{x})$	adjacency set of $\mathbf{x}$
$\mathbf{C}$	curve
$\mathbb{E}^3$	Euclidean space
$\mathbf{M}$	manifold
$\mathbb{N}$	natural numbers
$\mathbb{Z}$	integers
$\mathbb{R}$	real numbers
$\mathbf{S}$	surface
$\mathbf{T}$	triangle
$\mathscr{T}$	topology
$\mathscr{T}_X$	topology on X

Other

$\|\|\mathbf{x}\|\|$	Euclidean norm
∇f	gradient of f
∂	partial derivative

Appendix B

Curriculum Vitae

Andreas Silvan Berres

berres@cs.uni-kl.de

Education

Ph.D., Computer Science 2011 – 2015
University of Kaiserslautern
Kaiserslautern, Germany
Topic: *Discrete Geometric Methods for Surface Deformation and Visualisation*

M.Sc., Computer Science 2009 – 2011
University of Kaiserslautern
Kaiserslautern, Germany
Specialisation: *Computer Graphics and Visualisation*
Minor: *Biology*

B.Sc., Computer Science 2005 – 2009
University of Kaiserslautern
Kaiserslautern, Germany
Specialisation: *Human Computer Interaction*
Minor: *Mathematics*

Abitur 2005
Nicolaus Cusanus Gymnasium
Bonn, Germany
Advanced Courses: *English (bilingual branch), Mathematics*

Honors and Awards

FIT Award for Extraordinary Dedication, 2010
awarded by Förderverein Informatik der TU Kaiserslautern (FIT)

Research Visits

Los Alamos National Lab, Los Alamos, NM, USA, 3 months, 2015
Research with Dr. James Ahrens

German Aerospace, 3.5 months, 2014
Research with Dr. Andreas Gerndt and Dr. Ingrid Hotz

UC Davis, CA, USA, 2 x 3 months, 2013 & 2014
Research with Prof. Kenneth I. Joy and Dr. Harald Obermaier

University of Connecticut, CT, USA, 3 months, 2012
Research with Prof. Thomas J. Peters

University of Edinburgh, Scotland, UK, 4 months, 2008
Semester abroad during which I attended lectures.
Focus: *Genetic Algorithms, Logic Programming, Cognitive Science, and Scottish Ethnology*

Bibliography

[ADPS95] L-E. Andersson, S. M. Dorney, T. J. Peters, and N. F. Stewart. Polyhedral perturbations that preserve topological form, 1995.

[AOJ13] Alexy Agranovsky, Harald Obermaier, and Kenneth I Joy. A framework for the visualization of finite-time continuum mechanics effects in time-varying flow. In *Advances in Visual Computing*, pages 349–360. Springer, 2013.

[AT89] M. P. Allen and D. J. Tildesley. *Computer Simulation of Liquids*. Clarendon Press, New York, NY, USA, 1989.

[BB06] Michael Bender and Manfred Brill. *Computergrafik: ein anwendungsorientiertes Lehrbuch*. Hanser Verlag, 2 edition, 2006.

[BFTW09] Kai Burger, Florian Ferstl, Holger Theisel, and Rüdiger Westermann. Interactive streak surface visualization on the gpu. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):1259–1266, 2009.

[BH96] Frank Bossen and Paul S. Heckbert. A pliant method for anisotropic mesh generation. In *INTERNATIONAL MESHING ROUNDTABLE*, pages 63–74, 1996.

[BH15] Andreas S. Berres and Hans Hagen. Feature-preserving discrete normal fields for bézier polygons. 2015. Submission in progress.

[BHH15] Andreas Berres, Hans Hagen, and Stefanie Hahmann. Deformations preserving Gauß curvature. In Janine Bennett, Fabien Vivodtzev, and Valerio Pascucci, editors, *Topological and Statistical Methods for Complex Data*, Mathematics and Visualization, pages 143–163. Springer Berlin Heidelberg, 2015.

[BIA12] M Buck, O Iliev, and H Andrä. *Multiscale finite element coarse spaces for the analysis of linear elastic composites*. Fraunhofer-Institut für Techno-und Wirtschaftsmathematik, Fraunhofer (ITWM), 2012.

[Boi95] Eric Boix. *Approximation linéaire des surfaces de R3 et applications*. PhD thesis, École Polytechnique, France, 1995.

[BOJH15] Andreas Berres, Harald Obermaier, Ken Joy, and Hans Hagen. Adaptive particle relaxation for time surfaces. In *Pacific Visualization Symposium (PacificVis), 2015 IEEE*, pages 147–151, April 2015.

[Bou97] Jacques Bousquet. *Détection et élimination d'irrégularités sur les surface manipulées en C.A.O.* PhD thesis, Institut National Polytechnique de Grenoble, France, 1997.

[BWFS11] Jürgen Becker, Christian Wieser, Stephan Fell, and Konrad Steiner. A multi-scale approach to material modeling of fuel cell diffusion media. *International Journal of Heat and Mass Transfer*, 54(7):1360–1368, 2011.

[CBP05] Simon Clavet, Philippe Beaudoin, and Pierre Poulin. Particle-based viscoelastic fluid simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '05, pages 219–228, New York, NY, USA, 2005. ACM.

[CCG$^+$12] David Camp, Hank Childs, Christoph Garth, David Pugmire, and Kenneth I Joy. Parallel stream surface computation for large data sets. In *Large Data Analysis and Visualization (LDAV), 2012 IEEE Symposium on*, pages 39–47. IEEE, 2012.

[CKP$^+$13] David Camp, Hari Krishnan, David Pugmire, Christoph Garth, Ian Johnson, E Bethel, Kenneth I Joy, and Hank Childs. Gpu acceleration of particle advection workloads in a parallel, distributed memory setting. In *Proceedings of the 13th Eurographics Symposium on Parallel Graphics and Visualization*, pages 1–8. Eurographics Association, 2013.

[CMT01] David Coeurjolly, Serge Miguet, and Laure Tougne. Discrete curvature based on osculating circle estimation. In *Visual Form 2001*, pages 303–312. Springer, 2001.

[DBVKOS00] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. *Computational geometry*. Springer Berlin Heidelberg, 2000.

[DHKL01] Nira Dyn, Kai Hormann, Sun-Jeong Kim, and David Levin. Optimizing 3d triangulations using discrete curvature analysis. *Mathematical methods for curves and surfaces*, 28(5):135–146, 2001.

[Do 76] Manfred P. Do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, 1976.

[Efi57] N. Efimov. *Flaechenverbiegungen im Grossen (in German)*. Akademie-Verlag, Berlin, 1957.

[EH10] Herbert Edelsbrunner and John Harer. *Computational topology: an introduction*. American Mathematical Soc., 2010.

[EP09] Michael Eigensatz and Mark Pauly. Positional, metric, and curvature control for constraint-based surface deformation. In *Computer Graphics Forum*, volume 28.2, pages 551–558. Wiley Online Library, 2009.

[ESP08] Michael Eigensatz, Robert W. Sumner, and Mark Pauly. Curvature-domain shape processing. *Computer Graphics Forum*, 27(2):241–250, 2008.

[Far02a] G.E. Farin. *Curves and surfaces for CAGD: a practical guide*. Morgan Kaufmann Pub, 2002.

[Far02b] Gerald E. Farin. *Curves and Surfaces for CAGD: A Practical Guide*. Computer graphics and geometric modeling. Morgan Kaufmann, 2002.

[GGH$^+$89] W. Gellert, S. Gottwald, M. Hellwich, H. Kästner, and H. Künstner. *VNR Concise Encyclopedia of Mathematics*. 2 edition, 1989.

[GGRZ06] Eitan Grinspun, Yotam Gingold, Jason Reisman, and Denis Zorin. Computing discrete shape operators on general meshes. In *Computer Graphics Forum*, volume 25(3), pages 547–556. Wiley Online Library, 2006.

[GKTJ08] Christoph Garth, Hari Krishnan, Xavier Tricoche, and KI Joy. Generation of accurate integral surfaces in time-dependent vector fields. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1404–1411, 2008.

[GU02] J. Gravesen and M. Ungstrup. Constructing invariant fairness measures for surfaces. *Advances in Computational Mathematics*, 17(1-2):67–88, 2002.

[Hag09] Hans Hagen. Algorithmic geometry lecture notes, 2009.

[HGH$^+$10] Mathias Hummel, Christoph Garth, Bernd Hamann, Hans Hagen, Kenneth Joy, et al. Iris: Illustrative rendering for integral surfaces. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1319–1328, 2010.

[HH96] Stefanie Hahmann and Hans Hagen. Numerical aspects of stability investigations on surfaces. In *Proceedings of the 6th IMA Conference on the Mathematics of Surfaces*, pages 291–308, New York, NY, USA, 1996. Clarendon Press.

[HH98] H. Hagen and S. Hahmann. Stability conditions for free form surfaces. In *Proceedings of the Computer Graphics International 1998*, CGI '98, pages 41–48, Washington, DC, USA, 1998. IEEE Computer Society.

[Hop96] Hugues Hoppe. Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108. ACM, 1996.

[Hul92] Jeffrey PM Hultquist. Constructing stream surfaces in steady 3d vector fields. In *Proceedings of the 3rd conference on Visualization'92*, pages 171–178. IEEE Computer Society Press, 1992.

[HY00] G Haller and G Yuan. Lagrangian coherent structures and mixing in two-dimensional turbulence. *Physica D: Nonlinear Phenomena*, 147(3):352–370, 2000.

[IKS95] Ivanka Ivanova-Karatopraklieva and I Kh Sabitov. Bending of surfaces. Part II. *Journal of Mathematical Sciences*, 74(3):997–1043, 1995.

[JCNH10] Xiangmin Jiao, Andrew Colombi, Xinlai Ni, and John Hart. Anisotropic mesh adaptation for evolving triangulated surfaces. *Engineering with Computers*, 26(4):363–376, 2010.

[JMM$^+$08] K. E. Jordan, L. Miller, E. L. F. Moore, T. J. Peters, and A. C. Russell. Modeling time and topology for animation and visualization. *Theoretical Computer Science*, 405(1 - 2):41 – 49, 2008.

[JS07] P. Joshi and C. H. Sequin. Energy minimizers for curvature-based surface functionals. *CAD Conference, Waikiki, Hawaii*, pages 607–617, June 2007.

[KESW09] Gordon L Kindlmann, Raúl San José Estépar, Stephen M Smith, and C-F Westin. Sampling and visualizing creases with scale-space particles. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):1415–1424, 2009.

[KGJ09] Hari Krishnan, Christoph Garth, and Kenneth I Joy. Time and streak surfaces for flow visualization in large time-varying data sets. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):1267–1274, 2009.

[KKL02] Sun-Jeong Kim, Chang-Hun Kim, and David Levin. Surface simplification using a discrete curvature norm. *Computers & Graphics*, 26(5):657–663, 2002.

[KL08] Bertrand Kerautret and Jacques-Olivier Lachaud. Robust estimation of curvature along digital contours with global optimization. In *Discrete Geometry for Computer Imagery*, pages 334–345. Springer, 2008.

[KLN08] Bertrand Kerautret, J-O Lachaud, and Benoît Naegel. Comparison of discrete curvature estimators and application to corner detection. In *Advances in Visual Computing*, pages 710–719. Springer, 2008.

[Kre68] E. Kreyszig. *Introduction to differential geometry and Riemannian geometry*, volume 114. University of Toronto Press, 1968.

[LBS05] Torsten Langer, Alexander Belyaev, and Hans-Peter Seidel. Analysis and design of discrete normals and curvatures, 2005.

[Lee11] John M. Lee. *Introduction to topological manifolds*, volume 202 of *Graduate Texts in Mathematics*. Springer Verlag, 2011.

[LHK$^+$08] Heng Li, Ting Huang, C.W. Kong, H.L. Guo, Andrew Baldwin, Neo Chan, and Johnny Wong. Integrating design and construction through virtual prototyping. *Automation in Construction*, 17(8):915–922, 2008.

[LLV05] David Lesage, Jean-Claude Leon, and Philippe Veron. Discrete curvature approximations and segmentation of polyhedral surfaces. *International Journal of Shape Modeling*, 11(02):217–252, 2005.

[LP82] C Lin and MJ Perry. Shape description using surface triangulation. In *Proceedings of the Workshop on Computer Vision: Representation and Control*, pages 38–43, 1982.

[LP12] Ji Li and Thomas J. Peters. Isotopy convergence theorem, 2012.

[Max99] Nelson Max. Weights for computing vertex normals from facet normals. *Journal of Graphics Tools*, 4(2):1–6, 1999.

[McL01] Peter McLeod. *'Low-End' Virtual Prototyping*. PRIME Faraday Technology Watch. PRIME Faraday Partnership, Loughborough University, Loughborough, Leics LE11 3TU, UK, nov 2001.

[MDSB03] Mark Meyer, Mathieu Desbrun, Peter Schröder, and AlanH. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In Hans-Christian Hege and Konrad Polthier, editors, *Visualization and Mathematics III*, Mathematics and Visualization, pages 35–57. Springer Berlin Heidelberg, 2003.

[Mey08] Miriah Meyer. *Dynamic Particle Systems for Adaptive Sampling of Implicit Surfaces*. PhD thesis, School of Computing, University of Utah, 2008.

[Mez07] Abdelhamid Meziani. Infinitesimal bendings of high orders for homogeneous surfaces with positive curvature and a flat point. *Journal of Differential Equations*, 239(1):16–37, 2007.

[MH36] WE Morrell and JH Hildebrand. The distribution of molecules in a model liquid. *The Journal of Chemical Physics*, 4(3):224–227, 1936.

[MLP$^+$10] Tony McLoughlin, Robert S Laramee, Ronald Peikert, Frits H Post, and Min Chen. Over two decades of integration-based, geometric flow visualization. In *Computer Graphics Forum*, volume 29.6, pages 1807–1829. Wiley Online Library, 2010.

[MNG04] Niloy J Mitra, An Nguyen, and Leonidas Guibas. Estimating surface normals in noisy point cloud data. *International Journal of Computational Geometry & Applications*, 14(04n05):261–276, 2004.

[MWK$^+$08] Miriah Meyer, Ross Whitaker, Robert M Kirby, Christian Ledergerber, and Hanspeter Pfister. Particle-based sampling and meshing of surfaces in multimaterial volumes. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1539–1546, 2008.

[NRDR05] Diego Nehab, Szymon Rusinkiewicz, James Davis, and Ravi Ramamoorthi. Efficiently combining positions and normals for precise 3D geometry. *ACM transactions on graphics (TOG)*, 24(3):536–543, 2005.

[OHBKH09] Harald Obermaier, Martin Hering-Bertram, Jörg Kuhnert, and Hans Hagen. Volume deformations in grid-less flow simulations. In *Computer Graphics Forum*, volume 28.3, pages 879–886. Wiley Online Library, 2009.

[OHBKH11] Harald Obermaier, Martin Hering-Bertram, Jörg Kuhnert, and Hans Hagen. Generation of adaptive streak surfaces using moving least squares. *Dagstuhl Follow-Ups*, 2, 2011.

[OJ12] Harald Obermaier and Kenneth I Joy. Derived metric tensors for flow surface visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 18.12:2149–2158, 2012.

[Owe98] Steven J Owen. A survey of unstructured mesh generation technology. In *IMR*, pages 239–267, 1998.

[PM88] John G Proakis and Dimitris G Manolakis. *Introduction to digital signal processing*. Prentice Hall Professional Technical Reference, 1988.

[PMKW78] Piotr Pierański, Jerzy Małecki, Wojciech Kuczyński, and Krzysztof Wojciechowski. A hard-disc system, an experimental model. *Philosophical Magazine A*, 37(1):107–115, 1978.

[PP93] Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental mathematics*, 2(1):15–36, 1993.

[ST92] Richard Szeliski and David Tonnesen. Surface modeling with oriented particle systems. *Computer Graphics (SIGGRAPH'92)*, 26.2:185–194, July 1992.

[SWS09] Dominic Schneider, Alexander Wiebel, and Gerik Scheuermann. Smooth stream surfaces of fourth order precision. In *Computer Graphics Forum*, volume 28.3, pages 871–878. Wiley Online Library, 2009.

[SYI$^+$97] Kenji Shimada, Atsushi Yamada, Takayuki Itoh, et al. Anisotropic triangular meshing of parametric surfaces via close packing of ellipsoidal bubbles. In *6th International Meshing Roundtable*, pages 375–390, 1997.

[TDR99] Pierre Tellier and Isabelle Debled-Rennesson. 3d discrete normal vectors. In *Discrete Geometry for Computer Imagery*, pages 447–458. Springer, 1999.

[TW97] Grit Thürmer and Charles A Wüthrich. Normal computation for discrete surfaces in 3d space. In *Computer graphics forum*, volume 16(3), pages C15–C26. Wiley Online Library, 1997.

[TW98] Grit Thürmer and CA Wüthrich. Varying neighbourhood parameters for the computation of normals on surfaces in discrete space. In *Computer Graphics International, 1998. Proceedings*, pages 616–625. IEEE, 1998.

[VC11] Ljubica S. Velimirovic and Marija S. Ciric. On the total mean curvature of piecewise smooth surfaces under infinitesimal bending. *Applied Mathematics Letters*, 24(9):1515 – 1519, 2011.

[VRZ11] Ljubica S. Velimirovic, Svetozar R. Rancic, and Milan Lj. Zlatanovic. Visualization of infinitesimal bending of curves. In Walter Gautschi, Giuseppe Mastroianni, and Themistocles M. Rassias, editors, *Approximation and Computation*, volume 42 of *Springer Optimization and Its Applications*, pages 469–480. Springer New York, 2011.

[Wik12] Wikipedia. http://en.wikipedia.org, 2012.

[Xu06] Guoliang Xu. Convergence analysis of a discretization scheme for Gaussian curvature over triangular surfaces. *Computer Aided Geometric Design*, 23(2):193–207, 2006.

[ZGW$^+$13] Zichun Zhong, Xiaohu Guo, Wenping Wang, Bruno Lévy, Feng Sun, Yang Liu, Weihua Mao, et al. Particle-based anisotropic surface meshing. *ACM Trans. Graph.*, 32(4):99, 2013.