Processing of Meshes and Geometry for Visualization Applications

Von der Fakultät für Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

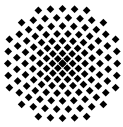
Vorgelegt von

Katrin Bidmon

aus Geislingen an der Steige

Hauptberichter:	Prof. Dr. Thomas Ertl
Mitberichter:	Prof. Dr. Hans Hagen

Tag der mündlichen Prüfung: 6. Oktober 2010



Visualisierungsinstitut der Universität Stuttgart

2010

Contents

List of Abbreviations

$\langle .,. \rangle$	the scalar product
I_p	the first fundamental form
II_p	the second fundamental form
C^n	the n times continuously differentiable functions
$\dot{c}$	derivative with respect to parameter t
c'	derivative with respect to arc length parameter s
f'	derivative with respect to the function parameter
$f^{(n)}$	n-th derivative with respect to the function parameter
iff	if and only if
lin(...)	the linear hull
T_pM	the tangent space
AFM	advancing front method
CAE	computer-aided engineering
DTI	diffusion tensor imaging
GPU	graphic processing unit
FE	finite element
FEA	finite element analysis
MLS	moving least-squares
MR	magnetic resonance
MRI	magnetic resonance imaging
ODE	ordinary differential equation

Abstract

The fast increase of computational power not only enables the simulation of complex non-linear and highly dynamic processes but also allows for further increase of the problem sizes and makes parameter studies with numerous simulation runs affordable. One of the often underrated consequences of this development is the resulting rampant amount of simulation data that has to be processed, analysed and evaluated accordingly. Therefore, the development of powerful and capable analysis tools likewise gains in importance with visualization playing an increasingly crucial role.

The visual conditioning of data – both in simulation pre- and post-processing – provides intuitive and fast insight. Hence, appropriate visualizations have to be developed and, where required, tailored to the specific needs of the particular application. As in visualization applications the principal purpose is not a visually pleasing appearance itself – although marvellous visual quality of course is preferable – but to provide an ideal blend of data compensation and emphasis on relevant features in order to enable and support intuitive data handling and analysis.

In many application fields, geometry plays a crucial role in analysis. The major contributions of this work are on the geometric aspects of visualization methods in the application fields of virtual prototyping in car industry on one hand and molecular dynamics on the other hand. In both, the challenge is to comply with application needs while satisfying the required correctness of the shape, geometry and topology in order to ensure reliable analysis support, while providing superior visual quality in interactive methods, elaborating the data characteristics without concealing relevant features. But still the focus with respect to geometry is different in both application fields.

On one hand, as in the area of car prototyping, reliable geometric models are of superior importance for both robust simulation set-ups and trustable results, since the evaluation of the geometric properties of the model is the principal purpose of simulation. The simulations in this field are usually based on finite element (FE) methods, thus the visualization is mesh-based accordingly. In this thesis new methods for processing, customization and (re-)construction of geometry and geometric characteristics are presented, tailored to the specific needs of automotive pre-processing.

On the other hand, as in the application field of molecular dynamics, the geometric shape of the simulation entities often is not relevant but dictates

the simulation constraints and, thus, still plays an essential role in analysis tasks. Therefore, the work presented in this field emphasises the power of geometric concepts as essential foundation for data structuring and intuitive evaluation during simulation data analysis. Since the molecules themselves do not have an intrinsic shape, geometric molecular representations always entail abstraction up to a certain extent. This fact, in turn, can be exploited to create semantically expressive molecular visualizations based on very different intrinsic and geometric properties of the data.

Being developed in close collaboration with scientists in the dedicated application fields, the methods presented in this thesis found their way into recent research in the case of molecular dynamics as well as into commercial application tools in the case of the finite element analysis methods.

Zusammenfassung

Der inzwischen sprunghafte Anstieg an verfügbarer Rechenleistung ermöglicht nicht nur die Simulation komplexer nicht-linearer und hochdynamischer Prozesse an sich, sondern erlaubt auch den weiteren Anstieg der Problemgrößen und macht Parameterstudien mit zahlreichen Simulationsläufen handhabbar. Eine der dabei häufig unterschätzten Folgen ist die daraus resultierende starke Zunahme an Simulationsergebnissen, welche anschließend entsprechend weiter verarbeitet, analysiert und ausgewertet werden müssen. Aus diesem Grund ist die Entwicklung leistungsfähiger und mächtiger Analysewerkzeuge von immer größerer Bedeutung, wobei die Visualisierung eine zunehmend entscheidende Rolle spielt.

Die visuelle Aufbereitung von Daten, sowohl in der Vor- als auch in der Nachbearbeitung, bietet einen intuitiven und direkten Einblick in die Ergebnisse. Darum müssen passende Visualisierungen entwickelt und gegebenenfalls auf die speziellen Bedürfnisse der Anwendung angepasst werden. Sinn und Zweck von Visualisierungsanwendungen ist letztlich nicht allein die visuell ansprechende Darstellung an sich, sondern die Tatsache, eine ideale Mischung aus Datenreduktion und Betonung der relevanten Eigenschaften und Charakteristika zu finden, um sowohl einen einfachen Umgang mit den Daten als auch eine intuitive Analyse dieser zu ermöglichen und zu unterstützen.

In zahlreichen Anwendungsgebieten spielt Geometrie eine entscheidende Rolle. Der Hauptbeitrag dieser Arbeit behandelt geometrische Kernpunkte von Visualisierungsmethoden, einerseits im Bereich der virtuellen Prototypenentwicklung in der Automobilindustrie, und in der Molekulardynamik andererseits. In beiden Anwendungsgebieten liegt die Herausforderung darin, abhängig von den jeweiligen Anforderungen, sowohl die notwendige Korrektheit von Gestalt, Geometrie und Topologie zu gewährleisten, um zuverlässige Unterstützung der Analyse sicherzustellen, als auch durch hochwertige Darstellungsgüte spezifische Eigenschaften der Daten herauszuarbeiten, ohne dabei wichtige Details zu verbergen. Nichtsdestotrotz ist der Fokus in Bezug auf Geometrie in beiden Themengebieten unterschiedlich gelagert.

Einerseits, wie im Gebiet der Prototypenentwicklung von Automobilen, sind zuverlässige Geometriemodelle entscheidend – sowohl für robuste Simulationskonfigurationen als auch für verlässliche Ergebnisse – da die Auswertung der geometrischen Eigenschaften den grundlegenden Zweck der Simulationen darstellt. Da die Simulationen in diesem Arbeitsgebiet üblicherweise auf der

Methode der Finiten Elemente (FE) beruhen, ist auch die Visualisierung entsprechend netzbasiert. In dieser Dissertation werden hierzu neue Methoden für die Bearbeitung, Anpassung und (Re-)Konstruktion der Geometrie und geometrischer Merkmale vorgestellt – jeweils auf die speziellen Anforderungen der Anwendung zugeschnitten.

Andererseits ist häufig, wie im Umfeld der Molekulardynamik, die geometrische Gestalt eines Modells nicht Gegenstand der Simulation, sondern legt die Randbedingungen der Simulation fest und spielt damit weiterhin eine tragende Rolle im Auswertungsprozess. Aus diesem Grund betont die vorliegende Arbeit in diesem Gebiet das Potential geometrischer Sachverhalte als Rahmenbedingungen für die Datenstrukturierung und die intuitive Evaluierung bei der Analyse der Simulationsdaten. Da die Moleküle selbst keine intrinsische Darstellung besitzen, bringen Moleküldarstellungen immer ein gewisses Maß an Abstraktion mit sich. Diese Tatsache kann wiederum ausgenutzt werden, um semantisch aussagekräftige Moleküldarstellungen zu definieren, die auf sehr unterschiedlichen intrinsischen und geometrischen Eigenschaften der Daten basieren.

Die praktische Relevanz der in dieser Arbeit vorgestellten Methoden konnte durch die enge Zusammenarbeit mit Wissenschaftlern der zugehörigen Forschungsgebiete sichergestellt werden, was dazu führte, dass diese Methoden ihren Weg in die Anwendung fanden: In die aktuelle Forschung im Falle der Molekulardynamik und in die kommerziellen Anwendungen im Falle der Analysemethoden im Bereich der Finiten-Element-Methode.

1 Introduction

Numerical simulations play a soaring role in many research fields as well as in product prototyping. The ongoing fast increase of computational power not only enables these simulations of complex non-linear and highly dynamic processes but also allows for further growing problem sizes and makes optimization processes and parameter studies with numerous simulation runs affordable. One of the often underrated consequences of this development is the pursuant resulting amount of simulation data that has to be processed, analysed and evaluated accordingly. Therefore, the development of powerful and capable analysis tools and methods likewise gains in importance, so that visualization plays an increasingly crucial role since the visual conditioning of the data – both in simulation pre- and post-processing – provides an intuitive and fast insight.

Consequently, a visual representation of the simulation data, based on its geometric properties, as well as of its derived meta data is necessary. Here, Geometry Processing applies, which is a research field "geared towards the creation of mathematical foundations and practical algorithms for the processing of complex geometric data sets, ranging from acquisition and editing all the way to animation, transmission and display. As such it draws on many disciplines spanning pure and applied mathematics, computer science, and engineering."[1] As a result "applications of geometry processing algorithms cover a wide range of areas from multimedia and entertainment, to bio-medical computing, reverse engineering, and to classical computer aided design."[2] Depending on the nature of the data the appropriate geometry may be easy to determine, especially if real geometric objects are the simulation subjects such as in structural mechanic simulations. Thus, the geometry can be directly represented while the visualization of the derived meta data and the interaction methods may still pose a challenge. In other simulation fields the geometry is not given intrinsically and, thus, has to be extracted or generated from data properties.

An example for the first kind of data is data from the area of car prototyping where reliable geometric models are of superior importance for both robust simulation set-ups and trustable results, since the evaluation of the geometric properties of the model are the principal purpose of simulation. The simulations in this field are usually based on finite element (FE) methods where

[1] From *Symposium on Geometry Processing* `http://www.lsi.upc.edu/~alvar/SGP07/`

[2] From *Symposium on Geometry Processing* `http://www-i8.informatik.rwth-aachen.de/old-site/SGP/sgp03/`

the geometry is subdivided into smaller netlike arranged elements. Thus, the visualization is mesh-based accordingly. In this work new methods for processing, customization and (re-)construction of FE geometry and geometric characteristics of the models are presented in Chapter 2, tailored to the specific needs of the simulation field. Namely a new method for volume-based remeshing of models to be used in finite element simulations of the early car development process stage is proposed in Section 2.2. In contrast to the large majority of common remeshing algorithms, usually developed for the triangulation of smooth and often closed surfaces, this approach specifically focuses on FE meshes using a binary volume and generating a quad mesh on isosurface reconstruction. Subsequently, valuable FE mesh optimization methods are presented in Section 2.3 that ensure essential element quality appropriate for FE-based simulations, while preserving the underlying geometry as well as providing interactive feedback and evaluation of the element quality and surface integrity. Additionally, a novel tool for filling holes in FE models is detailed in Section 2.4, where both the detection and closure of these holes is handled. This is needed, for instance, in later stages of car development when acoustic and fluid simulations are carried out, for example considering noise or air flow inside the passenger compartment. Unlike crash simulations, where the model may contain holes caused by constructive cut-outs or unmodelled parts that are of no structural relevance, these simulations require a watertight model of the simulation domain.

In other application fields the visual representation and, thus, the geometry has to be derived implicitly from the simulation data. For instance, in computational fluid dynamics (CFD) simulations the interaction of liquids and gases with explicitly defined surfaces is analysed. The geometry of the surface is thus given and can be visualised accordingly, but the flow, which is the simulation purpose, has no inherent geometric description. Therefore, meaningful geometric properties have to be derived from the data and an intuitive representation has to be found – which, to a certain extent, inevitably leads to abstraction. In some fields, where no inherent geometrical representation is given by the data, some kind of representation is commonly used and the user is familiar with it. Examples are arrows representing vector fields, such as flow, widely known from weather forecasts, but also molecular representations where the atoms are represented by spheres and/or lines symbolising the atom connections within the molecule. These commonly and widely known representations have to be kept in mind when developing new methods to picture simulation data. On one hand they have to be improved and updated where possible but, on the other hand, also alternative or supplementary representations have to be developed in order to enable and promote new analysis methods.

Furthermore, the progress in simulation applications and techniques also leads to the need for new visualization techniques since more and often new kind of

data has to be analysed and, thus, visualised. But also the consequent speed-up of the simulation and analysis process demands for fast techniques and therefore interactive visualization. Consequently, beside the development of these new representations and visualizations, both the inherent given geometric representations and the commonly used derived representations have to be updated in order to enable new analysis methods by providing more accurate but also faster and, thus, interactively usable representations to make further evaluations affordable.

Here, the application field of molecular dynamics is a valuable example, where the geometric shape of the model often is not the simulation purpose but dictates the simulation constraints, and, thus, still plays an essential role in analysis tasks. The work in this area presented in Chapter 3 emphasises the capability of geometric issues as basic conditions and context for essential data structuring and intuitive evaluation during simulation data analysis. Since the molecules themselves do not have an intrinsic geometry molecular representations always entail abstraction up to a certain extent. This fact, in turn, can be exploited to create semantically expressive molecular representations based on very different intrinsic and geometric properties of the data. There are various well-established representations but their often complicated geometrical structure poses essential challenges for the interactive visualization and analysis, particularly of time-based data.

One of the major difficulties in this application area is to grant interactive support for time-based data while providing superior visual quality. Hence, in Section 3.3 this work presents new methods for established protein representations although supporting time-based datasets. But the major contribution to the application field deals with the processing of the circumfluent solvent in the protein-solvent system during simulation post-processing – an analysis task often neglected by the well-established analysis tools, much to the researchers' regret. Therefore, beside various solvent filtering options, a new special method was developed, presented in Section 3.6.2, to analyse motion patterns within the solvent. Namely, possibly existing major paths of the solvent molecules are extracted in order to evaluate the protein behaviour and to detect the interesting regions for further investigation on the protein-solvent interaction as well as the solvent flow through a partially blocked protein tunnel. With this approach the focused solvent motion of the whole simulated trajectory is condensed into a single time frame yet giving a picture of the solvent motion in the trajectory as a whole. In addition, another attempt to emphasise the dynamics of time-based protein data is detailed in Section 3.5, giving haptic feedback of the protein flexibility.

A further step towards data abstraction was taken in the visualization of diffusion tensor imaging (DTI). In this application from neurosurgery, three-dimensional diffusion tensors are derived from magnetic resonance imaging.

The commonly chosen implementation by spherical tubes can only map two of the tensor eigenvalues whereas our approach of the so-called *hyperstreamlines*, tubes with ellipsoidal cross section, presented in Section 3.7.1, results in a considerably more complicated geometry in terms of raycasting ability but was the first one to visualise all three eigenvalues.

Derived from these different application situations, various geometric and algorithmic approaches may be used. Where inherent geometry is given by the data, this geometry should be visualised directly. An example is the area of structural mechanics, as mentioned before. There, the simulation is usually based on the finite element method where the simulated geometry is subdivided into smaller subgeometries, called the elements, for example quads or triangles. Obviously these elements have to be represented explicitly since their shape and additionally applied properties are the simulation subject. In areas where the geometry is not given directly but geometric properties first have to be derived from the data another technique is applicable: the procedural representation of the data. There, the geometry is described mathematically, for example by implicit representation equations, but not discretised into triangles. For the visualization, these equations are evaluated and the surface is then visualised, e.g. by raycasting: The intersections between rays from the view position to all areas of the object and the object itself are calculated analytically resulting in an implicit representation of the geometry, detailed in Section 1.1. The intersection calculation itself only gives the distance between the viewer and the object in that point but further necessary properties such as lighting can also be calculated from these equations. This procedural approach is pursued in this work in the field of molecular dynamic simulations where, to a large extent, the molecular representations as well as derived geometry like motion paths of molecules in the circumfluent solvent are visualised using raycasting.

All methods presented in this work, either tailored to meshed surfaces as in the FE-based context or mesh-free geometry in the latter one, focus on the geometric concerns of the visualization applications. The shared primary objective is the supply of intuitive specialised approaches for data analysis in the particular application field making use of the power of geometric issues to provide a better and deeper insight into the simulation data and thus the simulated processes. So, the representation of the data can be chosen according to the application needs, satisfying the required correctness of the shape as well as geometry and topology in order to ensure reliable analysis support, and providing superior visual quality, elaborating the data characteristics without concealing relevant features.

The applicability of the methods presented in this work were assured by the close collaboration with scientists in the dedicated application fields. As a result, they found their way to be used in recent research, in the case of

molecular dynamics, as well as in commercial application tools, in the case of the finite element analysis methods.

In order to provide the basis for the methods presented in Chapter 2 – mesh-based geometries – and Chapter 3 – mesh-free data representations –, the following sections briefly introduce the basic visualization process and a short introduction to the basic mathematics for geometry processing in the context of this work.

1.1 The Basic Visualization Process

Visualising something means, in general, to form a mental image, to bring it as a picture before the mind, to make something able to be seen or noticed – to make it *visible*. In the context of scientific visualization, this means making (often abstract) data visible in order to recognise relations and to get further insight. This can range from very simple visualizations like bar plots or mathematical graphs, over classical maps or the afore mentioned arrows symbolising the air flow in weather maps, up to very specialised representations tailored to specific application fields. Meanwhile visualization not only covers the generation of visual images of data but also enables the interaction with them.

The basic process from data retrieval to a rendered image is illustrated in Figure 1.1, the *visualization pipeline*. The visualization process starts with the data which is to be visualised – and which may originate from various data sources such as measurements (e.g. sensor data), data bases or from simulation – the *raw data*. Since the main objective is to achieve deeper insight and to explore important features in the data, the relevant data has to be extracted, unnecessary data has to be discarded – the data gets filtered to result in the *visualization data*. This step may also include processing such as data format conversion, cropping, interpolation and classification. In the next step the data is mapped to a geometric shape and graphical primitives in order to get a renderable representation and to be able to make the abstract quantities visible. These primitives include points, lines, surfaces and volumes with various attributes such as colour and texture. In the final step these primitives are rendered and thus the designated visualization is generated – which may, for example, be either an image or a video.

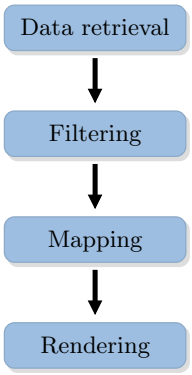


Figure 1.1: Basic steps of the visualization pipeline.

Advanced visualizations provide interactivity in the sense that the various stages of data in the visualization pipeline can be manipulated interactively by the user. These manipulations can range from rotating and moving the view of the dataset, over the interactive manipulation of the geometry and direct manipulation of the filtering process, up to interactive manipulation of the data acquisition, the *simulation steering*. Accordingly, visualizations may, but not necessarily have to, cover this whole scope. Due to the rapid increase in computational power and programming flexibility of modern graphics hardware on the other side, all these stages – depending on the application field – may be realised on the graphics hardware while this was reserved to the rendering stage just until few years ago. Especially the generation of geometry is increasingly performed on the graphic processing unit (GPU).

The more detailed technical process from geometric definitions to a rendered image on the screen is detailed in the *graphics* or *rendering pipeline*, illustrated in Figure 1.2. Based on the geometric model of the scene with curves, surfaces and volumes together with lighting, the geometric shapes are described by their geometry (vertices) and their topology. In the first stage of the pipeline the geometric primitives undergo various transformations: Starting with the (affine) model transformation, the model is transformed from its local coordinate system into the global *world coordinates*. In this world coordinate system the whole scene is composed, the view is defined and lighting is specified. With the subsequent *view transformation*, the scene items are converted into *view coordinates* where culling (e.g. backface elimination) is carried out and the scene is clipped into the 3D view volume. The *projection* or *normal transformation* converts the remaining scene items into normalised coordinates. In this state, the rasterization finally takes place and shading is applied to the remaining primitives resulting in the final screen image.

Depending on the rendering approach as well as the lighting mode, various variations of this basic and classical rendering pipeline evolved. The process description can be found more detailed in [FVF95, Wat99, AMHH08] and the regularly updated *red book* [Wat09].

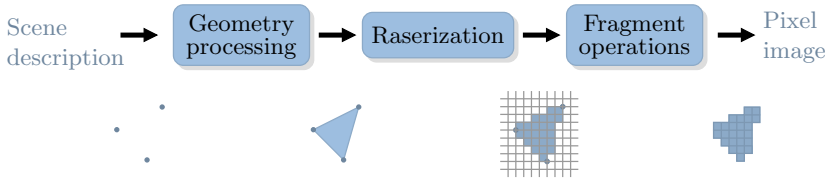


Figure 1.2: The main stages of the standard rendering pipeline, exemplified by the representation of a triangle.

An alternative approach to the widely used triangle-based rendering of geometry is raycasting – an image-space algorithm which is performed pixel by pixel. This method is especially beneficial for the rendering of surfaces which are described by mathematical implicit or parametric functions as described in Section 1.2.1 since the tesselation of the surfaces can be refrained from. The basic idea in raycasting is that light rays are cast from the viewer (the camera) position into the scene, where the rays are crossing the image plane and intersect with the geometry, as illustrated in Figure 1.3. This intersection point then defines the shading of the corresponding pixel. Thus, for each pixel of the image which is to be displayed such a ray is calculated. Unlike in raytracing no secondary rays are comprised in the calculation, thus, no reflections or shadows are determined by the ray.

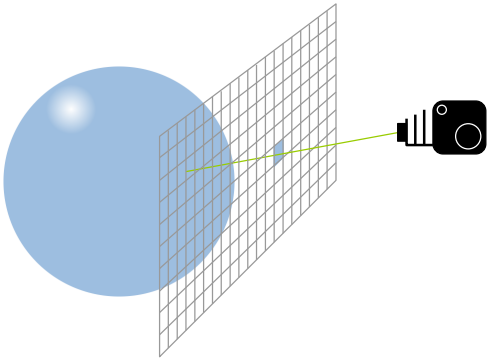


Figure 1.3: Schematic drawing of raycasting

In GPU raycasting the geometry to be displayed, that is the implicit function describing this geometry, is generated on the GPU while only a minimum of data is transferred from CPU to GPU. The geometry is usually represented by glyphs, often composed of several simple shaped elements like spheres or cylinders – but not restricted to these. For each of these glyphs a tight proxy geometry for the glyph's projection onto the image plane is calculated, favourably a square or a rectangle. This proxy geometry is transferred to the GPU, along with vectors containing the parameters for the implicit function describing the correct geometry. For each pixel that is covered by this primitive a ray is cast from the viewer's (or camera) position through that pixel. The intersection of this ray with the geometry decides whether the fragment is discarded or the intersection point and surface normal are calculated for the correct shading of the resulting pixel. This method clearly results in pixel-accurate silhouettes, while the lack of this accurate silhouette is one of the major drawbacks of tesselation.

This method is especially useful for datasets with geometric primitives that can be easily described by implicit functions of low degrees such as the afore mentioned spheres and cylinders, used in this work in the context of molecular representations detailed in Section 3.3. The drawback of this approach is that the intersection of the ray with the geometry has to be calculated analytically, which is not solvable for all functions. One possible solution to this problem is an approximation of the intersection, for example by sphere tracing [Har96], which was used in this work to raycast tubes of rotating ellipsoidal cross-sections, detailed in Section 3.7.

1.2 Mathematics for Geometry Processing

This chapter introduces some mathematical fundamentals providing the basis of the methods described later in this work. Starting with geometric aspects, such as the definition and representations of curves and surfaces, leading to their fundamental properties, and the very short presentation of splines as an alternative curve representation, the chapter closes with a brief discussion of ordinary differential equation systems and their appropriate numerical solvers – especially needed in geometry processing of meshed surfaces.

Getting into more detail about mesh representation and discretised curves and surfaces along with some remarks about discrete differential geometry, as a transition from the theoretical mathematical fundamentals to the consecutive mesh-based geometry representation in Section 1.3, we now focus on the mathematical description of curves and surfaces.

1.2.1 The Geometry of Curves and Surfaces

Curves and surfaces play a fundamental role in computer graphics and especially in geometry processing. Surfaces are used to represent the desired geometry, either tesselated in meshes or defined by a parametrised representation, for example in point-based graphics.

The major part of the following issues is part of the classical differential geometry and will be described only briefly here. Detailed introductions to differential geometry can be found, for example, in [Dar87, Kre59, Lau60, dC76, Spi79, Kli73, BL73]. For a more generalised approach, the semi-riemannian geometry, confer to [O'N83].

Representations of Curves and Surfaces

In 2D, curves are often associated with the graph of a real-valued function

$$\begin{aligned} f:\ x &\mapsto f(x), \\ \mathbb{R} \supseteq D &\to W \subseteq \mathbb{R} \end{aligned} \tag{1.1}$$

where for each x value there is one unique function value $y = f(x)$. This representation is called *explicit* representation. A simple example is the semicircle of radius r given by

$$y = f(x) = \sqrt{r^2 - x^2}, \quad \text{where} \quad |x| \leq r. \tag{1.2}$$

The graph of this function is given in Figure 1.4(a) for radius $r = 2$.

This definition can also be used for surfaces in $\mathbb{R}^n$:

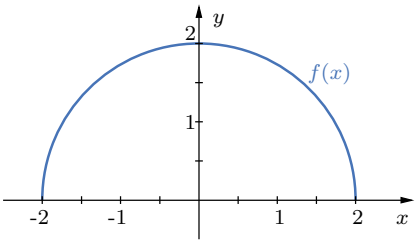
$$\begin{aligned} f:\ x &\mapsto f(x), \\ \mathbb{R}^{n-1} \supseteq D &\to W \subseteq \mathbb{R} \end{aligned} \tag{1.3}$$

where for each $x \in \mathbb{R}^{n-1}$ there is one unique function value $f(x) \in \mathbb{R}$. The definition of the northern hemisphere of radius r, thus, can be defined by

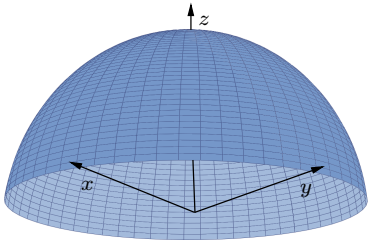
$$z = f(x, y) = \sqrt{r^2 - x^2 - y^2}, \quad \text{where} \quad x^2 + y^2 \leq r^2 \tag{1.4}$$

The corresponding plot is depicted in Figure 1.4(b).

Obviously, the explicit representation method lacks the flexibility of defining closed curves and surfaces or more complicated scenarios like self-intersections, as those need more than one function value per input point.



(a) Semicircle of radius $r = 2$ located at the origin and defined by (1.2)



(b) The northern hemisphere of radius $r = 1$ defined by (1.4)

Figure 1.4: Examples of simple explicitly defined functions.

A representation method able to define these kinds of curves and surfaces is the *implicit* representation

$$F(x) = 0 \tag{1.5}$$

with $x \in \mathbb{R}^n$. This representation includes more general curves and surfaces than the previous explicit ones (1.1) and (1.3), because the graphs do not have to be single-valued. For $x \in \mathbb{R}^2$, this representation corresponds to (1.1) iff the equation $F(x, y) = 0$ can be resolved uniquely by y. The example (1.2) above then results in

$$x^2 + y^2 - r^2 = 0, \quad \text{with} \quad -r \leq x \leq r \quad \text{and} \quad y \geq 0, \tag{1.6}$$

and the hemisphere of Equation (1.4), as an example in $\mathbb{R}^3$, can be rewritten as

$$x^2 + y^2 + z^2 - r^2 = 0, \quad \text{with} \quad x^2 + y^2 + z^2 \leq r^2 \quad \text{and} \quad z \geq 0. \tag{1.7}$$

More generally speaking, the set of vectors $x \in \mathbb{R}^n$ such that

$$a \cdot x - c = 0$$

with a constant $c \in \mathbb{R}$ and $a \in \mathbb{R}^n$ is a subspace in $\mathbb{R}^n$ and is called a *hyperplane*. Hence, a space curve in $\mathbb{R}^n$ can be defined by the intersection of two surfaces

$$F_1(x) = 0, \quad F_2(x) = 0$$

for $x \in \mathbb{R}^n$.

A very prominent example of a surface in 3D is the representation of a plane, especially the so-called *Hessian normal form*

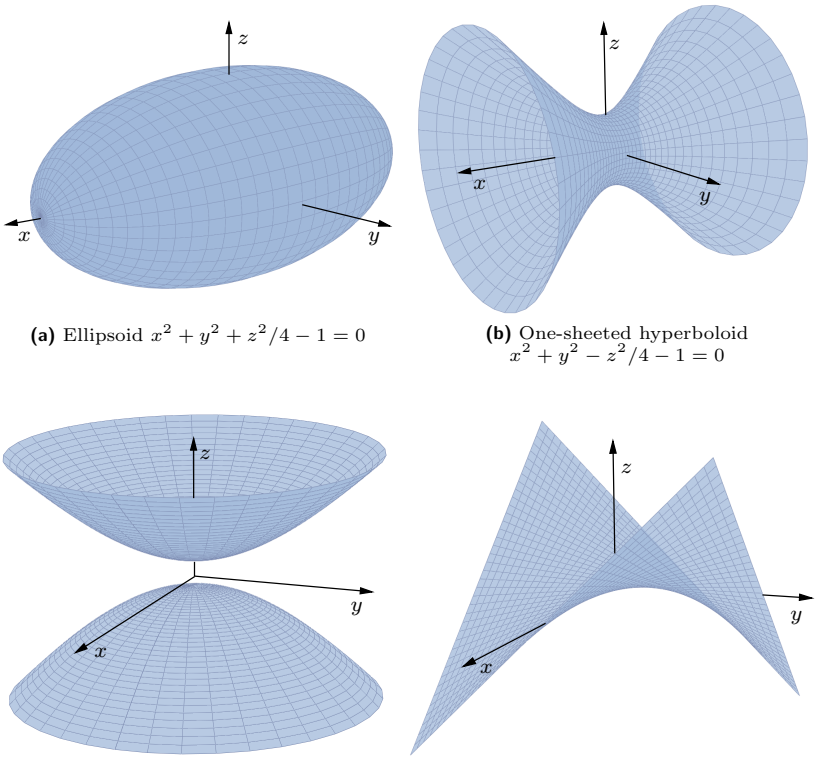
$$N \cdot x - d = 0, \tag{1.8}$$

where $N \in \mathbb{R}$ denotes the normalised normal vector of the plane and $d \in \mathbb{R}$ the plane's distance to the origin. Inserting a point $p \in \mathbb{R}^n$ into the left hand side of Equation (1.8), we get the point's (signed) distance $\tilde{d}$ to the plane: $\tilde{d} = N \cdot p - d$. The sign determines the side of the plane where the point is located: If $\tilde{d} > 0$ the point is in the half-space determined by the direction of N, and if $\tilde{d} < 0$ it is in the other half-space. For $\tilde{d} = 0$ the plane runs through the point, i.e. p lies in the plane.

Using a second order function of the form

$$F(x, y, z) = (a_1 x + a_2 y)^2 + (a_3 y + a_4 z)^2 + (a_5 x + a_6 z)^2 \tag{1.9}$$

in (1.5) we get the so-called *quadrics*. The intersection of a quadric with a plane determines a conic section – proper or degenerated. Some prominent examples of quadrics are depicted in Figure 1.5.



(a) Ellipsoid $x^2 + y^2 + z^2/4 - 1 = 0$

(b) One-sheeted hyperboloid $x^2 + y^2 - z^2/4 - 1 = 0$

(c) Two-sheeted hyperboloid $-x^2 + y^2 - z^2/4 - 1 = 0$

(d) Hyperbolic paraboloid $x^2/2 - y - z^2/2 = 0$

Figure 1.5: Examples of simple quadrics centred at the origin.

A third representation method is the *parametric* representation. Curves c defined on an interval $I \subset \mathbb{R}$ with $c : I \mapsto \mathbb{R}^n$ can be written as

$$x = c(t) = \begin{pmatrix} c_1(t) \\ \vdots \\ c_n(t) \end{pmatrix} \tag{1.10}$$

with the curve parameter $t \in I \subset \mathbb{R}$, and surfaces by

$$x = \Phi(u) \tag{1.11}$$

with $u \in \mathbb{R}^{n-1}$, respectively. The examples above can be expressed in parametric representation as follows: The semicircle of Equations (1.2) and (1.6),

depicted in Figure 1.4(a) with radius $r = 2$, is defined by

$$c(\varphi): \quad \begin{aligned} x &= r\,\cos\varphi \\ y &= r\,\sin\varphi \end{aligned} \tag{1.12}$$

with $0 \leq \varphi \leq \pi$ and r being the circle's radius. The northern hemisphere, shown in Figure 1.4(b), defined explicitly by Equation (1.4) and implicitly by (1.7) can be written as

$$\Phi(\vartheta, \varphi): \quad \begin{aligned} x &= r\,\sin\vartheta\,\cos\varphi \\ y &= r\,\sin\vartheta\,\sin\varphi \\ z &= r\,\cos\vartheta \end{aligned} \tag{1.13}$$

with $0 \leq \varphi \leq 2\pi$ and $0 \leq \vartheta \leq \pi/2$, where r denotes the hemisphere's radius.

This representation is of special importance in differential geometry, as major properties of curves and surfaces are usually defined on it.

Properties of Curves

A curve $x = c(t)$, defined by (1.10), with $x \in \mathbb{R}^n$ and $t \in I$, interval $I \subseteq \mathbb{R}$, is called *differentiable* iff every component $c_i(t)$, $0 \leq i \leq n$, is differentiable for all $t \in I$. The derivative of c is denoted by $dc/dt = \dot{c}(t)$ – as illustrated in Figure 1.6.

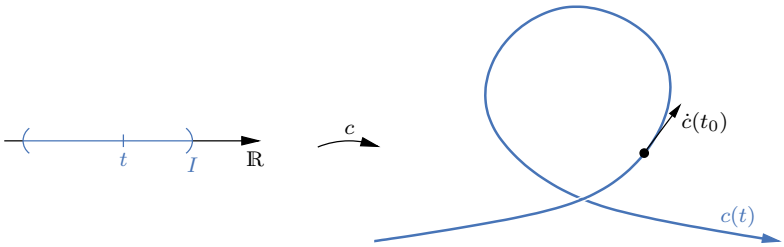


Figure 1.6: Example of a simple curve $x = c(t)$.

A curve $c(t)$ is called *regular* in case

$$\dot{c}(t) \neq 0 \quad \text{for all} \quad t \in I\,. \tag{1.14}$$

For regular curves the *arc length* s is defined by

$$s = \int_{t_0}^{t} |\dot{c}(\tau)|\,d\tau \tag{1.15}$$

where $|\dot{c}(\tau)| = \sqrt{(\dot{c}_1(\tau))^2 + \ldots + (\dot{c}_n(\tau))^2}$ and s measures the length of the curve arc between t_0 and t. Because of (1.14), every regular curve can be reparametrised by arc length using $ds/dt = |\dot{c}(t)|$. Hence, a curve is parametrised with respect to s iff $|\dot{c}(t)| = 1 = ds/dt$. This parametrization is also called *natural parametrization*. The derivative with respect to the arc length parameter s is denoted by $dc/ds = c'(s)$.

Curves $c : I \mapsto \mathbb{R}^n$ for which the first $n-1$ derivatives $\dot{c}(t), \ddot{c}(t), \ldots$ are linearly independent for all $t \in I$ are called *Frenet curves*. Therefore, the according *Frenet frame field* $c(t), e_1(t), \ldots, e_n(t)$, forming a right-handed coordinate system, can be defined for these curves:

$$\langle e_i(t), e_j(t)\rangle = \delta_{ij} = \begin{cases} 1 & \text{for} \quad i = j \\ 0 & \text{for} \quad i \neq j \end{cases} \quad 1 \leq i, j \leq n,\ t \in I$$

$$\operatorname{lin}\left(e_1(t), \ldots, e_i(t)\right) = \operatorname{lin}\left(\dot{c}(t), \ldots, c^{(i)}(t)\right)$$

$$\langle e_i(t), c^{(i)}(t)\rangle > 0, \quad 1 \leq i \leq n-1$$

where $\langle ., . \rangle$ denotes the scalar product, $\operatorname{lin}(\ldots)$ the linear hull of the enclosed vectors and $c^{(i)}(t) = d^i c/dt^i$. Furthermore, the following *Frenet formulae* can be defined:

$$c' = e_1 \tag{1.16a}$$
$$e_1' = k_1 e_2 \tag{1.16b}$$
$$e_2' = -k_1 e_1 + k_2 e_3 \tag{1.16c}$$
$$e_3' = -k_2 e_2 + k_3 e_4 \tag{1.16d}$$
$$\vdots \quad \ddots \tag{1.16e}$$
$$e_{n-1}' = -k_{n-2} e_{n-2} + k_{n-1} e_n \tag{1.16f}$$
$$e_n' = -k_{n-1} e_{n-1} \tag{1.16g}$$

where k_i is called the *i-th Frenet curvature* of curve c and $k_1 > 0, \ldots, k_{n-2} > 0$.

Theorem 1 (Fundamental Theorem of Space Curves)
For given $k_i(s) : I \to \mathbb{R}$, $1 \leq i \leq n-1$, $k_i \in C^{n-i-1}(I)$ and $k_1, \ldots, k_{n-2} > 0$ there exists a Frenet curve $c : I \to \mathbb{R}^n$ with $k_i, 1 \leq i \leq n-1$, as its i-th Frenet curvature and s as its arc length parameter. This curve is unique except for orientation and location in space.

A proof of Theorem 1 can be found in [Kli73].

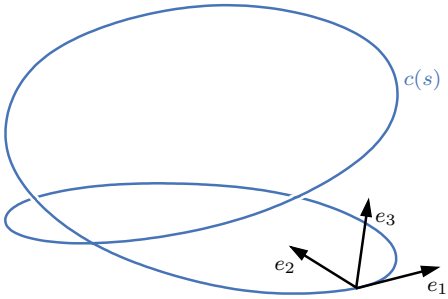


Figure 1.7: Example of a 3D curve $c(s) = (cos(s)cos(s), cos(s)sin(s), sin(s))^T$ with its Frenet frame field e_i.

Namely in $\mathbb{R}^3$, the Frenet frame field is given by the following normalised vectors, illustrated in Figure 1.7: The tangent vector $T = \dot{c}/|\dot{c}| = e_1$, the *binormal* vector $B = (\dot{c} \times \ddot{c}) / |\dot{c} \times \ddot{c}| = e_3$ and the principal vector $N = B \times T = e_2$. Furthermore, k_1 is called the *curvature* k of the curve and, roughly speaking, gives a measure of how much the curve differs from a straight line in this point, and k_2 is called *torsion* τ and gives a measure of how much the curve gets off a plane. Hence, curves with torsion $\tau \equiv 0$ are plane. Both, the curvature as well as the torsion of a space curve are independent of the curve's parametrization but do depend on the parameter along the curve.

A circle with radius r is a simple example of a plane curve (i. e. torsion $\tau \equiv 0$) of constant curvature $k = 1/r$. Given the circle

$$c(t) = \begin{pmatrix} r\cos(t) \\ r\sin(t) \\ 0 \end{pmatrix} \tag{1.17}$$

we get the Frenet formulae according to (1.16a) – (1.16c)

$$\begin{aligned} c'(t) &= \dot{c}(t)/|\dot{c}(t)| = \begin{pmatrix} -\sin(t) \\ \cos(t) \\ 0 \end{pmatrix} = e_1 \\ e_1'(t) &= \dot{e}(t)/|\dot{c}(t)| = \frac{1}{r}\begin{pmatrix} -\cos(t) \\ -\sin(t) \\ 0 \end{pmatrix} = ke_2 \\ e_2'(t) &= \frac{1}{r}\begin{pmatrix} \sin(t) \\ -\cos(t) \\ 0 \end{pmatrix} = -ke_1 + \tau e_3, \end{aligned}$$

and thus, $k = 1/r$ and $\tau \equiv 0$.

Figure 1.8(a) shows the circle (1.17) together with its Frenet frame.

In contrast, the helix defined by (1.18) and depicted in Figure 1.8(b) although has likewise constant curvature, but non-zero torsion:

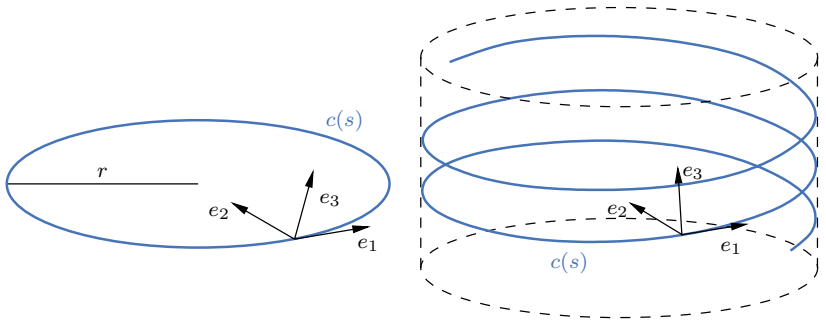
$$c(t) = \begin{pmatrix} r\cos(t) \\ r\sin(t) \\ h\,t \end{pmatrix} \tag{1.18}$$

reparametrised by arc length

$$\begin{aligned} s(t) &= \int_0^t |\dot{c}(u)|\,du = \sqrt{r^2+h^2}\,t \\ &\Rightarrow t(s) = s/\sqrt{r^2+h^2} \end{aligned}$$

we get the Frenet formulae

$$\begin{aligned} c'(s) &= \frac{1}{\sqrt{r^2+h^2}} \begin{pmatrix} -r\sin\left(s/\sqrt{r^2+h^2}\right) \\ r\cos(s/\sqrt{r^2+h^2}) \\ h \end{pmatrix} = e_1 \\ e_1'(s) &= -\frac{r}{r^2+h^2} \begin{pmatrix} -r\cos\left(s/\sqrt{r^2+h^2}\right) \\ -r\sin\left(s/\sqrt{r^2+h^2}\right) \\ 0 \end{pmatrix} = ke_2 \\ e_2'(s) &= \frac{1}{\sqrt{r^2+h^2}} \begin{pmatrix} \sin\left(s/\sqrt{r^2+h^2}\right) \\ \cos\left(s/\sqrt{r^2+h^2}\right) \\ 0 \end{pmatrix} = -ke_1 + \tau e_3 \end{aligned}$$



(a) Frenet frame field e_i at a circle of radius r, as defined in (1.17).

(b) Frenet frame field e_i at a helix of radius r and slope h, according to (1.18).

Figure 1.8: Simple space curves: The circle (a) has constant curvature and vanishing torsion, whereas the helix (b) has both constant curvature and torsion.

resulting in

$$\Rightarrow k = \frac{r}{r^2 + h^2}, \quad \tau = \frac{h}{r^2 + h^2}$$

On the other hand, a twisted curve with non-vanishing curvature is a cylindrical helix iff, at all of its points, the ratio of its curvature and torsion is the same [Kre59]: $k(s)/\tau(s) = \text{const}$.

Properties of Surfaces

A k-dimensional surface in $\mathbb{R}^n$ with $x : U \to M$ where $U \subset \mathbb{R}^k, M \subset \mathbb{R}^n$ and $k \leq n$ is called a *regular surface* if x is differentiable and an *immersion*, i.e.

$$\frac{\partial x}{\partial u^1}, \ldots, \frac{\partial x}{\partial u^k} \quad \text{linearly independent for all} \quad u = \left(u^1, \ldots, u^k\right) \in U.$$

Two special cases are $k = 1$ where the regular curve is defined by $\dot{x} \neq 0$, as stated in the previous section, and for $k = n-1$ the surface M is a *hyperplane*.

Furthermore, T_pM denotes the *tangent space* of the surface M in a point $p = x(u) \in M$ with

$$T_pM := \text{lin}\left(x_{,1}(u), \ldots, x_{,2}(u)\right), \tag{1.19}$$

where $x_{,i}$ denotes the partial derivative $x_{,i} := \partial x/\partial u^i$. The *tangent plane* in p is hence defined by $p + T_pM$, depicted in Figure 1.9.

According to the arc length of a curve (1.15), the metric of a surface is defined by the *first fundamental form* which defines the inner geometry of a surface – or more generally speaking, a manifold.

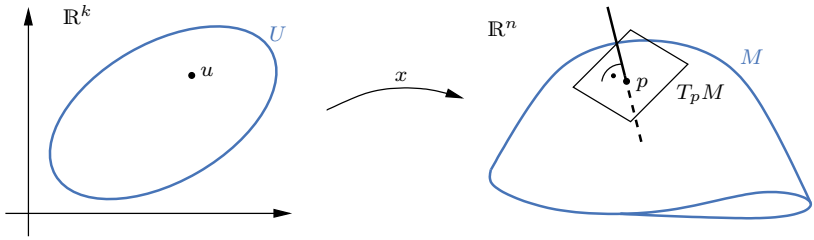


Figure 1.9: Example of a simple surface M with its tangent surface T_pM given in a sample point $p = x(u)$.

Definition 1 (First Fundamental Form)
The first fundamental form of a surface M in the point $p \in M$ is defined by

$$I_p(X,Y) := \langle X, Y \rangle \qquad \text{with} \quad X, Y \in T_pM. \tag{1.20}$$

I_p can also be written in parametric form: Using (1.19), the tangent vectors X and Y are transformed to

$$X = \sum_{i=1}^{k} \xi^i \, x_{,i}(u) \underset{\substack{Einstein \\ notation}}{=:} \xi^i \, x_{,i}(u),$$
$$Y = \sum_{j=1}^{k} \eta^j \, x_{,j}(u) \underset{\substack{Einstein \\ notation}}{=:} \eta^j \, x_{,j}(u).$$

and thus (1.20) results to

$$I_p(X,Y) = I_p\left(\xi^i \, x_{,i}(u), \eta^j \, x_{,j}(u)\right) = \xi^i \eta^j \, I_p\left(x_{,i}(u), x_{,j}(u)\right) = \xi^i \eta^j \, g_{ij}(u),$$

where $g_{ij}(u)$ are called the *coefficients* or *parametrization* of the first fundamental form. In matrix notation, (1.20) results to $I_p(X,Y) = \xi^T G \eta$ with $G = (g_{ij})_{1 \le i,j \le k}$ being a symmetric, positive definite matrix, $\xi^T = \left(\xi^1, \ldots, \xi^k\right)$, and $\eta^T = \left(\eta^1, \ldots, \eta^k\right)$. Gauss [Gau27] himself denoted the coefficients by E, F and G, a notation that remained common as well since then.

Using (1.20), the length of a tangent vector $X \in T_pM$ and the angle α between two tangent vectors $X, Y \in T_pM$ can be calculated by

$$\begin{aligned} |X| &= \sqrt{\langle X, X \rangle} = \sqrt{I_p(X,X)} = \sqrt{\xi^i \eta^j \, g_{ij}(u)}, \\ \cos\alpha &= \frac{\langle X, Y \rangle}{|X| \, |Y|} = \frac{I_p(X,Y)}{|X| \, |Y|}, \end{aligned} \tag{1.21}$$

and the length L of a curve c on the surface M results to

$$L(c) = \int_I |\dot{c}(t)| \, dt = \int_I \sqrt{\dot{u}^i(t) \, \dot{u}^j(t) \, g_{ij}(u(t))} \, dt, \tag{1.22}$$

because $\dot{c}(t) \in T_{x(u(t))}M$.

Accordingly, the area O of a surface patch M is

$$O(M) = \int_M \sqrt{\det(g_{ij}(u))} \, du^1 \ldots du^k.$$

As seen in the previous section, the curvature k of a curve describes the variation of the curve's tangent. This idea can be transferred to surfaces, leading to the question how much the surface drifts from its tangent plane within the

neighbourhood of a surface point p – or in other terms, the variation of the *unit normal vector field* on this surface patch.

The unit normal vector field N of a regular surface M is a differentiable map that assigns to each surface point p its corresponding unit normal vector $N(p)$:

$$N: x(U) \to \mathbb{R}^n \quad \text{with} \quad p \mapsto N(p).$$

This map has its values on the unit sphere S^{n-1}, thus the *Gauss map* can be defined [Gau27]:

$$\begin{aligned} N: M &\to S^{n-1} := \{v \in \mathbb{R}^n | \|v\| = 1\} \\ x(u) = p &\mapsto N(p) \quad \text{(abstract)} \\ u &\mapsto N(p) \quad \text{(parametric)} \end{aligned}$$

The Gauss map is a differentiable map, with its differential dN_p in p:

$$\begin{aligned} dN_p: T_pM &\to T_{N(p)}S^{n-1} \\ X &\mapsto dN_p(X) \quad \text{(abstract)} \\ X = \xi^i \left.\frac{\partial x}{\partial u^i}\right|_u &\mapsto \xi^i \left.\frac{\partial N}{\partial u^i}\right|_u \quad \text{(parametric)} \end{aligned}$$

As the Gauss map of a tangent vector is in the tangent space again ($dN_p(T_pM) \subset T_pM$), the following linear map $-dN_p$ called *Weingarten map* or *shape operator* can be defined

$$\begin{aligned} -dN_p: T_pM &\to T_pM \\ X &\mapsto -dN_p(X), \end{aligned} \tag{1.23}$$

which leads to the previously announced curvature of a surface, described in the following.

For each parametrised curve $c(t)$ on the surface M with $c(0) = p$ we consider the curve $N \circ c(t) = N(t)$ on the unit sphere S^{n-1}, which is equivalent to the restriction of the normal vector to the curve $c(t)$. The tangent vector $N'(0) = dN_p(c'(0))$ itself is a vector in T_pM (see Figure 1.10) and measures the variation of N restricted to the curve $c(t)$ around $t = 0$ [dC76]. Thus dN_p measures, metaphorically speaking, how much N turns away from $N(p)$ within the neighbourhood of p. For curves, this measure is the curvature k, for surfaces it results in a linear map.

Definition 2 (Second Fundamental Form)
The second fundamental form of a surface M in the point $p \in M$ is defined by

$$II_p(X, Y) := I_p(-dN_p(X), Y) \tag{1.24}$$

with $X, Y \in T_pM$, I_p the first fundamental form (1.20) *and $-dN_p$ the Weingarten map* (1.23).

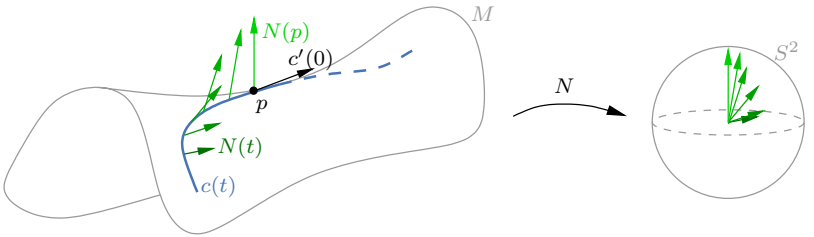


Figure 1.10: The variation of surface normals (coloured green) along a curve $c(t)$ in M illustrates the Weingarten map $-dN_p$. The map N is the Gauss map, shown for sample points of the curve $c(t)$.

In parameter form the tangent vectors X and Y are

$$X = \xi^i \left.\frac{\partial x}{\partial u^i}\right|_u \qquad \text{and} \qquad Y = \eta^j \left.\frac{\partial x}{\partial u^j}\right|_u ,$$

hence, (1.24) results to

$$II_p = \sum_{1 \le i,j \le n-1} h_{ij}(u)\, \xi^i\, \eta^j$$

with h_{ij} (also denoted by L, M, N) being the *coefficients* of the second fundamental form:

$$h_{ij}(u) = \left\langle -\frac{\partial N}{\partial u^i}, \frac{\partial x}{\partial u^j} \right\rangle = \left\langle N, \frac{\partial^2 x}{\partial u^i\, \partial u^j} \right\rangle .$$

Plugging this back into the Weingarten map (1.23), $-dN_p(X)$ can be written in matrix form

$$-\, dN_p(X) = (h^i_{\ j})(\xi^i) \tag{1.25}$$

where $(h^i_{\ j}) = (g_{pq})^{-1}(h_{kl}) := (g^{rs})(h_{kl})$ and with the coefficients

$$h^i_{\ j} := g^{ik}\, h_{kj}.$$

Based on (1.25) finally the curvature terms of the surface can be defined:

The eigenvalues $k_1(p), \ldots, k_{n-1}(p)$ of the Weingarten map (1.25) of the surface M in p are called the *principal curvatures* in p, the according directions, that is, the directions given by the eigenvectors, are called the *principal directions* at p. If for a regular connected curve c on M in all $p \in M$ the tangent line of c is a principal direction at p, then c is called a *line of curvature* of c.

Furthermore, the *i-th mean curvature* H_i is closely related to the principal curvatures k_i:

$$\begin{aligned} H_0 &:= 1 \\ H = H_1 &:= \frac{1}{n-1}(k_1 + \ldots + k_{n-1}) \\ H_2 &:= \frac{1}{\binom{n-1}{2}}(k_1k_2 + k_1k_3 + \ldots + k_{n-2}k_{n-1}) \\ &\vdots \\ H_i &:= \binom{n-1}{i}^{-1} \sum_{1 \le j_1 < \ldots < j_i \le n-1} k_{j_1} \cdot \ldots \cdot k_{j_i} \\ K = H_{n-1} &:= k_1 \cdot \ldots \cdot k_{n-1}. \end{aligned} \tag{1.26}$$

Two special cases of mean curvatures, as can be seen in the notation in (1.26), are $H_1 = H$ – called the *mean curvature* – and $H_{n-1} = K$ – called the *Gaussian curvature*, which can alternatively be calculated by

$$K = H_{n-1} = \prod_{1 \le i \le n-1} k_i = \det(h^i_j) = \det(g^{ik}\, h_{kj}) = \frac{\det(h_{ij})}{\det(g_{ij})} \tag{1.27}$$

Both curvatures play a crucial role in surface geometry. First, we go into detail about the Gaussian curvature K, as various properties are defined by it. In particular, points on the surface are classified depending on the Gaussian curvature in that point:

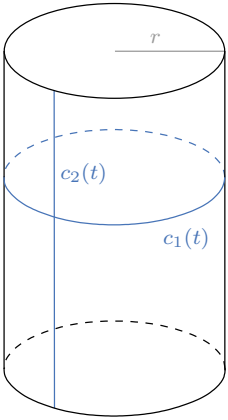
A point p on a surface is called
- *elliptic* if $K > 0$
- *hyperbolic* if $K < 0$
- *parabolic* if $K = 0$ with $k_i = 0$ and $\exists\, j \neq i \colon k_j \neq 0$
- *planar* if $K = 0$ with $k_1 = \ldots = k_{n-1} = 0$.

An example for elliptic points is the sphere S^{n-1}, where all points are elliptic, because the sphere has constant positive curvature: $N = -(p-m)/r$ and, thus, $N_{,i} = -x_{,i}/r$, where N is the surface normal in p, m the sphere centre and r the radius of the sphere. Therefore, $II_p = I_p/r$ and, thus, $h_{ij} = g_{ij}/r$ leading to $k_1 = \ldots = k_{n-1} = 1/r$, and the Gaussian curvature results to $K = \prod_i k_i = (1/r)^{n-1} = \text{const} > 0$. At hyperbolic points the principal curvatures have opposite sign (with an odd number of negative curvatures), like the points on a hyperbolic paraboloid which as previously depicted in Figure 1.5(d).

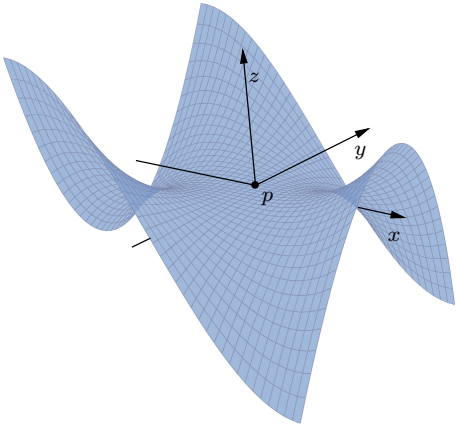
Parabolic points appear if at least one principle curvature vanishes in p, but at least one other principle curvature is non-zero, which, for example, can

be found at the cylinder surface where the two principle directions point one along the circle perpendicular to the rotation axis, the second one parallel to the axis, as shown in Figure 1.11(a).

In planar points, not only one but all principle curvatures vanish. A trivial example is any point on a plane, a non-trivial example in $\mathbb{R}^3$ is the point $p = (0, 0, 0)$ on the *monkey saddle* defined by $M(u, v) = (u, v, u^3 - 3v^2u)$, illustrated in Figure 1.11(b). This point p, on the other hand, is the only planar one on that surface.



(a) Cylinder with lines of curvature $c_1(t)$ and $c_2(t)$ marked in blue.



(b) Monkey saddle $M(u, v) = (u, v, u^3 - 3v^2u)$ with a planar point $p = (0, 0, 0)$.

Figure 1.11: Parabolic and planar points on 3D surfaces.

If, as in the case of points on a sphere or a plane, all mean curvatures in p are identical (i.e. $k_i = k_j \; \forall \; i \neq j, \; 1 \leq i, j \leq n - 1$), this point is called *umbilical point*. Hence, planar points are umbilical points likewise, and, conversely, if all points on a surface M are umbilical points, then M is either part of a sphere or a plane. Proofs can be found in [Kre59, Lau60].

Additionally, the total Gaussian curvature is defined as follows [Kli73, Spi79, O'N97]:

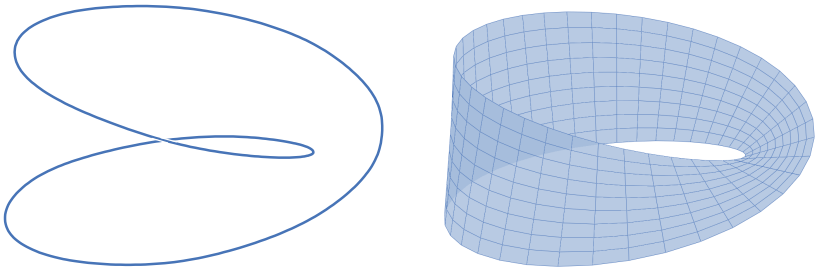
Theorem 2 (Gauss-Bonnet)
The total Gaussian curvature of a compact orientable geometric surface M is defined by

$$\iint_M K \, dA = 2\pi\chi(M) \, .$$

On the other hand, the mean curvature H can be used to define another special class of surfaces, the *minimal surfaces*. A regular parametrised bounded surface M with boundary curve c is called minimal surface if M has minimal area among all bounded surfaces with boundary curve c. This is equivalent to the fact that the mean curvature vanishes everywhere on M: $H \equiv 0 \ \forall p \in M$. For a proof of this equivalence refer to [dC76, Str69, Kre59]. As a consequence of the vanishing mean curvature everywhere on the surface, the Gaussian curvature of a minimal surface is non-positive with at most isolated zeros only, unless the surface is a plane. Obviously, the minimal surface enclosed by a plane curve is a part of a plane and thus the simplest example of a minimal surface. Regular curves result in smooth minimal surfaces, which makes them so interesting in many application fields. In the context of this thesis minimal surfaces are applied in Section 2.4 where holes in finite element models are being filled.

Bounded soap films are, in general, examples for minimal surfaces: The closed curve illustrated in Figure 1.12(a) bounds a minimal surface, a so called *Möbius strip* shown in Figure 1.12(b). Note that not all soap films are minimal surfaces according to the definition above, as the surface is claimed to be regular, whereas the boundary curve is not necessarily regular. A counterexample would be the soap film resulting from a cubical wire frame. A detailed introduction to minimal surfaces in $\mathbb{R}^n$ is given in [Nit75, Oss69] (originally in Russian [Oss67]).

The problem of finding the minimal surface enclosed by a given simple, closed, rectifiable curve c is the *Plateau Problem*. The existence of a solution to the general case was proven by Radó [Rad30] and Douglas [Dou31] independently. However, both had to admit the possibility of isolated singularities. The regularity of the classical solution to the Plateau Problem was finally shown by Osserman [Oss70]. For a detailed survey on the Plateau Problem confer to [Mee81].



(a) 3D boundary curve.

(b) Enclosed minimal surface: A Möbius strip.

Figure 1.12: A schematical example for a soap film minimal surface.

Splines

A frequent task on curves, especially in modelling and graphics, is to find a smooth interpolating or approximating curve between given points. A classical solution is the interpolation with polynomials, which works well for smaller problems, i.e. a small number of points. Since the polynomial degree increases according to the number of interpolation points, the polynomial curve shows oscillation for larger problems and gets unsuitable for the task of interpolation. To tackle this problem polynomial curves of lower degree can be assembled. This enhances the interpolation characteristics but the resulting curve is, in general, not differentiable in the points where two curves connect. This drawback can be overcome with *spline functions*, shortly called *splines*.

In their general definition they meet three conditions [SK06]:

- They are piecewise polynomial with degree $\leq k$.
- They are p times continuously differentiable.
- Beneath those functions interpolating a given set of points and meeting the previous two conditions, they are the most smooth ones.

The most frequently used splines are cubic splines with $k = 3$ and $p = 2$.

Cubic Splines Physically, a cubic spline corresponds to the curve defined by a thin wooden slat fixed at several positions and woven between these fixations, resulting to represent a smooth curve of minimal curvature [SK06].

Mathematically this can be formulated as follows:

Given a set of disjoint nodes x_i in the interval $[a, b]$

$$a = x_0 < x_1 < \ldots < x_n = b$$

with the corresponding node values $y_0, y_1, \ldots, y_n$. Find a function s that meets the following conditions:

(i) s interpolates the given points: $s(x_i) = y_i$ with $i = 1, \ldots, n$.

(ii) In the interval $[a, b]$: $s \in C^1$ at least.

(iii) Inside each subinterval $(x_i, x_{i+1}), i = 0, 1, \ldots, n-1$: $s \in C^4$. At each node x_i both the left-hand limit and the right-hand limit of the derivative exist but are not necessarily identical.

(iv) s minimizes the functional

$$J(s) := \frac{1}{2} \int_a^b s''(x)^2 dx. \tag{1.28}$$

It can be shown that there exists a unique function s meeting these conditions and on each subinterval $[x_i, x_{i+1}]$ this function is represented by a cubic polynom [SK06].

The minimization of the functional (1.28) leads to additional boundary conditions which are not suitable for all kinds of applications. These additional boundary conditions can be avoided by using *B-Splines* as the basis of the interpolation or approximation problem.

B-Splines In contrast to the previously described cubic splines, B-splines are not directly defined by a set of nodes but on a set of *knots* t_i, where several successive knots may have the same value. This multiplicity of knots, together with the order of the polynomial representing the spline function in the related interval, defines the smoothness of the resulting curve in the corresponding node.

For the knots t_i the B-spline of order 0 is defined by

$$B_{i,0}(x) = \begin{cases} 1 & \text{if } t_i \leq x < t_{i+1} \\ 1 & \text{if } x = t_{i+1} = x_n \\ 0 & \text{else.} \end{cases}$$

For multiple nodes $t_i = t_{i+1}$ the B-spline results to $B_{i,0}(x) \equiv 0$. B-splines of higher order k are defined recursively:

$$B_{i,k}(x) = w_{i,k}(x)\ B_{i,k-1}(x) + (1 - w_{i+1,k}(x))\ B_{i+1,k-1}(x)$$

with

$$w_{i,k}(x) = \begin{cases} \frac{x - t_i}{t_{i+k} - t_i} & \text{if } t_i < t_{i+k} \\ 0 & \text{else.} \end{cases}$$

If t_j is an m-fold inner knot, the B-spline is C^{k-m} at t_j [SK06].

The interpolation of a given set of nodes x_i by a spline function $s(x)$ can be realised by a linear combination of B-splines:

$$\sum_{k=0}^{n} \alpha_k B_k(x).$$

Finally, the general interpolation problem for given nodes x_i and node values y_i can be described by finding the coefficients $\alpha_i, i = 0, \ldots, n$ to fulfil

$$\sum_{i=0}^{n} \alpha_i B_i(x_k) = y_k, \quad k = 0, 1, \ldots, n$$

and thus by solving the linear equation system $B\alpha = y$.

A detailed introduction to B-splines can be found in [dB01].

In the context of this thesis splines are used in Section 3.3.2 for a smooth course of the primitives of the protein secondary-structure-based cartoon representation.

1.2.2 Differential Equation Systems

Ordinary differential equations (ODE) are equations involving a function together with its derivatives up to a certain order n:

$$F\left(x, y, y', \ldots, y^{(n)}\right) = 0, \tag{1.29}$$

where y is a function of x, $y(x) \in C^n$, y' the first derivative dy/dx and $y^{(n)}$ the n-th derivative $d^n y/dx^n$. Obviously, the first order ordinary differential equations are the most simple form and can be written as $y'(x) = f(x, y(x))$, or simply

$$y' = f(x, y). \tag{1.30}$$

The representation (1.29) is called *implicit* and (1.30) *explicit* differential equation.

Every explicit n-th order ODE can be transferred to an equivalent system of first order ODEs

$$z' = \begin{pmatrix} z_1' \\ \vdots \\ z_{n-1}' \\ z_n' \end{pmatrix} = \begin{pmatrix} z_2 \\ \vdots \\ z_n \\ f(x, z_1, z_2, \ldots, z_n) \end{pmatrix} \tag{1.31}$$

by introducing auxiliary functions

$$\begin{aligned} z_1(x) &:= y(x) \\ z_2(x) &:= y'(x) \\ &\vdots \\ z_n(x) &:= y^{n-1}(x). \end{aligned} \tag{1.32}$$

In general, (1.29) – and thus (1.30) – has an infinite number of solutions which can be reduced by the definition of additional constraints. A very common example are the *initial value problems* where a solution y to (1.30) is searched with y satisfying an initial condition

$$y(x_0) = y_0 \tag{1.33}$$

for given x_0 and y_0. For the ODE system (1.29) the initial condition is defined accordingly by

$$y(x_0) = y_0 = \begin{pmatrix} y_{10} \\ \vdots \\ y_{n0} \end{pmatrix}. \tag{1.34}$$

Concerning n-th order ODEs the initial value problem is defined by finding a function $y(x) \in C^n$ which satisfies (1.29) as well as the initial condition

$$y^{(i)}(x_0) = y_{i0} \tag{1.35}$$

for all $i = 0, 1, \ldots, n-1$.

Besides the initial value problem, the *boundary value problems* are very common, where $y(x)$ is required to satisfy the boundary condition

$$r(y(a), y(b)) = 0$$

with two different values $a \neq b$ and a vector

$$r(u, v) := \begin{pmatrix} r_1(u_1, \ldots, u_m, v_1, \ldots, v_m) \\ \vdots \\ r_m(u_1, \ldots, u_m, v_1, \ldots, v_m) \end{pmatrix}$$

of m given functions r_i of $2m$ variables $u_1, \ldots, u_m, v_1, \ldots, v_m$.

First, we will focus on the initial value problems. Usually no analytic solution to the initial value problem can be determined. Thus, a numerical solution is calculated by finding approximative values $\eta_i := \eta(x_i)$ for the exact ones $y_i = y(x_i)$ at certain designated abscissae x_i. These designated x-coordinates x_i are often chosen equidistant in an interval $[a, b]$: $x_i = x_0 + ih$ with $i = 1, \ldots, N$ and $h = (b-a)/N$. Therefore, the approximative values η_i are sometimes written by $\eta(x_i; h)$ to emphasise their dependency on the *step-size* h. One significant problem then is the question if and how fast $\eta(x; (x - x_0)/N)$ converges towards $y(x)$ for $N \to \infty$, i.e. $h \to 0$.

There are several different approaches to this problem, a short overview will be given in the following. The prerequisites for the existence of a unique solution are assumed to be satisfied. For further details about these prerequisites refer to [SB78].

The easiest way to solve the initial value problem

$$\begin{aligned} y'(x) &= f(x, y(x)) \\ y(x_0) &= y_0 \end{aligned} \tag{1.36}$$

is the dicretization of the differential equation. As $f(x, y(x))$ is the gradient of the exact solution $y(x)$ of (1.36), the forward difference can be applied, for $h \neq 0$ leading to

$$\frac{y(x+h) - y(x)}{h} \approx f(x, y(x))$$

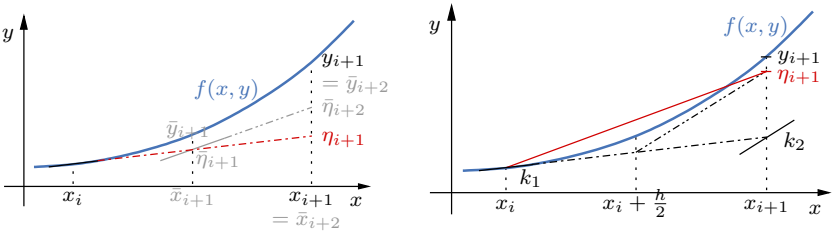
or in other terms

$$y(x+h) \approx y(x) + h\, f(x, y(x)). \tag{1.37}$$

Using a non-vanishing step-size $h \neq 0$ the approximative values $\eta_i \approx y_i = y(x_i)$ can be calculated by

$$\begin{aligned} \eta_0 &:= y_0 \\ \eta_{i+1} &:= \eta_i + h\, f(x_i, \eta_i), \quad i = 0, 1, 2, \ldots \end{aligned} \tag{1.38}$$

with $x_{i+1} := x_i + h$. This *explicit Euler method* is a typical one-step method – as the values η_i only depend on the preceding values η_{i-1} – and approximates the exact solution $y(x)$ by a polygonal strip, illustrated in Figure 1.13(a).



(a) First-order Euler method (1.38) with two different step-sizes

(b) Second-order Heun method with intermediate steps

Figure 1.13: Geometric comparison between (a) Euler and (b) Heun method. In (a) an alternative Euler approximation with halved step-size is additionally marked in grey.

One-step methods have the general form $\eta_{i+1} = \eta_i + h_i\, f_h(x_i, \eta_i, x_{i+1}, \eta_{i+1})$ where f_h is said to be the *method function*.

The comparison of the Euler method with Taylor series

$$\begin{aligned} y'(x_i) &= \frac{y(x_{i+1}) - y(x_i)}{h} + O(h) \\ y(x_{i+1}) = y(x_i + h) &= y_i + f(x_i, y_i)h + O(h^2) \end{aligned}$$

gives the local discretization error $\|\eta_i - y(x_i)\|$ of the method having the order of magnitude of $O(h)$. Thus, the explicit Euler method is said to be *of order* $p = 1$.

Obviously, the accuracy can be increased by either decreasing the step-size h (within a certain range) or by applying higher-order terms of the Taylor series and thus achieving higher-order methods.

The methodical derivation of higher-order one-step methods was primarily done by Runge [Run95] and Kutta [Kut01] who developed special third- and fourth-order methods. They imply a method function $f_h(x, y)$ which is a linear combination of n functions $k_j(x, y)$, i.e. the slope is a weighted average

of slopes, and the methods are therefore referred to as *Runge-Kutta method of n stages*. The generic form [SK06] of the Runge-Kutta (RK) method is

$$\eta_{i+1} = \eta_i + h \sum_{j=1}^{n} \gamma_j \, k_j(x_i, \eta_i)$$

$$k_j(x_i, \eta_i) = f\left(x + \alpha_j h,\ \eta_i + h \sum_{l=1}^{n} \beta_{jl} \, k_l(x_i, y_i)\right).$$

The only explicit Runge-Kutta method with only one stage, and, thus, the simplest one, is the previously described Euler method (1.38). This method has been extended to a two-stage second-order RK method by Heun [Heu00] and therefore named *modified Euler* or *Heun method*. There, the slope is the average of the slope in x_i and the slope in $x_i + h$, as drafted in Figure 1.13(b).

In contrast, the most commonly used Runge-Kutta method is the 4-th order method illustrated in Figure 1.14, often simply referred to as *the* Runge-Kutta method or *RK4*:

$$\begin{aligned} \eta_{i+1} &= \eta_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ \text{with} \quad k_1 &:= f(x_i, \eta_i) \\ k_2 &:= f\left(x_i + \frac{h}{2},\ \eta_i + \frac{h}{2} k_1\right) \\ k_3 &:= f\left(x_i + \frac{h}{2},\ \eta_i + \frac{h}{2} k_2\right) \\ k_4 &:= f(x_i + h,\ \eta_i + h k_3) \end{aligned} \tag{1.39}$$

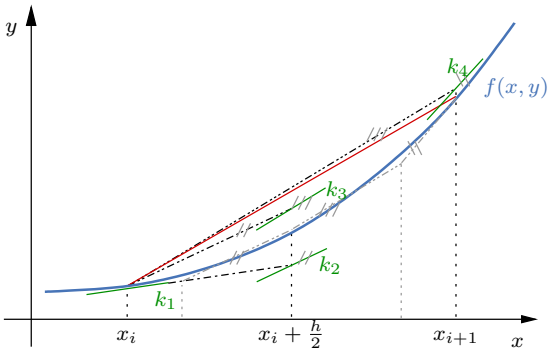


Figure 1.14: Geometric interpretation of the classical 4th order Runge-Kutta method RK4 (1.39). The final slope $\frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$ is marked in red.

The inevitable discretization error can be reduced by applying an adaptive step-size routine, i.e. step-size control. As the global error is usually hard to calculate and often overestimates the real error, an estimate of the local error is commonly used instead. Beside other means [SK06] *embedded* methods can be used, following the idea of Fehlberg [Feh68]. There, two methods of different order but with identical step-size are calculated in parallel, where the coefficients of shared stages are identical and their shared values k_j have to be computed only once. Accordingly, a 4th order Runge-Kutta method can be embedded in a 5th order Runge-Kutta method which is of 6 stages though.

$$\begin{aligned}
k_1 &= f(x_i, \eta_i) \\
k_2 &= f\left(x_i + \frac{2}{9}h, \eta_i + \frac{2}{9}hk_1\right) \\
k_3 &= f\left(x_i + \frac{1}{3}h, \eta_i + \frac{1}{12}hk_1 + \frac{1}{4}hk_2\right) \\
k_4 &= f\left(x_i + \frac{3}{4}h, \eta_i + \frac{69}{128}hk_1 - \frac{243}{128}hk_2 + \frac{135}{64}hk_3\right) \\
k_5 &= f\left(x_i + h, \eta_i - \frac{17}{12}hk_1 + \frac{27}{4}hk_2 - \frac{27}{5}hk_3 + \frac{16}{15}hk_4\right) \\
\eta_{i+1} &= \eta_i + h\left(\frac{1}{9}k_1 + \frac{9}{20}k_3 + \frac{16}{45}k_4 + \frac{1}{12}k_5\right)
\end{aligned} \tag{1.40}$$

This 4-stage method can be embedded with the extension to

$$\begin{aligned}
k_6 &= f\left(x_i + \frac{5}{6}h, \eta_i + \frac{65}{432}hk_1 - \frac{5}{16}hk_2 + \frac{13}{16}hk_3 + \frac{4}{27}hk_4 + \frac{5}{144}hk_5\right) \\
\bar{\eta}_{i+1} &= \eta_i + h\left(\frac{47}{450}k_1 + \frac{12}{25}k_3 + \frac{32}{225}k_4 + \frac{1}{30}k_5 + \frac{6}{25}k_6\right).
\end{aligned} \tag{1.41}$$

The resulting error estimation τ for (1.40) is the difference of the two approximative solutions $|\bar{\eta}_{i+1} - \eta_{i+1}|$

$$\tau(x_{i+1}, h) = \frac{h}{300}\left|-2k_1 + 9k_3 - 64k_4 - 15k_5 + 72k_6\right|. \tag{1.42}$$

Depending on τ, the decision can be taken whether to in- or decrease the step-size or to keep it constant for the next step. When updating the step-size for the next step, Stoer [SB78], for example, proposes a new step-size

$$h_{new} = h \sqrt[p+1]{\frac{\varepsilon}{|\bar{\eta}_{i+1} - \eta_{i+1}|}} \tag{1.43}$$

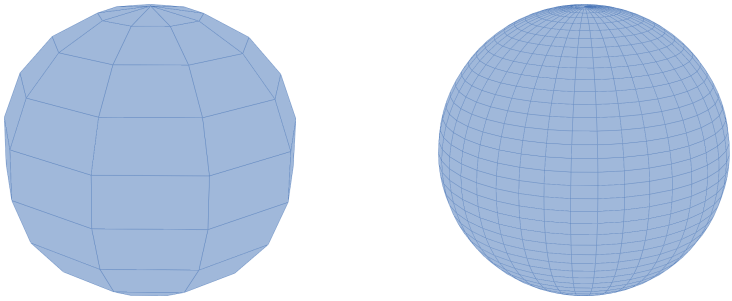
for a user-given error bound ε.

This approach – the embedded 4-th order Runge-Kutta method with adaptive step-size control – is widely used in practice because in many applications this

method is a good trade-off between accuracy and stability on one hand and computational effort on the other hand. Hence, this method is also used in the context of this thesis for mesh relaxation, described in Section 2.3.2. But in the end, the question which numerical method suits a problem best highly depends on the problem itself. For further details and theoretical fundamentals about numerical solvers for ODEs and ODE systems confer, for example, to [Col55, Lam73, SB78, SK06].

1.3 Mesh-based Geometry

Visualization approaches and techniques highly depend on the application and thus on the underlying data. Beside the distinction drawn between scattered data and grid-based data, the visualization methods can be classified by their geometric representation of surfaces: Mesh-based representations like triangulated surfaces versus topology-free representations such as raycasting of analytically defined surfaces. The backbone of the finite element method further detailed in Section 2.1, for example, is the discretization of the problem by approximating continuous quantities by a set of quantities at discrete points, i.e. by a subdivision of the structure or region of physical interest into smaller elements – the *finite elements*. Therefore, the mesh has a semantic meaning for the simulation analysis and a mesh-based visualization is not only acceptable but rather essential for this application – detailed in Section 2 – as deeper insight into the data can be gained from the provision of the mesh topology. On the other hand, the discretization of surfaces can be distracting if it has no semantic meaning, and the tesselation of the surfaces should be avoided if possible. A simple example is the tesselation of a sphere where poor discretizations change the silhouette tremendously as can be seen in Figure 1.15.



(a) Coarse tesselation of a sphere

(b) Fine tesselation of a sphere

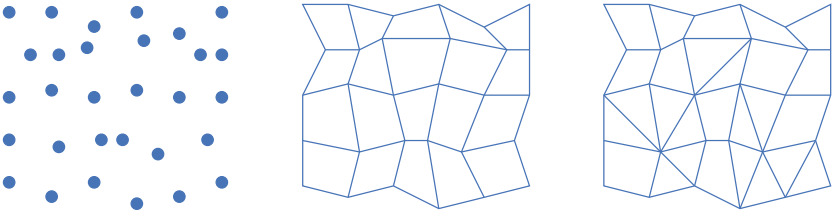
Figure 1.15: The discretization's influence on shape impression and silhouette.

An example where mesh-based rendering can be neglected to a large extent in favour of mesh-free geometry rendering is the visualization of molecular data – detailed in Section 3 using raycasting, previously presented in Section 1.1, page 17.

The remaining of this section deals with the basics of mesh-based geometry and visualization together with a short presentation of the discretization of the differential geometric properties of curves and surfaces previously discussed in Section 1.2.1 for the analytical case.

Conventionally, surfaces are discretised for rendering – mostly triangulated since this is the most general discretization. Depending on both the origin of the data and the application task various mesh types emerged, differing in the element type they are constructed from, such as triangles or *quads* (quadrilaterals), and in structure.

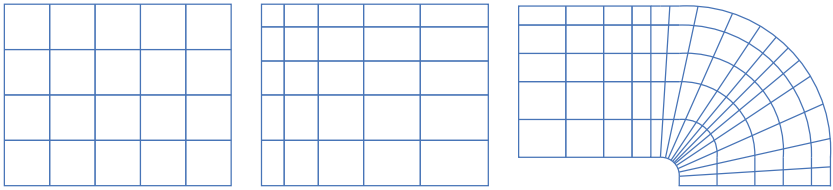
Figure 1.16 gives an overview over the principal data structures and grid types. Scattered data, as depicted in Figure 1.16(a), consists of discrete data points without topological relation between each other. But still n-dimensional data can be attached at each point. This data structure applies to the datasets used in molecular dynamics for instance. All grid types, in contrast, do have a topological cross-linking (*edge*) between the data points (*vertices*): Structured grids, as illustrated in Figure 1.16(b), have regular topology but may have irregular geometry; unstructured grids, such as given in Figure 1.16(c), have irregular topology and, thus, irregular geometry as well.



(a) Scattered data **(b)** Structured grid **(c)** Unstructured grid

Figure 1.16: Common data structures and grid types used in visualization

The main advantage using structured grids is that they are quite easy to handle since the vertices can be easily addressed by matrix-like indices. Structured grids can be characterised by the fact that every interior vertex has the same number of neighbours. Usually the grid cells (*elements*) are rectangles or parallelograms. The most simple type of structured grids thus is the Cartesian grids, exemplified in Figure 1.17(a), consisting of square-shaped



(a) Cartesian grid **(b)** Rectilinear grid **(c)** Curvilinear grid

Figure 1.17: Three different types of structured grids.

elements only. Rectilinear grids, as in Figure 1.17(b), in contrast, are constructed of rectangular-shaped elements but still show straight lines along the element borders. But also curvilinear bordered quads can be used, as depicted in Figure 1.17(c).

The major drawback of structured grids is their lack of flexibility with respect to local refinement and to adaptation of local surface features, especially if high distortion in element shapes is to be avoided. These problems are overcome by the unstructured grids where the elements are often triangles, less commonly quads, or in 3D tetrahedra, hexahedra or prisms. Unstructured grids can be either constructed from a unique element type or a combination of several types resulting in hybrid cell shapes. As a result of their flexibility in element shapes they can easily adapt to local surface features and adaptive refinement of the grid is possible. The drawback of unstructured grids is the fact that, in contrast to structured grids, the connectivity (topology) is not given implicitly but has to be stored and managed along with the data points. This leads to overhead in storage consumption and the addressing of the vertices is more complicated.

Unstructured grids are often derived from scattered data, such as point set surfaces, as they emerge, for example, from 3D scanners. This scattered data is commonly triangulated, since there are many possible triangulations for the same point set which vary in their element quality. Another tesselation method leading to unstructured grids is the isosurface extraction from volumetric data via the *Marching Cubes* algorithm [LC87], detailed in the following.

1.3.1 Marching Cubes

The Marching Cubes algorithm [LC87], computes an approximation of an isosurface in volumetric datasets and evolved to the standard geometry-based isosurface extraction method. The basic idea is that the data set is subdivided into a regular cuboidal grid, usually cubed. Then element by element (cube

by cube) is checked for intersection of its edges with the isosurface to be extracted – thus the name of the algorithm. Depending on the configuration of intersections the cubes and thus the emerging surface patches can be classified into lookup tables.

Each vertex of a cube can either be classified positive or negative with respect to the isosurface which is defined to be the union of the zero values. The original lookup table, depicted in 1.18, consists of 15 cases, covering all possible configurations of vertex classifications per cube – congruent cases excluded.

Nevertheless, this lookup table does not solve ambiguous cases that can arise

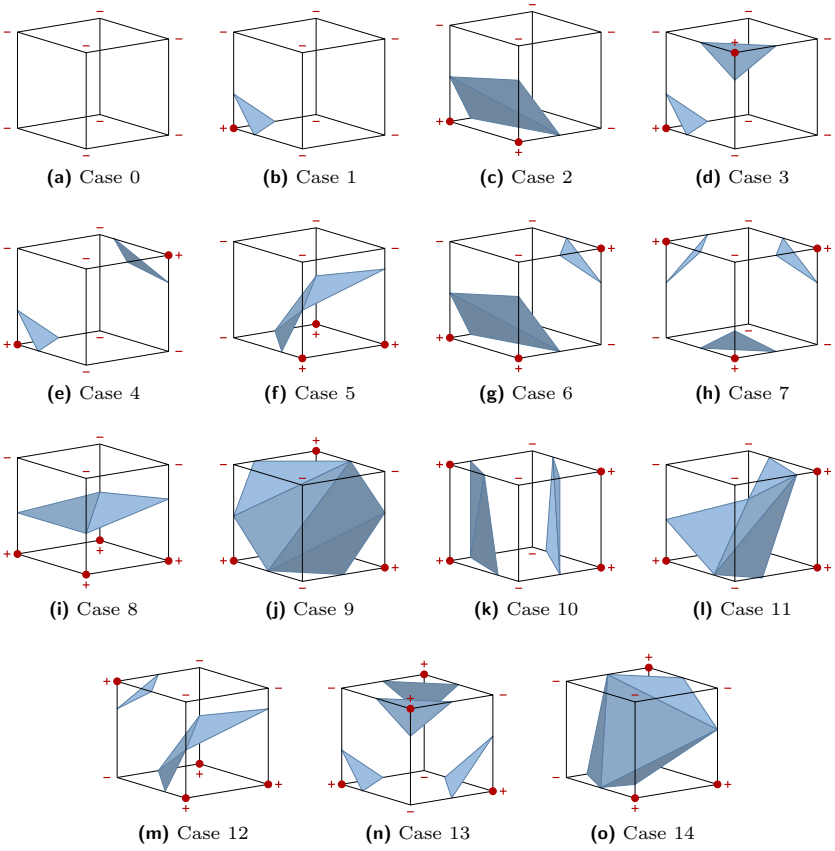


Figure 1.18: Original Marching Cubes lookup table as defined by [LC87]

due to connection inconsistencies between adjacent cubes. Thus, extensions and enhancements have been proposed to overcome this problem. Additionally, variants of the classical algorithm arose, like the *Discretized Marching Cubes* algorithm [MSS94], where the intersection point along an edge is not interpolated but always set to the edges midpoint resulting in a much lower amount of triangles. These extensions and enhancements will be further detailed in Section 2.2.3 when discussing the application case.

1.3.2 Differential Geometry Properties of Discretised Curves and Surfaces

The differential geometric properties of curve and surfaces discussed in preceding Section 1.2.1 are based on the differentiability of these geometric primitives and, thus, the mathematical definitions cannot be directly applied to discretised geometries such as meshed surfaces. Therefore, a specialised research area has emerged to close this gap: Discrete differential geometry. There, meshes are interpreted as piecewise linear approximation of such smooth and sufficiently often differentiable surfaces and the previously described properties are adapted and redefined – purely depending on the geometry instead of the parametrization. A survey on discrete differential operators is given in [Pet02], a nice overview is given e.g. in the Eurographics Tutorials [KBB+00] and [BPK+08]. The general idea is to define the discrete analogue to the differential properties as a spatial average over the local neighbourhood of the point on the mesh – usually a mesh vertex. Consequently, more smoothing is introduced the larger the neighbourhood is chosen. This smoothing leads to more stability for noisy data whereas small neighbourhoods provide better accuracy especially for fine-scale variations in clean datasets. To simplify matters but without loss of generality, discrete differential properties on triangle meshes are discussed in the following.

One of the basic problems is the definition of the surface normal vector N at a mesh vertex p. The most simple and most common way is to use a unit vector parallel to the arithmetic mean of the unit normal vectors N_i of the mesh faces that surround the vertex. But also weighted averages

$$N = \frac{\sum \omega_i N_i}{\| \sum \omega_i N_i \|}$$

may be used, where N_i denotes the unit normal vectors of the triangles adjacent to p and ω_i the corresponding weight. Various weight functions have been proposed, such as but not restricted to [Pet02] area-weighted and angle-weighted averages [MW00].

For the approximation of the various differential quantities and properties such as the principal curvatures and directions, mean and Gaussian curvature various approaches have been published. Several of these are based on local surface

fitting such as parabolic fitting (e.g. [Ham93]) and circular fitting (e.g. [CS92]). In other approaches differential geometric theorems and formulas such as the *angle deficit* or *Gauss-Bonnet* method are used to get the approximation of the curvatures (e.g. [MDSB03]). A short overview is given in [MW00]. In [SMS+03, MSR07] various curvature estimation methods are compared.

According to [MDSB03], the discrete form of the Gaussian curvature (1.27) in a vertex p can be defined by

$$K(p) = \frac{1}{A} \iint_A K \, dA \, .$$

When applying the Gauss-Bonnet Theorem 2, page 31, this leads to the simple equality

$$\iint_{A_M} K \, dA = 2\pi - \sum_i \theta_i \, ,$$

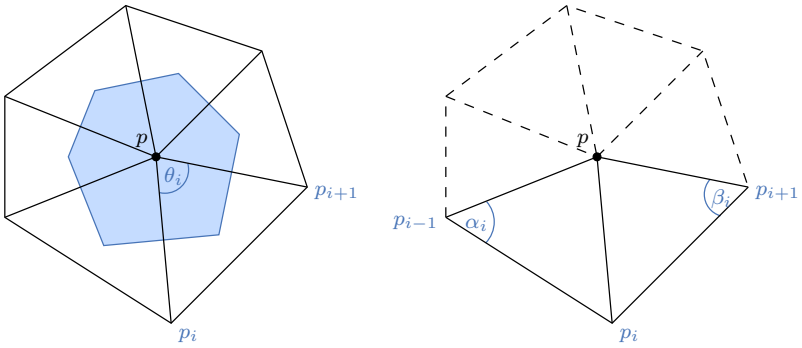
where θ_i is the inner angle of the i-th triangle at the vertex p, as illustrated in Figure 1.19(a), and finally one can get

$$K(p) = \frac{1}{A(p)} \left(2\pi - \sum_i \theta_i \right) \tag{1.44}$$

with $A(p)$ denoting the Voronoi area [MDSB03] around p (marked blue in Figure 1.19(a)), or

$$K(p) = 3 \, \frac{2\pi - \sum_i \theta_i}{\sum_i A_i}$$

with A_i being the area of the triangle corresponding to θ_i [KBB+00].



(a) Voronoi area [MDSB03] around p. **(b)** Angles for Laplace-Beltrami operator.

Figure 1.19: Neighbourhood of the vertex p.

The discretization of the mean curvature as well as the principle curvatures usually base on the Laplace-Beltrami operator

$$\Delta_S f = \operatorname{div}_S \nabla_S f \tag{1.45}$$

for a given function f on a manifold surface S. This evaluates the mean curvature normal by $\Delta_S x = -2Hn$ when applied to the coordinate function x [BPK+08]. The (current standard) discretization [BPK+08] of (1.45) is

$$\Delta_S f(p) := \frac{2}{A(p)} \sum_i (\cot\, \alpha_i + \cot\, \beta_i)(f(p) - f(p_i))$$

with $A(p)$ and α_i, β_i according to Figure 1.19(b). Using this together with the approximation (1.44) of the Gauss curvature the estimate of the mean curvature (1.26) can be computed as

$$H(p) = \frac{1}{2A(p)} \sum_i (\cot\, \alpha_i + \cot\, \beta_i)(p - p_i)$$

and the principal curvatures result to $\kappa_{1,2}(p) = H(p) \pm \sqrt{H(p)^2 - K(p)}$

These approximations presented are widely used for geometry processing operations such as surface smoothing, enhancement, quality checking, parametrization and shape modelling [BPK+08, MDSB03]. However, they entail disadvantages that make further processing or approximation improvement necessary. Details can be found e.g. in [BPK+08].

2 Mesh-based Geometries in the Application of Finite Element Modelling

Virtual prototyping more and more replaces real mock-ups and experiments in industrial product development such as in automotive industry. The fast increasing processing power of modern computers together with more and more efficient algorithms allows to calculate complex non-linear and highly dynamic processes – such as crash worthiness simulations or acoustics simulations of the interior of the passenger compartment – within short time.

First numerical full vehicle crash simulations were performed in the mid 1980's by many car manufacturers. Since then the virtual development for vehicle safety gained in importance as well as influence during the complete vehicle development process, driven by the increasing economic pressure in automotive industry and the related forceful development and enhancement of simulation methods and application tools [BFG05, Blu01]. So, expensive car prototypes are increasingly replaced by numerical simulations, to a large extent based on *finite element analysis* (FEA). This resulted in an approach combining the strengths of both, virtual and hardware based development. A corresponding finite element (FE) car model is exemplified in Figure 2.1.

Over the time the significance of this *computer-aided engineering* (CAE) gained essential importance in the vehicle development process, and, thus, not only

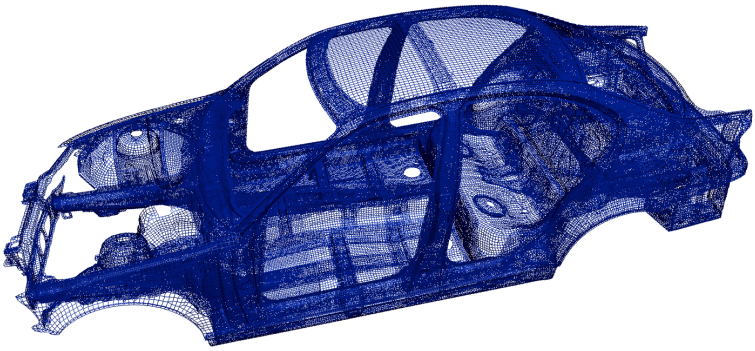


Figure 2.1: Finite element model of a car for structural mechanic simulations.

the impact and validity of the numerical simulation runs rose significantly, but also the range of application fields in the development process broadened increasingly. While in the beginning, only the deformation of single parts during the crash impact could be analysed and the importance of the simulations was often restricted to feasibility studies and to enhance the correlation with hardware tests, simulations are now well integrated into the overall standard vehicle development process and became an essential analysis method where relatively few physical hardware tests are finally used to confirm the results. Thus, costly and time-consuming physical hardware tests are, in large parts, substituted by simulations.

The broadened application fields not only span vehicle safety development with structural mechanic simulations, sensitivity analysis, especially for structural optimization, as well as the optimization of airbag sensing systems, but also vehicle dynamics and fluid mechanics, such as aerodynamic properties and air flow of heating, ventilation and air conditioning, vibration and acoustic properties, such as noise vibration harshness, are nowadays subject to simulation [Blu01, BFG05, MT08]. This became feasible as a result of the mutual benefit of increasingly powerful algorithms and soaring model sizes, which allow for more and more accurate simulation of the vehicle parts, together with the fast expansion of computational power. Model sizes raised from a thousand elements for the simulation of individual parts and about 100 000 or even less elements for complete vehicle models in the early days to model sizes of several million elements [Kra06] spread over hundred independently meshed parts forming the complete car model. Consequently, the need for powerful pre- and post-processing tools, tailored to the specific application fields escalated accordingly in order to cope with the broad range of tasks as well as the increasing amount of data.

The work presented in this chapter provides these tools specifically for various parts of the pre-processing pipeline of FE-based simulations.

The basic design of a new full car model is usually developed based on computer-aided design (CAD) models. In a subsequent processing step, these models have to be made suitable for the numerical simulation process, in particular they have to be meshed according to the prerequisites of FE-based simulations. Meanwhile, this meshing step can be automated to a large extent, but nevertheless some delicate parts and regions need to be treated manually in order to fulfil the prerequisites for further processing – still a very time-consuming step which altogether takes in the range of days. Consequently, if the geometry of one or more components is changed, for instance as a result of a simulation run in order to optimise the shape or the structural behaviour, this time-consuming meshing process has to be repeated once more – at least for the revised components. Especially in the early stage of development it is important to overcome this dead time and to provide interactive tools to the

simulation engineer making the model suitable for simulation again without undergoing the whole meshing process – within the bounds of possibility. The classical workflow together with the described optimization shortcut is depicted in Figure 2.2 and this is where the work presented in this chapter applies.

In the early stage of the model development it is important to be able to quickly generate new component variants or to edit existing ones interactively and to mesh them sufficiently suitable for the numerical simulation. In Section 2.2 this task is tackled by a method to remesh a model component based on a volumetrical representation. In combination with the mesh optimization methods detailed in Section 2.3, where the element shape is optimised with respect to the numerical simulation prerequisites – either after remeshing or after editing the model component –, this enables the simulation engineer to overcome the conventional remeshing. This method does not substitute the conventional remeshing in crucial stages of development, but is a time-saving, profitable and innovative tool during the early stages.

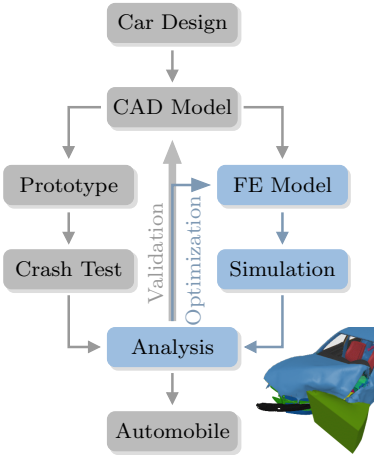


Figure 2.2: Basic car development process with the shortcut provided by the approaches presented in this chapter.

The previously briefly mentioned mesh optimization, detailed in Section 2.3, does not change the mesh topology, e.g. the node connectivity, but does optimise the node location on the modelled surface. This is a crucial process step with regard to the subsequent numerical simulation, since the element shape and properties, described in Section 2.1.2, play a decisive role with respect to reliability and stability of the numerical methods applied during simulation [Pam00, Kle07]. Both, the generation of mesh variants via volumetrical representation of the model and the optimization of the mesh elements, are tools which are developed as part of the comprehensive pre-processing tool and framework *scFEMod*[1].

In Section 2.4, a method for filling arbitrarily shaped holes in FE meshes is described, tailored for the application field of fluid simulation, e.g. acoustic simulation of the passenger compartment. There, water-tight models are necessary for the fluid simulation, which is usually not fulfilled for car models. Holes in the model occur from unmodelled components, such as wind- or rear screens, and missing component connections, as well as constructive cut-outs

[1] `http://www.science-computing.de/software/3d-visualisierung.html`

– but not from erroneous meshes, which is the common scenario for hole filling in visualization. Hence, the boundary of a hole can be spread over several independently meshed model components and special treatment was to be applied. Up to then, the majority of the numerous holes, according to the definition of a hole in the application context, had to be detected and processed by hand. In order to overcome this time-consuming process a detection and filling pipeline was developed and implemented as part of the pre- and post-processing tool *VisPER*, integrated into the multi-physics solver *PERMAS*[2], providing a powerful tool to speed up the simulation pre-processing step.

2.1 Introduction to Finite Element Methods

Finite element methods (FEM) are the most commonly used methods for calculating complex structures in construction analysis, civil and mechanical engineering, in automotive as well as in aerospace engineering. Usually the analysis is not the one posing the problem but the complex geometry of the underlying model, whereas beam structures or rectangular planes are far more easy to handle. Hence, the basic idea is to subdivide, or *discretise*, the complex model into numerous smaller and simple shaped elements which are analytically, or at least approximatively, manageable. On the one hand the structure, thus, loses its exact and continuous shape, on the other hand the more elements are used for discretization the better the original shape is approximated. So, the solution of the original problem is approximated by the union of the solutions on the small elements.

Generally speaking, FEM can be used in those applications where physical phenomena can be described by partial differential equations, depending on time or position. Examples are structural mechanics applications based on the equation of motion

$$M\ddot{u} + C\dot{u} + Ku = F(t) \tag{2.1a}$$

such as oscillation or fluid dynamics problems, problems like heat conduction described by the diffusion equation

$$C\dot{u} + Ku = F(t) \tag{2.1b}$$

and equilibrium problems like stiffness and stress problems specified by

$$Ku = F(t) \tag{2.1c}$$

where M is the mass matrix, C the damping matrix and K the stiffness matrix of the problem, $u(t)$ the displacement vector and $F(t)$ the applied forces vector.

[2] `http://www.intes.de/kategorie_permas/uebersicht`

These equations can be transformed into a differential equation system as detailed in Section 1.2.2. Furthermore, these problems can then be rewritten in a weak formulation in form of extremal or variational problems – the prerequisite for the finite element method to be applied [Sch91].

Historically the method dates back to the 1950's. The idea of getting a solution to a complicated problem by a finite subdivision and approximation, was already used by Archimedes of Syracuse (287 BC – 212 BC) to calculate the ratio of the circumference of a circle to its diameter (nowadays known as π), but related approaches in a narrower sense [Ode87] were presented e.g. in 1851 when Schellenbach [Sch51] used an approximation based on a triangle mesh to solve the Plateau problem. In 1941 Hrennikoff [Hre41] used a global approximation of a differential equation based on a sequence of local approximations over subdomains, and in 1943 Courant [Cou43] presented a piecewise linear approximation of the Dirichlet problem over a triangulated domain, both methods were already very similar to the one of finite elements. Subsequently, in the 1950's Argyris [Arg54, Arg55, Arg71] extended and generalised the approach, introduced its matrix formulation and applied it to thermal stress analysis problems in aircraft engineering. Independently, but roughly at the same time, a group at Berkley presented pioneering work [TMT56] on the method, applying it to the structure computation of an airplane wing – the first time use of surface elements and the FEM in a computer programme. The term *finite element method* was first used [CW99] by Clough [Clo60] in 1960. Thereupon, in the 1960's and 70's rapturous further developments took place and the method was widely used and applied to new application fields. The first definitive book [Zie67] was then published in 1967. But especially the high increase of computing power made the method manageable for further problems, making it to the probably most widely used method in virtually every conceivable area of engineering that makes use of models characterised by partial differential equations. Detailed historic overviews are given in [Ode87] and [CW99], an early overview over the method in [Zie70].

2.1.1 General Method of Finite Elements

The general method works as follows [Sch91]. The structure or domain is discretised by partitioning into a finite number of very small parts, called *elements*. These elements are defined by their endpoints or corners, called *nodes*. In some applications, like in a space truss, these elements are predetermined to a large extent, in others, like in field problems or elasto-mechanical tasks, the discretization has to be determined with care in order to get an appropriate approximation of the domain. In two-dimensional problems elements are usually chosen to be triangles, quads or even spherical triangles. The latter ones improve the approximation significantly but also increase the computational effort drastically. Volumetrical problems require a discretization into tetrahedral, cuboid or other volumetric elements.

In the next step, functions describing the problem are defined on the elements. These basis functions u may be polynomials of up to third degree for one-dimensional problems, for two-dimensional ones higher-order polynomials are applied as well as bilinear functions. The approach to be used depends on both the element shape and the problem type. In order to fulfil the usually given continuity requirements the function values or even derivatives, called *node variables* are defined at the element nodes. Hence, the basis functions result to a linear combination of functions N_i with the node variables as coefficients. For a two-dimensional element with p nodes and only function values u_i as node variables this would lead, for instance, to

$$u^{(e)}(x,y) = \sum_{i=1}^{p} u_i^{(e)} N_i^{(e)}(x,y) \tag{2.2}$$

where (e) denotes the dependency on the element. Since this description (2.2) should hold for any node variable $u_i^{(e)}$ the functions $N_i^{(e)}(x,y)$ need to satisfy the interpolation property to be 1 in node $P_i^{(e)}$ with the coordinates $\left(x_i^{(e)}, y_i^{(e)}\right)$ and to vanish in all other nodes of the element:

$$N_i^{(e)}\left(x_i^{(e)}, y_i^{(e)}\right) = \begin{cases} 1 & \text{for} \quad j = i \\ 0 & \text{for} \quad j \neq i \end{cases}$$

The domain is the union of all elements, so the global approach for $u(x,y)$, thus, is the combination of all $u^{(e)}(x,y)$ on all elements. Denoting all node variables subsequently by $1,\ldots,n$, $u(x,y)$ can be written as

$$u(x,y) = \sum_{k=1}^{n} u_k N_k(x,y) \tag{2.3}$$

where $N_k(x,y)$ denotes the union of those functions $N_i^{(e)}(x,y)$ that equal 1 in node P_k with node variable u_k. Therefore, $N_k(x,y)$ are non-vanishing only on a restricted subdomain.

The third step consists of integrating this approach into the problem describing Equations (2.1). These differential equations can either be reformulated equivalently as an extremal problem or as a variational problem [Sch91], depending on the underlying physical problem. For extremal problems such as stationary field problems or static elasto-mechanical problems this finally leads to an equation system of the general form [Sch91]

$$Su + d = 0$$

with a symmetric (and often even positive definite) matrix S called *stiffness matrix*, the vector of node variables u and the coefficient vector d of linear

terms. By contrast, oscillation problems lead to eigenvalue problems of the form [Sch91]

$$Su = \lambda Mu$$

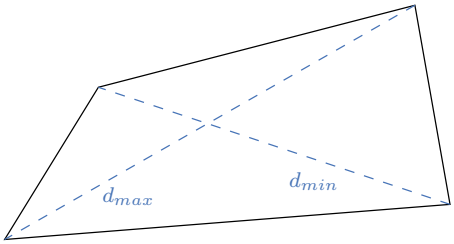
with symmetric matrices S and M where M is additionally positive definite. In both cases the difficulty lies in the formulation and properties of the stiffness matrix whose form also highly depends on the element numbering order. These matrix equations can then be solved numerically using standard approaches.

2.1.2 Mesh Property Prerequisites for Simulation

The element and mesh properties and, thus, the quality of the mesh have a big influence on the simulation results [Kle07]. In particular in the present application case of the car development process, usually thin shell elements are used. In this case, quad elements lead to better stability and accuracy of the simulation process and, thus, to better robustness. It is quite evident that the element size, relative to the model component size, has influence on the simulation accuracy, on the other hand tiny elements may increase the computation time excessively, so that a trade-off has to be found.

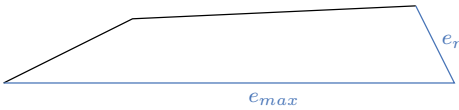
Finally, the quality of each element in itself is of crucial importance for computation. The perfect element, in terms of finite element simulation prerequisites, is a plane square and, if a triangle cannot be evaded, a equilateral triangle. Since this shape is not always feasible, degenerated elements may occur in the mesh during the pre-processing stage. Some typical degenerations that have to be overcome prior to simulation are detailed (values according to [Kle07]) and illustrated in the following. Highly distortedly shaped elements can cause bad results, deteriorated time steps or even divergence [Pam00].

Distortion: The ratio between the lengths of the shorter and the longer diagonal of a quad should be as close as possible to 1.0, where ratios down to 0.4 may still lead to acceptable results.



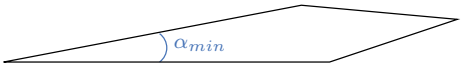
$\frac{d_{min}}{d_{max}} \approx 0.4$ to 1.0

Aspect ratio: The edge length aspect ratio should not drop below 0.5, whereas a value of 1.0 is the optimum.



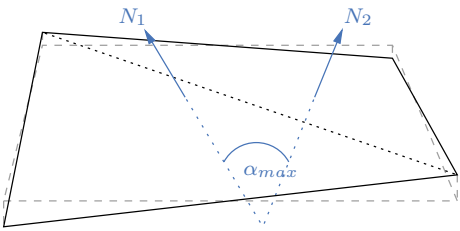
$\frac{e_{min}}{e_{max}} \approx 0.5$ to 1.0

Angle size: Acute angles cause stability problems and thus have to be refrained from. The minimum acceptable angle is approximately 10°. Extremely obtuse angles, on the other hand, lead to interpolation problems for the derivative and should be avoided likewise. In productive use angles in the range of 40° to 140° are strived for.



$\alpha_{min} \gtrsim 10°$

Warping: Since the three vertices of a triangle define a plane, triangles are always plane. In contrast, the vertices of a quad may be non-planar, leading to a deviation of the normals of the virtual triangles the quad can be split in. The angle between these normals must not exceed 45°. However, a maximum of 15° is the aim for productive simulations.



$\alpha_{max} \lesssim 45°$

These values are theoretical limits of the algorithms but should be even further limited for reliable results. Methods to optimise the observation of these limits are discussed in Section 2.3.

2.2 Generation of Mesh Variants via Volumetrical Representation

Usually the car parts are designed as analytical surfaces using computer aided design (CAD) and have to be transformed into an FE mesh for simulations in a subsequent step. For this purpose several meshing algorithms have been

developed but still a lot of expert knowledge and manual intervention is needed to adapt those meshes to fit the demanding prerequisites of a reliable numerical simulation. Significant improvement in simulation algorithms made it possible to use individually meshed model components instead of a consistent mesh for the whole model. This was a crucial progress which made it possible to provide more flexible tools for the engineer and thus to speed up the development cycle as now single car parts can be exchanged or varied without the necessity of a time-consuming remeshing of the whole model.

As development times for new car models are claimed to decrease more and more, the modelling of prototypes in the early phase of development is not always started from scratch but some model parts may be taken from earlier models and edited according to the new needs in simulations. Thus, engineers can gain important insight in the early stage of development in relatively short time to prevent major structural grievances in the model. In this phase of the model development sometimes large-scale editing of model parts is necessary, so that a remeshing of the affected part is needed. Because of the potential of coupling coarse-scale editing operations with the remeshing procedure, we developed a tool [BE05] to remesh an FE surface model – taking into account the needs for good FE meshes – via volumes: First, the surface is voxelised to a uniform distance volume and then a new quad mesh is generated via isosurface extraction with subsequent mesh optimization.

The mesh generated by this method will usually not fulfil the stringent conditions for simulations in late production stages but it is a fast tool for early stages to get rough estimates of the impact of the editing applied – a step of utmost importance in the development process.

2.2.1 Related Work on Voxelization and Isosurface Reconstruction

Voxelization is a common method in many application fields of computer science. Accordingly, there is a lot of work and publications about voxelization, often bound to special needs of the application they were developed for. An algorithm fitting all interests at the same time does not exist. Especially for closed surfaces, many, sometimes simple but powerful, algorithms have been invented – such as binary volumes, e.g. [Kau87, HYFK98], or distance volumes, e.g. [Gib98, VKK$^+$03], – some also taking into account special requirements like accuracy of surface details [Sra01, HLC$^+$01]. A nice overview of classical voxelization literature is given in [COK95].

In our application case the surfaces are not closed but bounded, which leads to new problems: How to handle the boundary and, in case of distance volumes, how to treat distances measured to the boundary. Using binary volumes, on the other hand, leads to problems with thin surfaces and the surface's thickness would have to be increased to get a hole-free voxelization. Additionally,

it would lead to double surfaces during reconstruction, as the surface between *inside* and *outside* would be extracted – which is of no use for the given purpose. So signed distance volumes turned out to be most promising though, since the zero isosurface can be extracted as a single surface. Another approach, presented in [KBSS01], is to directly save the points where voxel and surface intersect each other instead of storing a scalar distance value per voxel – an idea adopted in addition for the presented application case for handling the special cases of borders and feature lines, detailed in Section 2.2.3.

Consequently, there is also a lot of related work in the field of isosurface extraction from volume data. Starting from the fundamental *Marching Cubes* algorithm, introduced in [LC87] and detailed in Section 1.3.1, many variants of this method exist. The algorithm has been simplified to *Discretized Marching Cubes* in [MSS94] where each intersection of the marching cube with the isosurface is set to be in the middle of the respective cube's edge instead of the interpolated position along the edge. In addition, many extensions and improvements have been applied to the Marching Cubes algorithm in order to solve ambiguous cases, which can occur in the original algorithm, or maintain topological properties, e.g. proposed in [NH91, Che95, Nie03] and [LLVT03], geometrical properties like sharp edges, e.g. illustrated in [KBSS01], or to get smoother reconstructed surfaces such as presented in [LB03]. A newer approach [DSS+09] improves the resulting mesh quality by modifying the sampling grid the Marching Cubes algorithm operates on while preserving the mesh topology.

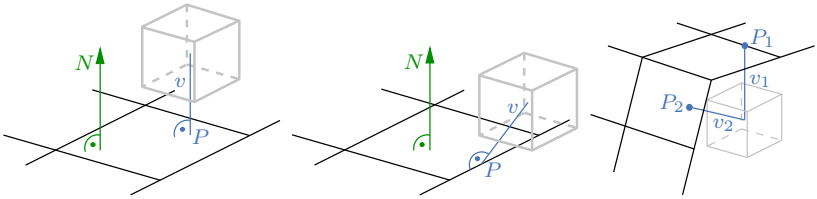
Marching Cubes and its derivatives usually generate triangle meshes. In FE simulations we need quad meshes or at least quad dominated meshes with only few triangles, since triangle meshes lead to unstable numerical simulation results in FEA. So a quad mesh has to be extracted from the volume, similar to the *Dual Marching Cubes* introduced in [Nie04] where the dual grid is used to enhance the triangle mesh. But still generating a mesh dual to the one generated by a Marching Cubes variant leads to new problems on bounded surfaces as data might be lost at the boundary, a task tackled in Section 2.2.3.

2.2.2 Voxelization of the Model

Generating a volume representation of the FE mesh is done by distance calculation. The voxel size (uniform in each direction respectively) depends on the elements' globally minimal edge length to make sure not to miss out small surface features. When the volume dimensions are calculated, the shortest Euclidean distance to the model surface is computed for each voxel. Due to bounding volumes, this calculation can be done quite fast. If the distance to the surface is larger than a specified threshold (e.g. more than the doubled diagonal length of the voxels), the voxel value is set to 255, the maximum

value codable in one byte, representing an *invalid* value that should not be taken into account for volume rendering or later isosurface extraction.

The point P on the surface that has the shortest distance to the voxel might have different properties depending on its location in respect of the present surrounding mesh, illustrated in Figure 2.3. The default case, Figure 2.3(a), is that P lies inside an element, whereas Figures 2.3(b) and (c) represent cases that have to be treated specially, detailed below.



(a) Closest point on surface lies inside an element

(b) Closest point on surface lies on mesh boundary

(c) Ambiguous case of closest point.

Figure 2.3: Possible locations of the closest point on the surface in respect to the voxel's position.

Ideally, P lies inside an element, as depicted in Figure 2.3(a), and the distance vector v is perpendicular to the element plane. For voxels in the neighbourhood of surface borders, the closest point P lies on an element edge or vertex which is part of the surface boundary, as shown in Figure 2.3(b). In this case the distance stored in the voxel would lead to wrong assumptions during volume rendering or surface reconstruction.

In order to avoid these problems, the voxel value is set to *invalid* and the voxels are skipped in case the angle enclosed by the corresponding distance vector v and the element normal is larger than a specified threshold ($v \cdot N > \varepsilon$). In the other cases P lies on an inner edge or an inner vertex and the distance vector is compared to the average of the neighbouring elements' normals accordingly.

Another problem is when the closest point is ambiguous, i.e. the same shortest distance from the voxel to the surface is measured to an edge as well as to an element, which can especially occur in the surrounding of surface feature lines. In that case, the element gets favoured. This scenario is illustrated in Figure 2.3(c).

A 2D outline of the signed distance volume is given in Figure 2.4. The red dashed line exemplifies the cut-off distance beyond which the voxel value is set to *invalid* for reconstruction. In Figure 2.5 an example of a resulting voxelised FE mesh is given, showing that this method preserves surface features such as small holes.

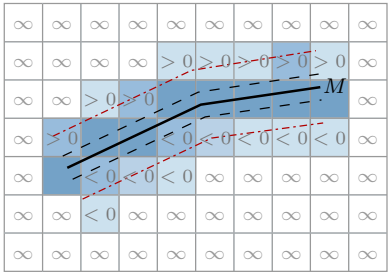


Figure 2.4: 2D outline of the signed distance volume. The different shades of blue represent the distances to the given surface M. The red line shows the cut-off distance.

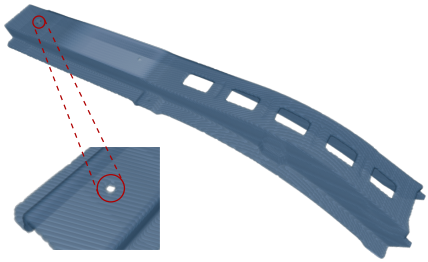


Figure 2.5: Volumetric representation of a resulting voxelised surface. Small surface features such as constructive holes are well preserved.

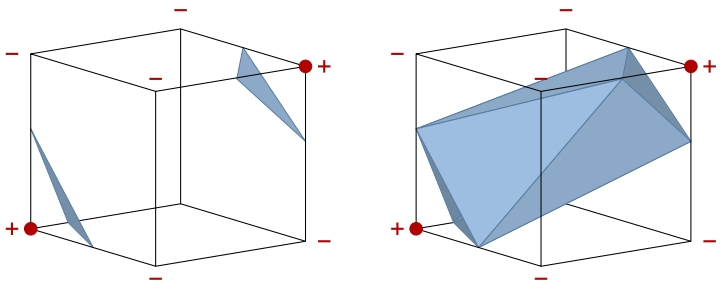
2.2.3 Surface Reconstruction

In order to retrieve a meshed surface, an isosurface has to be extracted from the volume. The Marching Cubes algorithm, previously described in Section 1.3.1, computes for each cube the zero points along the edges by linear interpolation between the values at the cell vertices. If one vertex value is negative and one positive, there has to be a zero point on the connecting edge or diagonal, which means an intersection of the cube with the surface. As we use signed distance fields to generate the volume, the zero isosurface is the surface to be reconstructed.

Isosurface Extraction

The classical Marching Cubes look-up table [LC87], previously depicted in Figure 1.18, contains some ambiguous configurations that may lead to topological problems like holes in the mesh. If e.g. the only two negative vertex values are situated on the vertices connected by an inner diagonal of the cube, the surface inside the cube may either separate or connect these two vertices, as depicted in Figure 2.6.

Choosing the wrong configuration would lead to problems when connecting the generated triangles to the ones of the neighbouring cubes, possibly causing cracks and discontinuities in the triangle mesh. The development of enhanced look-up tables taking into account and solving these ambiguities, e.g. in [Che95], allowed algorithms – such as [LLVT03] – preserving the original topology and getting rid of unintentional holes in the mesh. This algorithm, an enhanced version of the *Marching Cubes 33* [Che95], not only



(a) Surface separates marked vertices

(b) Surface connects marked vertices

Figure 2.6: Ambiguous cases in surface extraction: Two possible triangulations with the same configuration of vertex values.

takes into account the vertices' values, but also interpolates values at inner points or on surfaces to distinguish between configurations. This is important to ensure topological correctness. To provide consistent triangulations neighbouring configurations have to be considered additionally.

As avoiding cracks in the mesh is among the most important prerequisites for FEA, the meshing method described in this work is based on this algorithm. The connectivity and thus the triangulation within each cube is determined using the look-up table of [LLVT03]. But since a triangle mesh is not suitable for FE simulations, the triangle mesh is converted to a quad mesh in a subsequent step. This step is described in the following.

Quad Mesh Generation

In 2004, Nielson [Nie04] used the dual operator to smooth triangle meshes by applying the operator twice on the mesh. Since the previously described voxelization generates a uniform grid, the triangles generated by the Marching Cubes algorithm are always arranged in a cube and are connected to the triangles inside the adjacent cubes. Provisionally disregarding voxels on the mesh borders and only once applying the dual operator thus automatically leads to a quad mesh: As a first step we consider only cubes of the inner surface, thus not containing triangles of the surface's boundary. Imagine four cubes connected in one conjoint edge. Connecting the centres of these four cubes leads to a square. This method can now be applied to the triangulation obtained by the algorithm as described before. Considering connected triangles within one cube as a single polygon, we can connect the centre of the polygon (Q_i) with the ones of the neighbouring cubes, as depicted in Figure 2.7, and get quads instead of triangles.

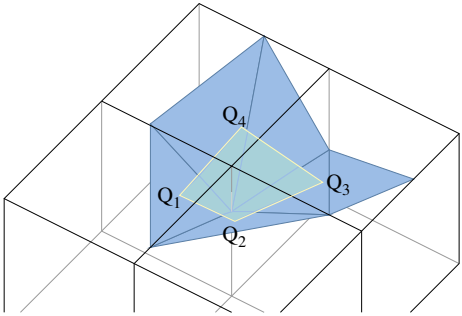


Figure 2.7: Construction of a quad element (yellow) dual to the triangle mesh (blue) generated by Marching Cubes.

Even if there is more than one polygon within a cube, the statement holds for each of these surfaces respectively. The only problems arise on the surface boundary as the quad mesh is slightly smaller than the triangle mesh, due to the duality construction. To avoid this shrinking of the reconstructed surface the borders are identified and polygons are added and subsequently split or merged to quads in order to retain the original surface boundaries.

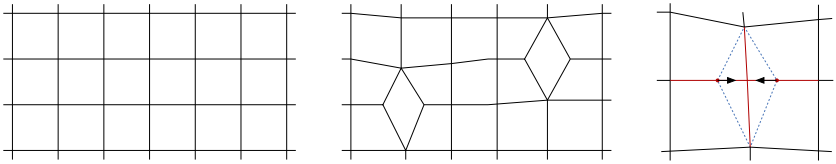
Feature Line Preservation

One of the main problems with isosurface extraction is the fact that sharp feature lines are being wiped out. To avoid this, these feature lines have to be treated explicitly. In the voxelization process, feature lines are detected depending on the angle between two neighbouring elements and the voxels which are intersected by the feature lines are marked. Since the dual grid, described on page 59, is used for the quad mesh generation, new vertices are located in the centre of the polygon extracted by Marching Cubes within each voxel. So, if the voxel is one of those originally intersected by a feature line, the new vertex is moved to the closest point on the extracted feature line. If there is a feature point within the voxel, e.g. a corner in the surface boundary, the new vertex is moved to this point. As the vertices are being moved only slightly within a single voxel the shape of the elements is only little affected but the feature lines are well preserved.

With these methods applied the resulting mesh already looks pretty appropriate for FE simulations. Nonetheless, the quad mesh still lacks some enhancements, described in the following, to fulfil the demands of FE Analysis.

2.2.4 Quad Element Adjustment

As explained in Section 2.1.2, elements in FE meshes have to satisfy special properties to be suitable for the simulation. One of the prerequisites is that the mesh should consist of quads, shaped as close to squares as possible, which are all oriented in the same direction, depicted in Figure 2.8(a) – ideally resulting to a structured mesh as defined in Section 1.3. As a trade-off, few triangles can be included in the mesh where needed.



(a) Well oriented elements

(b) Accidentally oriented elements

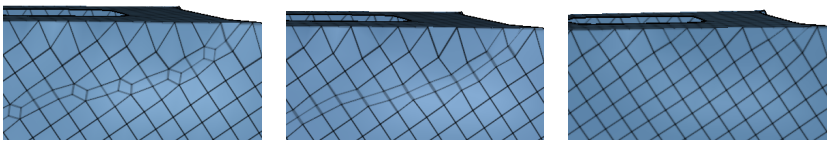
(c) Element correction

Figure 2.8: Correction of the orientation of quads in an FE mesh.

The mesh constructed using the dual grid approach described in Section 2.2.3 usually contains rhombic shaped quads or quads oriented diagonally to its neighbours as depicted in Figures 2.8(b) and 2.9(a). These artefacts lead to problems during numerical simulation and therefore should be removed.

As can be seen in the illustration (Figure 2.8), as well as in Figure 2.9(a), the vertices of a rhombic quad have a different degree than the ones around it: The vertices within the regular grid have even degree, the ones part of the rhombus have odd degree. Taking that fact as a reference to detect these quads, they can be removed by contracting the diagonal shared by the vertices with lower degree to one single vertex, which is illustrated in Figure 2.8(c) and exemplified in Figure 2.9(b). This leads to four equally oriented quads.

To further enhance the quads' shape and equalise the elements' size and inner angles, further mesh optimizations can be applied, detailed in the following section, leading to a mesh as shown in Figure 2.9(c) for the given example.



(a) Reconstructed mesh with rhombic quads

(b) Reconstructed mesh with corrected quads

(c) Finally optimised mesh

Figure 2.9: Adjustment of quad elements created by surface reconstruction.

2.2.5 Discussion on Quad Mesh Generation via Voxelization

The presented method for quad mesh generation via volumetrical representation was designed for a fast generation of a mesh variant as an intermediate step in FE mesh editing in the special application field of automotive prototyping. This approach does not lead to a perfect mesh for the final FE simulation but satisfies the prerequisites for the intermediate simulation steps during mesh editing and model optimization. Accordingly, the proposed technique proved its applicability in the designated application area and additionally found its way to productive use in pre-processing of steel casting simulations.

In recent years the generation of quad meshes gained in importance in the community and, thus, several new approaches were published. Besides the parametric approach, e.g. [KNP07, DBG+06, TACSD06] the generation of a quad (or quad dominated) mesh from a given triangle mesh emerged, such as [LKH08, BZK09] both leading to impressive results.

2.3 Optimization of Finite Element Meshes

The quality of the finite element (FE) meshes is a crucial factor for reliable FE simulation results. An important aspect is the shape of the elements – predominantly quad elements and, where required, triangles. The ideal elements are a plane square, as detailed in Section 2.1.2, and a equilateral triangle accordingly. Usually this criterion should be satisfied for the original meshes generated from the CAD model, but gets commonly violated during pre-processing – for example by mesh editing as detailed in [RBE04] and [BRE04] or by the remeshing process previously described in Section 2.2. Another mesh quality aspect is warping – the problem that the vertices of a quad are not bound to lie on a common plane as vertices of a triangle inevitably do. This problem can even arise in the original mesh resulting from CAD, but usually gets introduced during mesh deformations [RBE04, BRE04] or the generation of quad meshes on curved surfaces such as described in the previous Section 2.2.

Each of these violations to the mesh criteria, detailed in Section 2.1.2, leads to huge problems during the numerical simulation – both stability problems, and, thus, the raise of the question of reliable results, and simulation run time increase. Therefore, these element problems have to be detected and the mesh has to be optimised allowing for the particular characteristics of the surface (and, thus, the mesh), e.g. constructive feature lines, in a pre-processing step.

The repair of warped quads by levelling out the quad vertices is described in the following Section 2.3.1 in more detail. The violation of the element shape criterion can, to a large extent, be fixed by *mesh relaxation* and is explained in Section 2.3.2.

In case of large-scale deformations and modifications the mesh usually cannot be fixed without remeshing at least a part of the mesh [BRE04] or even the whole mesh, for example by the method described in the previous chapter. But as already said, even in that case the mesh optimization should be applied subsequently because in general the remeshing does not guarantee optimally shaped elements.

2.3.1 Warping Removal

Compared to triangle meshes quad meshes on curved surfaces bear the risk of warping. This might occur already during meshing, as described in the previous Section 2.2 and in [BE05], or very common as a result of manipulations of the surface or elements as detailed in [RBE04, BRE04]. Due to the problems this causes in FE simulations warping has to be avoided and thus be removed from the FE mesh. So, in order to make the mesh suitable for simulation again, these erroneous elements have to be detected and unwarped.

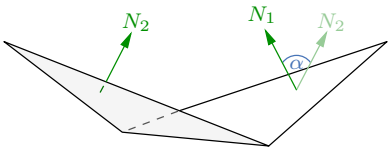


Figure 2.10: Warped quad element with $\cos\alpha > \varepsilon$.

In order to detect warping each quad is divided into two triangles whose normals N_1 and N_2 are compared as illustrated in Figure 2.10. If the enclosed angle α is above a specified threshold ε the element is identified as being warped, is the angle below the threshold the element is adequate for FE simulation. This can also be marked on the mesh to provide this important information about element quality to the user. The highlighted marking of warped elements in an FE mesh is exemplified in Figure 2.11 and works as follows: The diagonals of the quad are coloured yellow in case they are warped but still within the prerequisites for simulation and red in case they have to be fixed prior to simulation. A quad can be divided into two triangles in two different constellations, resulting in two different angles between the triangle normals. Thus, the warping also usually differs for these two possible triangulations of a reflect this fact in the representation the diagonals are marked with different thick lines: The thick line rep-

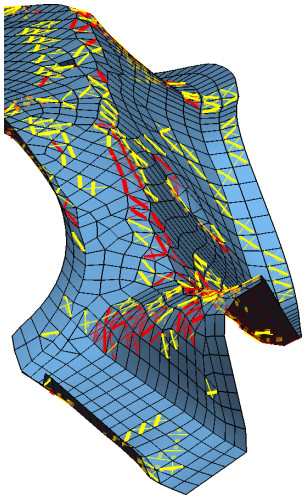


Figure 2.11: Highlighting of warped mesh elements. Red lines indicate severe warping, yellow minor warping acceptable for FE simulation.

resents the triangulation with higher warping, the thin line the triangulation with less warping.

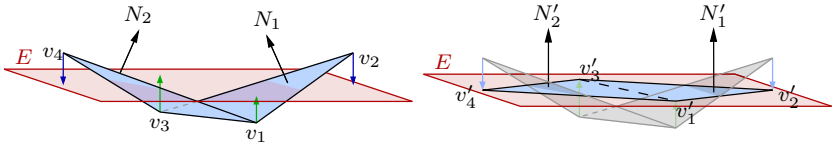
The unwarping itself is handled as follows: Both possible triangulations of a quad are examined and the direction corresponding to the wider angle α between the two normals N_1 and N_2 is the one with major warping and thus considered further on during warping removal. All elements are analysed with respect to warping and if the angle exceeds the given threshold the element is marked for unwarping. First, the linear smoothing plane E, depicted in red in Figure 2.12, through the middle point p of the four vertices v_i is calculated using the average of the two normals:

$$E: \; n_1 x + n_2 y + n_3 z - d = 0 \tag{2.4}$$

with $N = (n_1, n_2, n_3)^T = (N_1 + N_2)/\|N_1 + N_2\|$, $d = \sum n_i p_i$ and $p = (p_1, p_2, p_3) = \sum v_i$. Then the vertices' new coordinates v_i' are calculated by projecting the original vertices v_i onto this plane E:

$$v_i' = v_i + t\,N \tag{2.5}$$

with $t = (d - N \cdot v_i)/\|N\|$. This procedure is shown in Figure 2.12.

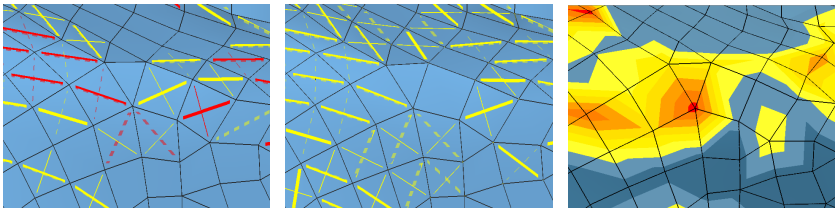


(a) Warped quad **(b)** Unwarped quad

Figure 2.12: Vertices of warped quads are projected into the linear smoothing plane in order to remove warping.

As a consequence neighbouring elements get affected by this vertex treatment. Therefore, the new coordinates are set no earlier than all elements are checked and the new coordinates are then set to the average of the ones calculated for each vertex respectively. This calculation is repeated iteratively until no element is warped more than allowed by the specified angle threshold.

Since warping is closely related to the vertex coordinates of a quad the warping removal inevitably affects the surface shape. On the other hand the vertices are only moved minimally to the smoothing plane of the neighbouring vertices. Therefore, the change on the surface shape is automatically kept to a minimum by this procedure which can be seen in Figure 2.13, where the Euclidean distance Δ between the unwarped mesh and the original one is depicted in the range of $\Delta < 0.01$ mm coloured yellow to red for $\Delta \geq 1$ mm – with quads of an average edge length of 10–20 mm. The run of surface feature lines and other characteristics is barely affected because these characteristics run along element edges and the vertices are only moved along the surface normal.



(a) Original mesh with marked warping

(b) Mesh after unwarping severely warped elements

(c) Euclidean distance between warped and unwarped surface

Figure 2.13: Warping removal process: Warped elements can be highlighted – as exemplified in (a) and (b) – and warping can be removed. The resulting deviation from the original surface is kept to a minimum and can be indicated (c) by distance mapping. The Euclidean distance Δ is colour-coded from yellow $\Delta < 0.01$ mm to red for $\Delta \geq 1$ mm.

2.3.2 Mesh Relaxation

The above explained removal of warping improves the mesh quality considerably. Nevertheless, the element shape has to be taken further care of. As already mentioned before the ideal FE element, with regard to the simulation, is a square. Usually this shape cannot be achieved for the majority of the elements but they should be ensured to be shaped as close as possible to a square. This can be achieved by applying a *mesh relaxation* to the mesh. In that process, the mesh retains its topology, i.e. particularly the mesh connectivity stays unchanged, but the vertex coordinates are adjusted to attain optimised element shapes.

The most common relaxation method in the literature is Laplacian smoothing, a simple recursive method where the vertices are directly adjusted depending on their adjacent vertices they have a common edge with. This method has been improved and extended by several authors to fit some special needs, e.g. [ZZHW91, BS91]. Another technique is the optimization-based smoothing where the vertices are moved depending on the minimization of a special distortion metric. Also combinations of these techniques were presented, e.g. [Fre97]. In [ABE97, CTS98] a more detailed overview over the various methods is given. Another approach is physically-based smoothing as presented e.g. in [LMZ86] where the edges in the mesh are represented by springs and the forces applied at the vertices depend on the ratio between the desired and the actual length of the edge. This approach is quite similar to the one developed in this work. As we have to deal with both quadrilaterals and triangles, we also have to take care of the diagonals in quadrilaterals: In order to avoid

parallelograms, springs are added along the diagonals of quadrilaterals – unlike [LMZ86] – but their desired length is adapted to be $\sqrt{2}$ times longer than the edges of the element.

A nice overview over various smoothing methods, especially applicable for meshes on scanned data, is detailed in [BPK+08] – based on spectral analysis as well as diffusion flow and energy minimization, just to mention few.

Most smoothing and relaxation algorithms have been developed for smoothly curved or even closed surfaces, but in the case of FE meshes the surface is bounded and often contains holes or constructive visible edges. Additionally, in order to preserve the significance of the numerical simulations, it is important that the readjusted vertices really lie on the surface originally modelled in CAD. This is why the most important improvement in the proposed relaxation technique is that surface features (boundaries and visible edges) as well as the shape of the surface itself are detected and preserved.

The relaxation model presented in this work is based on a spring-mass model, where the edges of the mesh are represented by springs. In order to prevent equilateral parallelograms being classified as well-shaped quadrilaterals additional springs are added along the diagonals of each quad in the mesh. An outline of the springs model applied to the mesh can be seen in Figure 2.14.

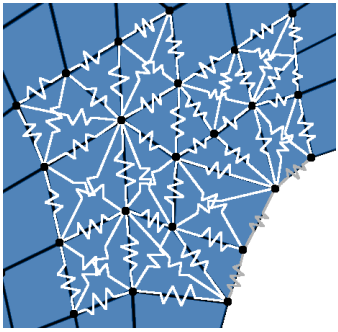


Figure 2.14: Outline of the spring-mass model used for relaxation.

Hence, the key problem is to find the differential equation that describes the spring model appropriately.

The general differential equation for a spring model can be written as

$$m\ddot{\xi}(t) = F_s(t,\xi) + F_d(t,\dot{\xi}) + F_{ext}(t), \quad (2.6)$$

where m denotes the mass of the object, ξ the object (vertex) coordinates, F_s the spring force, F_d the attenuation and F_{ext} the external force affecting the object. In the case of the here described relaxation model solid point-shaped particles – the mesh vertices –, thus $m := 1$, are connected by massless springs and no external forces are applied to the system. The remaining forces are the attenuation which can be defined by $F_d := -\gamma\dot{\xi}$, with an appropriate constant γ, and the spring force $F_s = c_1 \Delta l$, where c_1 is the corresponding spring constant and Δl the deviation of the current spring length from the spring rest length l_0 – caused by the force. In case of the FE mesh the rest length l_0 of the springs, and therefore the designated length of the edge represented by this spring, is set to the average length of the adjacent springs (edges and

element diagonals), where the diagonals are weighted with $1/\sqrt{2}$ according to the length of the diagonals in a square.

Let $v \in \mathbb{R}^3$ be the current vertex, then the spring rest length l_0 is calculated as

$$l_0 = \frac{\sum \|v_{uc,i} - v\|/\sqrt{2} + \sum \|v_{c,j} - v\|}{\#v_{uc,i} + \#v_{c,j}}$$

where $v_{uc,i} \in \mathbb{R}^3$ are the adjacent vertices *not* connected to v (i.e. the diagonals in a quad), and $v_{c,i} \in \mathbb{R}^3$ are those connected to v by a common edge in the mesh. Accordingly $\#v_{uc,i}$ denotes the number of adjacent vertices not connected to v and $\#v_{uc,i}$ the number of vertices connected to v. Consequently, the force $F_s \in \mathbb{R}^3$ applied in this vertex results to

$$\begin{aligned} F_s = &\sum (v_{uc,i} - v) \left(||v_{uc,i} - v|| - l_0/\sqrt{2} \right) / ||v_{uc,i} - v|| \\ &+ \sum (v_{c,j} - v) \left(||v_{c,j} - v|| - l_0 \right) / ||v_{c,j} - v|| \,. \end{aligned} \tag{2.7}$$

Taking this into consideration, the defining differential Equation (2.6) reduces to

$$\ddot{\xi} = c_s \, F_s - \gamma \dot{\xi}, \tag{2.8}$$

According to Section 1.2.2, where ordinary differential equations are discussed in more detail, this second order ODE can be transformed into a system of two first order ODEs by substitution:

$$\zeta := \dot{\xi} \quad \text{and thus} \quad \dot{\zeta} = \ddot{\xi}$$

So, Equation (2.8) finally results to

$$\begin{aligned} \dot{\xi} &= \zeta \\ \dot{\zeta} &= c_1 F_s - \gamma \zeta \end{aligned} \tag{2.9}$$

with F_s defined by (2.7). Additionally, the current state of the mesh before relaxation, i.e. the fixed vertex positions, defines the initial conditions to the ODE system (2.9):

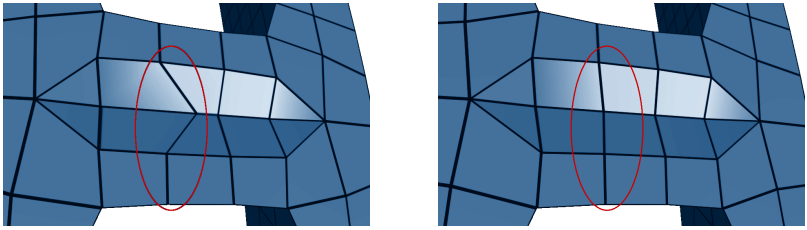
$$\xi_0 = v \quad \text{and} \quad \zeta_0 = 0 \tag{2.10}$$

This ODE system (2.9) with the initial conditions (2.10) can be solved according to the methods presented in Section 1.2.2. Simply solving this initial value problem without further constraints would indeed lead to well shaped elements, but the surface itself would get flattened and especially the features of the surface would be mostly wiped out, i.e. the boundary curve would change as well as edges within the surface. In order to avoid this some control mechanisms have to be introduced: After calculating the new position of each vertex by solving the ODE system, the new coordinates are checked in terms of their location relative to the original surface. If the new coordinates are too

far from the original surface, they are adjusted in a subsequent step. Since dealing with FE meshes, there is no parametric representation of the surface available but only the original mesh and thus the original surface as well as the surface feature lines have to be interpolated in terms of the coordinates of the neighbouring vertices in the previous step. In order to adjust the calculated vertices to the interpolated original surface inner vertices belonging to a continuously curved part of the surface get projected onto that interpolated original surface.

The question if a vertex is on a continuously curved surface or not is decided by calculating the angle between the normals of the adjacent elements adjoining in this vertex. If one or more of these angles are larger than a specified threshold angle δ the neighbouring edges *in line* with the processed one are also checked in order to detect visible feature lines on the surface. If at least one of these edges also matches this criterion, the feature line resulting from these edges is classified as a visible edge of the surface which is designated to be preserved. The described crease angle can be defined by the user whereas the default is set to 15°, the standard crease angle in the application field. The detected feature lines are then computed according to the interpolation of the surface using the original coordinates of the adjacent vertices involved and the current vertex is moved constrained to this visible edge – which is exemplified in Figure 2.15 – by projecting the newly computed vertex position onto this curve.

Nodes where two or even more of such feature lines adjoin are classified to be corners that do not get moved at all. With this method the characteristic features of the surface concerning later computations stay unchanged. This adjustment procedure can also be applied to the vertices being part of the surface boundary curve in order to ensure the protection of that prominent surface feature.



(a) Original mesh with poorly shaped elements

(b) Relaxed mesh with preserved feature line

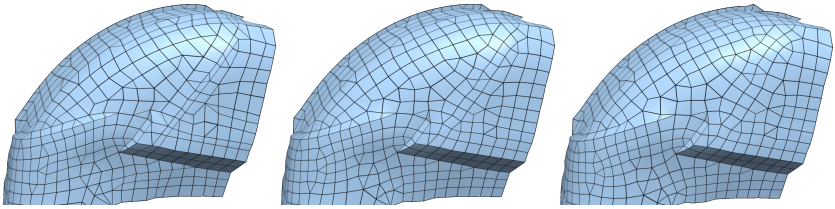
Figure 2.15: Preservation of surface features during mesh relaxation. Vertices which are identified to be part of a surface feature line were readjusted along that line solely.

Since the relaxation of one whole meshed car part, typically consisting of 500–1000 elements, by solving the ODE system with the given initial values and the previously described enhancing boundary conditions may take up to one minute and the aim was to give the engineer interactive tools to process the mesh, a restricted version of the relaxation method – producing similar results but with interactive performance – was developed alternatively.

In the restricted relaxation implementation the mesh is relaxed hierarchically starting at a vertex selected by the user, e.g. in the region of highest irregularity, instead of applying a spring-mass model as in the method described above. The displacement is calculated – similar to Laplacian smoothing – by adjusting each vertex *in the middle* of its neighbours, but with the same constraints on the vertex placement as explained in the method previously described. Starting with a user selected vertex, thus defining the centre of the relaxation area, vertices are moved in each iteration step successively. After adjusting the initial vertex with respect to its neighbours' position, the relaxation proceeds in circles around that centre vertex by adjusting the adjacent vertices of those moved in the previous round. This procedure is repeated until no new vertex, in respect of the set of vertices moved in the current or the previous round, gets displaced further than a user-defined threshold. This also finalises the current iteration step.

In the subsequent iteration step only those vertices are checked for displacement again which were moved themselves and whose neighbours were displaced in the last iteration step. With this procedure many vertices of the mesh can be ignored in most of the iteration steps, and thus a lot of computing time can be saved. The disadvantage of this method is the fact that the mesh is not completely relaxed after the iterations finished and the method still relocates vertices when called once more. But the experience with this tool shows that after the first or at least second time the new movements are very small.

The comparison of the results of both methods, exemplified in Figure 2.16, shows that even a single initial call to the hierarchical method already leads to appropriate results.



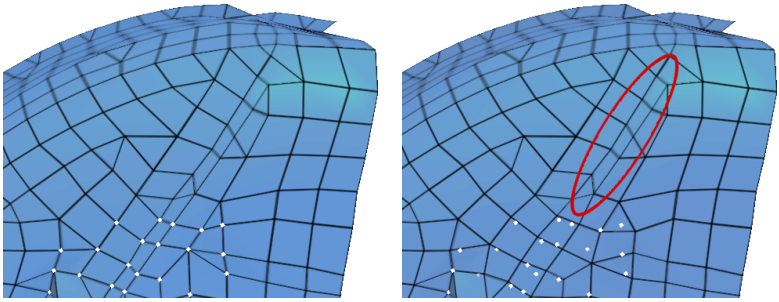
(a) Original mesh

(b) Hierarchically relaxed mesh after one iteration

(c) Mesh relaxed by solving the ODE system

Figure 2.16: Comparison of mesh relaxation methods.

In order to give more control to the user, it is also possible to restrict both relaxation methods to a group of selected vertices. Vertices can be interactively selected using a free-hand selection tool – detailed in [RBE04, BRE04] – provided by the framework. Non-selected vertices then do not get affected by the relaxation as can be seen in Figure 2.17: The selected vertices, highlighted with white octahedrons, were adjusted but all the others, such as those in the encircled area kept their position although the elements are shaped very badly.

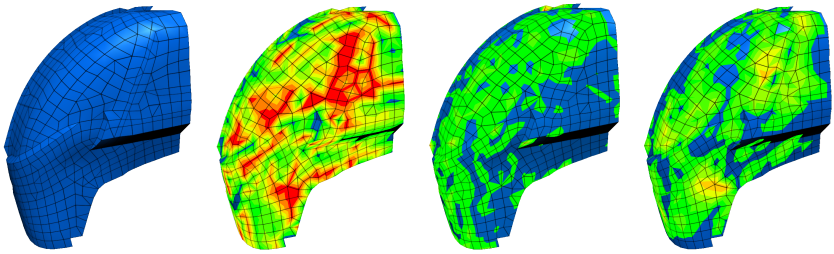


(a) Original mesh with vertices selected

(b) Mesh relaxation restricted to selection

Figure 2.17: Relaxation restricted to selected vertices: (a) vertex selection (white marked vertices); (b) relaxed region. Unselected vertices (encircled in red) are not adjusted.

The compliance of the optimised mesh with the original surface is crucial in FE applications. Therefore it is important to give detailed feedback to the user and thus the framework provides the possibility to evaluate the deviation between two surfaces, illustrated in Figure 2.18, e.g. of the relaxed mesh from the original surface, analogous to the evaluation of the unwarped surface previously shown in Figure 2.13(c). This evaluation is based on the calculation of the Euclidean distance between the vertices of the actual mesh and a reference mesh, e.g. the original one, and the subsequent texture-based mapping of this distance onto the meshed surface. This method was first presented in [SE00] and further developed in [SE01] particularly to represent the deviation in the FE mesh during crash simulations or to evaluate the influence of parameters in multiple simulation runs. The calculation is based on a bounding volume hierarchy [GLM96] and performs very well. Thus, the distance mapping can be invoked interactively. In that representation the distance within a user-defined range is mapped onto the surface – distances smaller than the lower limit are mapped to a transparent texture signaling the unmodified status of the surface shape, and values between the lower and the upper limit are mapped to a colour gradient representing the scale of distances. In Figure 2.18(c) this mapping is illustrated for the deviation between the original mesh and the relaxed mesh



(a) Original mesh **(b)** Vertex offset due to relaxation **(c)** Deviation of relaxed mesh from original surface **(d)** Hierarchical versus ODE-based relaxation

Figure 2.18: Comparison of mesh relaxation methods. (b) Displacement Δ of each vertex in the relaxed mesh compared to its position in the original mesh (given in (a)) with the colour gradient from green $\Delta \geq 1$ mm to red $\Delta \geq 5$ mm; (c) the distance between the original surface and the relaxed one using ODE-based relaxation mapped in the range of $\Delta \geq 0.05$ mm in green to $\Delta \geq 1$ mm in red; (d) the Euclidean distance Δ between the ODE-based relaxed mesh and the hierarchically relaxed mesh after one iteration step for the range $\Delta \geq 1$ mm (green) to $\Delta_r \geq 5$ mm (red).

using the ODE-based approach. The distances are mapped between 0.05 mm (green) up to over 1 mm (red). It can be seen from the picture that the relaxation process preserves the surface shape pretty well, due to the applied control mechanisms.

In summary it can be said that the two presented approaches for FE mesh optimization, the removal of warping and mesh relaxation, are powerful tools in FE simulation pre-processing. Since the compliance with interactive processing times was always kept in mind not only high quality results are attained but also instant feedback is provided which is essential for usability.

2.4 Filling Arbitrary Holes in Finite Element Models

Raised standards for the introduction of new industrial products require the close collaboration of different numerical simulation disciplines during virtual prototyping. One example is the interaction between structure and bounded or surrounding fluid achieved by the combination of computational fluid dynamics and structural dynamics simulation. Therefore, the various properties of a mixed CAE model need to be represented by adequate finite elements and their interface areas have to be coupled. Some application fields of such models are ship simulation, acoustic analysis of car parts or a whole car passenger

compartment. The corresponding CAE model can be constructed based on an existing structure model where, in a pre-processing step, the fluid volume has to be meshed by 3D finite elements. Though, a tetrahedral mesher requires a watertight bounding structure which is generally not given since today's car body models, as depicted in Figure 2.19, consist of hundreds of independently meshed car body parts which are connected by special finite elements.

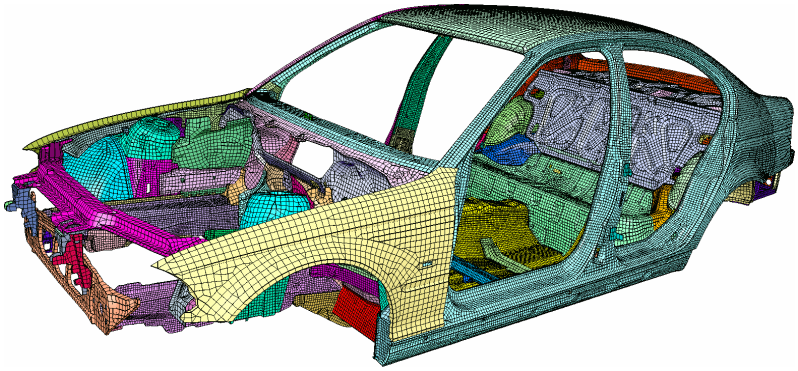


Figure 2.19: FE models generally consist of many independently meshed parts (coloured individually in the image) that provide arbitrary holes due to unmodelled parts as well as constructive cut-outs.

In the application considered here, a fluid mesh is generated using a hexahedral volume growing approach. The user specifies a seed point inside the structure model that surrounds the fluid volume. The volume mesh extends with a specified minimal cell size in all directions until it will be stopped by any intersecting structure as illustrated in Figure 2.20. Therefore, all holes in the surrounding structure that are at least as large as the minimal cell size need to be closed to prevent the generation of fluid cells outside the fluid volume. Until now those holes had to be found visually by the user or they were detected after the volume has grown to the outside. These pre-processing steps took several days. The approach presented here and originally published in [SBSE08] allows for cutting down on this processing time significantly by providing the potential for processing the majority of those holes automatically.

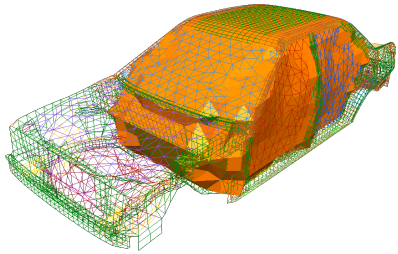


Figure 2.20: Closed FE model with volume mesh for fluid simulation.

Aside from ordinary holes inside a structural mesh, such as constructive cutouts, the CAE models also contain holes with complex boundaries, e.g. composed and defined by more than one continuous finite element mesh. These holes, for example, may result from missing model parts which are not available in the current structural model, e.g. the front shield or doors. Whereas the former type of holes can be detected automatically, the latter need to be defined interactively by the user. A method to define holes effectively will be detailed in the following and an algorithm will be presented that allows the meshing of detected and defined holes in consideration of requirements included by the application area in CAE modelling. A hole classification facilitates the meshing automation to a large extent which leads to a significant speed-up of the overall pre-processing step.

The work described in this chapter was published on the 17th International Meshing Roundtable [SBSE08] and was mainly implemented in the context of the diploma thesis of Andreas Schilling.

2.4.1 Related Work on Hole Filling in Meshes

Filling holes in meshes is a common problem in several fields of visualization. But in contrast to holes in CAE meshes – which are intended constructive cutouts or unmodelled parts and thus may spread over numerous independently meshed parts – these holes are usually erroneous parts in the model resulting from missing data in scanned geometry or polygonization artefacts during isosurface extraction. Hence, there is a wide range of methods to handle these holes, but mainly developed for closed, oriented 2-manifold meshes, which does not hold for our application case where the surrounding meshes are not necessarily oriented and 2-manifold and especially are not closed surfaces. However, some steps in the workflow require a watertight model in some parts and, therefore, these gaps and holes have to be closed.

The related algorithms can be split into two main categories: Volumetric approaches and geometry-based techniques. The volumetric methods split the region around a hole into inner part and outer part, usually based on the volumetric hull. Hence, the boundary layer defines the missing surface. In [DMGL02] the model is converted into signed distance volume representation and a diffusion process is applied. Thus, the distance function is extended through the volume until it spans all holes which can be meshed subsequently. Additional constraints have to be added to avoid unintended merging of surfaces. Tailored to repair meshes from range scans, the quality of the resulting surface reconstruction is not sufficient for CAE models.

A very promising algorithm to fill complex holes was presented in [PR05]. So-called *atomic volumes* are generated by space decomposition and determined to be either completely inside or outside the model and different topological

filling styles can be chosen by the user. Even though this is a very powerful method, it does not fit the special needs of CAE models consisting of numerous independently meshed parts. The same holds for the method presented in [SACO04]. There, the holes are filled using a context-based method: The characteristics of the surface are analysed and the hole is filled by copying parts of the mesh determined to fit the region around the hole. Although this gives impressive results for surfaces resulting from erroneous surface sampling, it is not applicable to our scenario, due to the constraints mentioned above.

Radial basis functions and the resulting implicit surface are used in [BPB06] and [CBC$^+$01]. This approach was extended in [CC08] to recover sharp feature lines in holey areas using an iterative and time-consuming error reduction step. Yoo [Yoo07] presents a method capable of handling large polygon models using domain decomposition to solve the problems locally and an implicit surface is defined going through the hole. The final mesh results from a combination of a smoothing and a refinement scheme together with Marching Cubes.

Altogether, volumetric methods do not fit the needs for filling holes in CAE models. In this context geometrical approaches are more promising.

In [BK97] an early method for closing gaps in CAE models is presented, but only stitching and filling of small holes is performed. Newer methods are far more flexible. In [WO03] an algorithm based on moving least squares is presented, reconstructing a locally smooth surface. This limitation makes the approach unsuitable for the more general shapes of holes in CAE models. An updated version of this method was presented in [WO07] but the algorithm itself stayed mainly unchanged. The algorithm presented in [PMV06] fills holes considering the curvature of the enclosing mesh.

Another geometrical approach is [TC04]. The algorithm uses an advancing front method to close the holes incrementally and moving least-squares (MLS) projection, evaluating the vertices' neighbourhood, is used to insert new triangles at the front. A pro of this method is that it is also applicable to noisy and unstructured data. By contrast, [Lie03] uses minimal surfaces (defined in Section 1.2.1), to fill arbitrary, not necessarily plane holes. Using dynamic programming, an ideal solution for the entire problem is computed. This approach holds the advantage of being applicable to arbitrary holes as well as being independent of the surrounding mesh and its characteristics. On the other hand, minimal surfaces do not always meet the requirements of FE models.

Nevertheless, the methods in [Lie03] and [TC04] turned up to be most promising to be adapted to our problem and, thus, were used as a base of the solution detailed in the following.

2.4.2 Preliminaries in the Application Case

In the first step of the complete hole filling process, the holes present in a model have to be identified. Before explaining the detection process for holes, the meaning of a hole in the scope of this work has to be clarified. Two classes of holes can be distinguished:

Topological holes: A topological hole is a hole that can be identified by topological properties, i.e. by searching a loop of boundary edges. A boundary edge in a mesh is an edge that is adjacent to only one element. Such topological holes are shown in Figure 2.21.

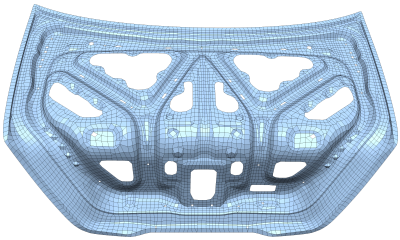
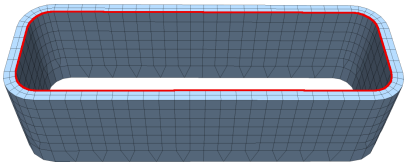


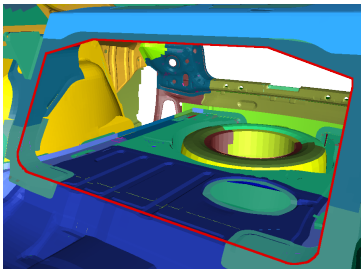
Figure 2.21: Topological holes enclosed by a continuous mesh.

Semantic holes: A semantic hole is a hole that, in general, cannot be identified by topological properties. This usually may occur if the hole spans over several independent meshes or if it is not made up by boundary edges, e.g. a cut-out in a thick-walled part, depicted in Figure 2.22(a). An example of a semantic hole bounded by several independently meshed parts and resulting from an unmodelled part is depicted in Figure 2.22(b).

Since topological holes can easily be found, the detection process can run automatically. They can be detected by searching a boundary edge and tracing



(a) Semantic hole in a thick-walled part.



(b) Semantic hole enclosed by several independent meshes.

Figure 2.22: In contrast to topological holes, semantic holes mostly do not provide a topological boundary. The various colours define a separate mesh each, the red curves mark the boundaries of the semantic holes to be closed.

connected boundary edges until a closed loop is found. This process is repeated until all edges of the mesh have been processed. Alternatively, the user can interactively select a seed boundary edge to start the detection. For bounded surfaces, the algorithm will classify the surface boundary as a hole, since it applies to the above definition of a hole. In that case either an automated test is applied or the user can interactively remove these falsely detected holes from the list of holes to be further processed.

Semantic holes, in contrast, can not be identified automatically in most cases. Holes of this class are e.g. unmeshed parts or constructive openings as illustrated in Figure 2.22(b). In the most complex cases these holes may appear by a certain configuration of several independent meshes without any boundary edges at all and without any connection information between meshes, so that only the user can identify the presence of a hole by looking at the complete model. Those semantic holes that span over several meshes but still consist of boundary edges though can be detected if information about the connectivity of the meshes is provided. Algorithms using sophisticated heuristics may also be able to detect some of the semantic holes that do not contain boundary edges, however they are error-prone and it is unsure whether they produce satisfactory results when being applied to a model consisting of hundreds of independent meshes or even whether they will terminate at all. Thus, the user needs the possibility to manually define a loop of edges that can be used as the hole boundary, regardless whether these edges are boundary edges of the mesh or not. This user-driven definition of holes is detailed in Section 2.4.3.

Once all holes have been identified, they have to be closed by an appropriate patching mesh. This mesh either may be connected with the surrounding mesh or may be defined as a new and independent mesh. This procedure of filling the holes is described in Section 2.4.4.

The following requirements shall hold for the hole patching meshing algorithm:

- Geometric features, such as constructive edges or design engineered curves, in the surrounding mesh have to be preserved.
- If neighbourhood information of boundary edges is missing, e.g. at gaps, the algorithm should still produce acceptable results.
- The algorithm must provide interactive influence on the shape of the patching mesh, e.g. whether it shall smoothly blend into the surrounding mesh or create a sharp edge.

2.4.3 Definition of Semantic Holes

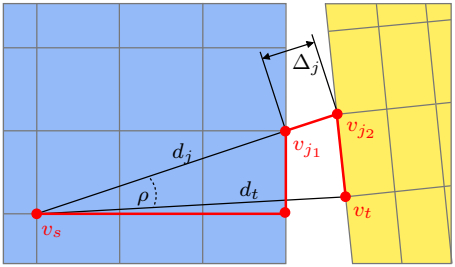
Topological holes and some of the semantic holes can be detected automatically, but, as described in the previous section, complex semantic holes require user interaction. A compromise between a moderate effort for the user and

decent speed of the definition process has been chosen to do so: The user has to select few nodes on the mesh, whereas the paths in between are calculated by a greedy algorithm, to run as straight and short as possible, while paths along feature lines or boundary edges are favoured. The most important functionality is the definition of semantic holes across several meshes, which is explained in more detail below.

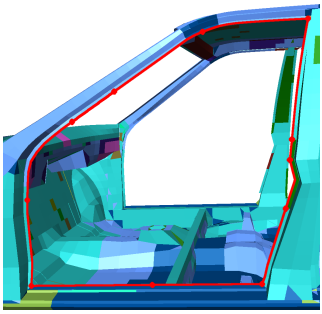
The path between two selected nodes that lie on the same mesh is chosen to run as straight and short as possible along element edges as mentioned above. In case the next selected target node v_t belongs to a different mesh an additional test is introduced and potential connecting node pairs are retrieved: Utilizing a bounding volume hierarchy, the minimum distance between the precedent selected path node v_s and the target mesh is calculated. If this distance is below a given threshold value, the nearest node on the target mesh is queried, again using the bounding volume hierarchy. This node v_{j_2} and the last node along the path on the current mesh v_{j_1} are then a potential connecting node pair between the two meshes.

To ensure an optimal connection, all valid connection candidates are stored and the one fitting best the users' expectations regarding the path run is chosen based on three criteria illustrated in Figure 2.23(a):

1. The angle ρ enclosed by the vector d_t (from the start node v_s to the target node v_t) and the vector d_j (from v_s to the first connecting node v_{j_1}) should be smallest possible, so that the resulting path runs as straight as possible towards the target node.



(a) Criteria for connection between meshes: 1. The angle ρ, 2. The distance $\|d_j\|$, and 3. The distance $\Delta_j = \|v_{j_2} - v_{j_1}\|$



(b) Selection of a semantic hole: Red diamonds indicate user-selected nodes, the red line indicates the automatically generated path in between.

Figure 2.23: Interactive selection of semantic holes: Local criteria for mesh connection and resulting selection of such a semantic hole.

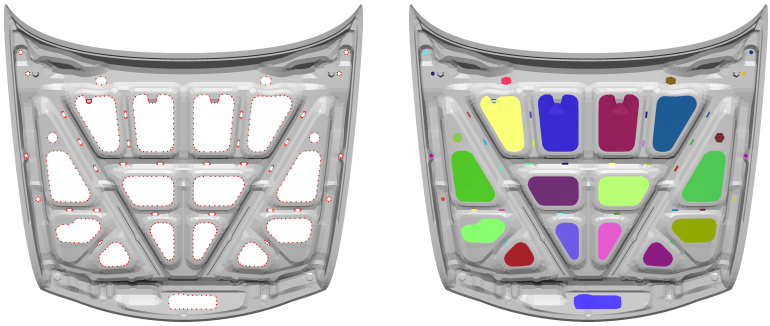
2. The path length $\|d_j\|$ from the start node to the first connecting node is preferred to be as short as possible.
3. The node pair distance Δ_j should be as short as possible.

The angle ρ is the primary criterion as it is also the most intuitive one. However, there still may be cases where a better angle does not result in a good path, e.g. if the chosen node lies on a completely different part of the mesh which has a slightly better angular deviation. Therefore, the second criterion is applied. The third one is the least important criterion as it at best has aesthetic impact. Nevertheless, if the rating for several node pairs according to the criteria above is identical, the node pair distance Δ_j is used to choose the best pair.

Figure 2.23(b) gives an example of a user-defined selection of a semantic hole spanning over several independent meshes: The red diamonds mark the nodes defined by the user, the red line represents the automatically generated selection path in between and thus the border of the semantic hole. Independently meshed parts are coloured individually.

2.4.4 Filling Holes Using an Advancing Front Algorithm

The numerous small and simply shaped holes in the CAE models, such as constructive cut-outs, are usually plane bound and, thus, satisfactory patching can be applied automatically without any need of user interaction. Therefore, the approach [Lie03] previously outlined in Section 2.4.1 was adapted for filling these small holes with minimal surfaces. An example is depicted in Figure 2.24.



(a) Automatically detected topological holes

(b) Topological holes closed by minimal surfaces

Figure 2.24: Topological holes, in general, can be both detected and closed automatically, here applied to an engine hood. Since these holes are mostly plane bound they are closed by a minimal surface meshed independently of the surrounding mesh.

Nevertheless, the more complicated shaped holes and especially semantic holes cannot be closed satisfactorily with this algorithm as constructive features are not preserved and perpetuated into the filling mesh. For these constellations a second approach with supplementary user influence using an Advancing Front Method (AFM) was implemented and adapted to the needs of CAE meshes in order to gain more satisfactory results, described in the following. In case the hole is too weirdly shaped for that method to be applied with moderate user interaction, the previously mentioned minimal surface algorithm is used to patch the hole – maybe imperfectly but sufficiently.

An existing Advancing Front Method [TC04] was adapted and extended in order to create a patching mesh filling the non-simple shaped holes. Advancing Front Methods mesh an area in an iterative manner by inserting new nodes and elements at the borders – the *front* of the area. This way, the front advances into the inside of the area until it is closed. The basic concept of AFMs is pretty simple and advanced functionality and heuristics can easily be added without having to reimplement the whole algorithm. Thus, these methods offer an easy possibility to be extended to suit a certain domain of problems, as in the given case.

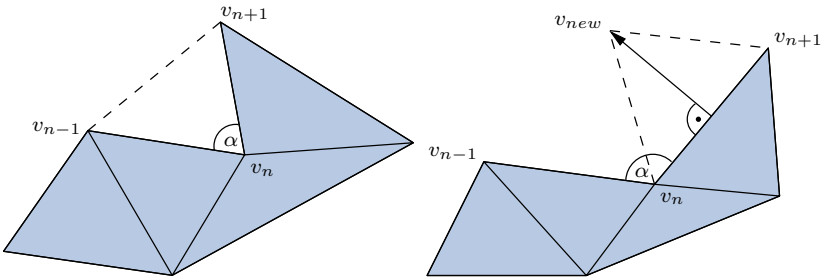
For simplicity, in the following section an *edge* always denotes an edge being part of the advancing front – if not stated otherwise. Without loss of generality, the presented algorithm creates a triangulation. It can be extended to create quad elements, which would be desirable if the resulting mesh was used for FE simulation and not simply as a volume delimiter as in the present case.

Basic Concept

The method basically consists of two steps named *makeConvex* and *addVertices*, illustrated in Figure 2.25. The steps are applied alternately where the choice which of them should be applied depends on the angle α enclosed by two adjacent edges:

makeConvex Adjacent edges enclosing an angle α smaller than a given threshold angle φ are completed to a new triangle by introducing a new edge, illustrated in Figure 2.25(a). Before the new triangle is committed to the mesh, a test is performed to assure that no other triangles are intersected, which would lead to illegal geometry concerning the application case.

addVertices In order to make the front shrink, a new triangle is introduced inside the hole by adding a new vertex v_{new} along the perpendicular bisector b of one of the two adjacent edges enclosing an angle $\alpha > \varphi$, depicted in Figure 2.25(b). Since the new vertex has to suit the surrounding mesh MLS Projection and a biquadratic polynomial is used to adjust the final position of the node on the surface.

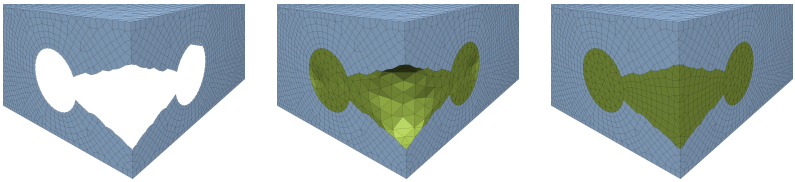


(a) MakeConvex step applied for $\alpha \leq \varphi$ **(b)** AddVertex step applied for $\alpha > \varphi$

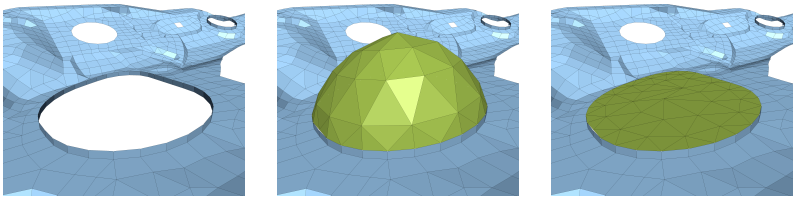
Figure 2.25: The basic steps of the Advancing Front Method: Interlinking adjacent edges (a) to make the front at these edges and (b) inserting new vertices to propagate the front.

Each of the steps is repeated until no edges comply with the given angle criterion, then the other step is applied in the same way. This is repeated alternately until no more front edges exist and hence the hole has been closed.

When applying this procedure without further intervention, two problems arise, illustrated in Figure 2.26: 1. Constructive feature lines cut by the hole boundary are removed in the makeConvex step, as can be seen in Figure 2.26(a), and 2. Holes that are virtually perpendicularly aligned to the surrounding mesh get patched with inappropriately aligned meshes because the MLS projection in the latter step blends the new elements in the surrounding mesh, which is not always intended as can be seen in Figure 2.26(b).



(a) Enclosed feature lines are smoothed out by the MLS projection but are required to be conserved



(b) Smooth blending into the surrounding mesh is not always desirable

Figure 2.26: Problems arising from the use of the MLS projection method.

In order to overcome these problems, improving extensions have been applied to the original algorithm. The problem of smoothed out feature lines is approached by testing whether the node shared by two adjacent edges is part of a feature line and thus the feature line crosses the advancing front. The detection of a feature line is based on the angle β enclosed by the normals of the two adjacent elements: If this angle exceeds a specified threshold angle γ a feature line has been detected. Therefore, in the makeConvex step the two edges must not be completed to a triangle by inserting an individual edge but a new vertex is introduced along the run of the feature line instead, applying rather two new elements which further provide the feature line than one new element ignoring the feature line. Accordingly, in the addVertices step new nodes at these feature lines are handled with particular care and get explicitly aligned with the run of the feature line as depicted in Figure 2.27.

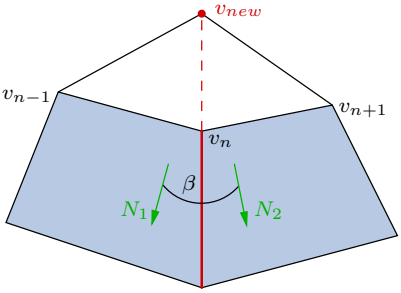


Figure 2.27: Alignment of a new node along a feature line (red).

On the other hand, i.e. if the edge to be processed is not adjacent to a feature line, a new node is inserted along the perpendicular bisector b of the edge – as described previously. Since this vector is calculated by the cross-product between the associated element normal and the edge itself, b lies on the plane of the associated element. This bears the risk of propagating or even increasing noise in the mesh and, thus, the following user-defined support is applied. The neighbouring elements to both sides are taken into account when placing new vertices, and they are weighted with weights w_i using a function similar to a Gaussian distribution:

$$w_i = \frac{1}{\sqrt{2\pi}} e^{\frac{1}{2}i}, \qquad i = 0, \ldots, n-1 \tag{2.11}$$

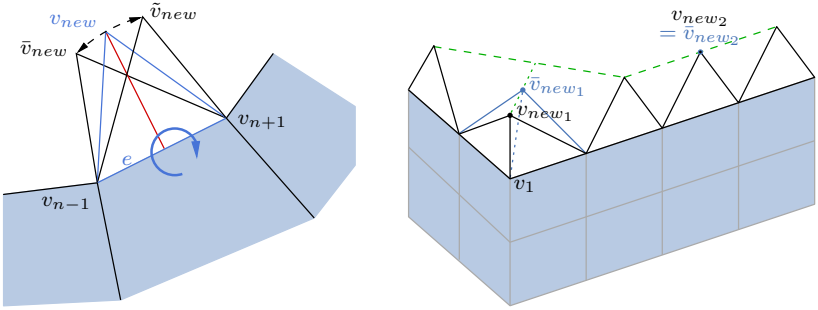
In the extreme example, the support could involve the whole front, which however would cancel the intentional local approach of the algorithm – beside the significant increase in computational complexity. In the end the support leads to a slight smoothing of the filling mesh resulting in a more robust algorithm.

The second problem, namely whether to blend the new mesh into the surrounding one or not, is approached by leaving aside the MLS projection during the addVertices step, preliminary placing the new vertices in the plane of the adjacent element and applying the bending in a subsequent step where the angle can be influenced by a user-given parameter. This procedure is explained in the following paragraph.

Application of Bending Angles

In case not only the hole boundary but also the associated elements are coplanar the initial placement of new nodes – coplanar with the adjacent element – would lead to satisfactory results. But since, in general, neither the boundary edges nor the boundary elements are coplanar the post-processing application of a bending angle is necessary in order to achieve adequate filling surfaces. Especially for complex shaped holes bending is essential – either to blend the new mesh well into the surrounding surface and achieve C^1 continuity or to bend away from the boundary elements even if the connection between the new mesh and the surrounding mesh is only C^0 continuous, previously illustrated in the simple example in Figure 2.26(b). Both scenarios coexist legitimately in the application case since the user expectations of the filling surface highly depend on the hole context.

The bending of the patching mesh is achieved by iteratively rotating the newly inserted nodes around their associated edge according to Figure 2.28(a). The rotation angle is optimised with respect to the neighbouring vertices so that the new front is as short as possible, resulting in a quasi-minimal area triangulation, and upper bound by a user-specified value. This means, that each node is rotated to minimize the distance to its neighbours, i.e. the node is as close as possible to the average of the neighbours, but it is rotated by at most the given angle. Similar to the calculation of the growing direction, a support of weighted adjacent vertices (2.11) along the front is used. The final node position is calculated iteratively by applying the rotation until the positions converge towards a stable situation.



(a) Rotation of a new node v_{new} around the associated edge e, influenced by the adjacent vertices

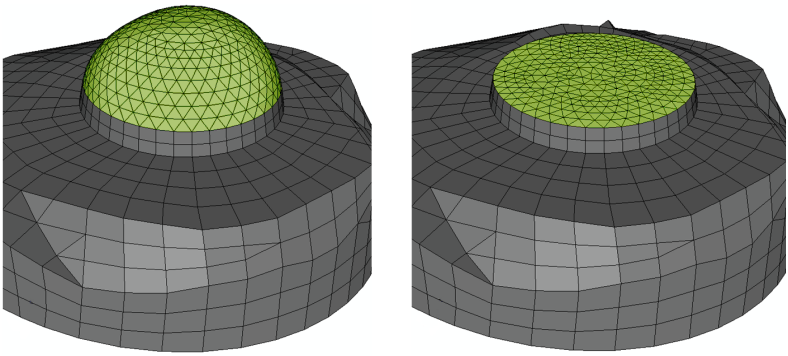
(b) Collinear vertices along the front prevent bending of new vertices

Figure 2.28: Vertex placement optimization by node rotation and its limitations in case of collinear vertices.

Nevertheless, this approach not always leads to the desired result. One problem results from collinear adjoining vertices along the front as they prevent the new nodes from rotating around their associated edge and, thus, the expected optimization of the patching mesh fails. This happens because any rotation would increase the distance between the new node and its neighbours, as depicted in Figure 2.28(b). The bending of other nodes, like the corner vertex v_1 in the Figure, will propagate but not sufficiently fast to achieve satisfactory results. Additionally, front sections growing towards each other may miss to meet if the initial hole borders were not oriented towards each other, resulting in unpleasant and unwanted or even unsolvable constellations.

These problems are tackled by taking points on the opposite segment of the front into account for the vertex placement optimization process. Therefore, a plane normal to the associated edge is placed through the node, and the intersection point of this plane with the opposite part of the front is then considered as an additional neighbour, weighted by its distance to the current vertex. Special care has to be taken for concave bound holes in order to avoid the involvement of intersection points lying behind the vertex. A user-defined cut-off restricts the distance where opposite front segments are considered as neighbours. Applying these enhancements to the algorithm smoothly shaped patching meshes can be achieved. Nevertheless, the problem of meshing holes with boundary elements perpendicular to the plane of the hole, as depicted in Figure 2.29, is not sufficiently solved.

Due to the local approach, the mesh will not bend sufficiently in the first step to allow for a flat mesh patching the hole illustrated in Figure 2.29(b). So if the user wants the connection between the patching mesh and the surround-



(a) Meshed without initial bending **(b)** Meshed with initial bending

Figure 2.29: Influence of initial bending on the patching mesh: Meshing without (a) and with initial bending (b), causing the hole to be meshed flatly.

ing surface to be only C^0 continuous, an additional control mechanism called *initial maximum bend* is applied. Similar to the bending control in case of multiple collinear vertices along the front, a point on the opposite segment of the front is taken into account for the vertex positioning. But in contrast to the scenario described above, that opposite point is not weighted proportional to its distance from the current edge but with a fixed large weight in order to ensure its predominant influence in vertex placement optimization. The influence of this initial bending option is illustrated in Figure 2.29.

Vertex and Triangle Validation

The new nodes and triangles then have to be committed and included in the patching mesh. Before, they are validated in order to ensure good quality and to prevent self-intersections of the patching mesh.

In a first step, nodes that are not distant enough from each other to make a valid edge in the new mesh are merged into a unified vertex in order to avoid awkward constellations as well as cracks in the patching mesh. Subsequently, the new triangles are tested for validity: Intersections with other triangles or parts of the mesh have to be prevented as well as the situation that two non-connected triangles approach each other excessively. Both would lead to degenerated triangles or irresolvable problems in the further process. Thus, the concept of *fences*, proposed in [SSFS06], was adapted for intersection retrieval. These fences are auxiliary geometry appended at each edge of the advancing front, oriented normal to the associated triangle respectively, as illustrated in Figure 2.30.

Using these fences, the validation of new vertices and triangles to be committed can be reduced to an intersection test of the new geometry with these fences.

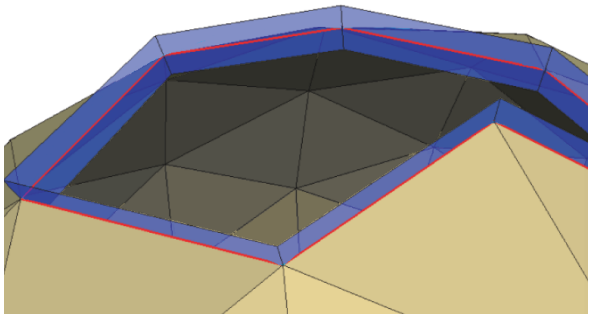


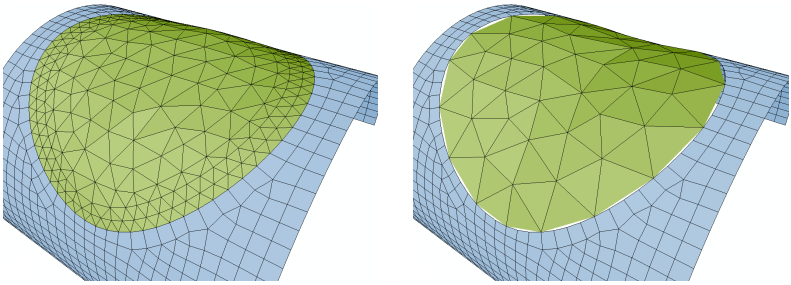
Figure 2.30: Fences as auxiliary geometry used for validation of new vertices and triangles.

The height of the fences is essential for the efficiency of the algorithm since non-connected front segments will be able to approach further than allowed. Are the fences chosen to be too high the intersection test may reject triangles or vertices because of the intersection with the high fences of rather uninvolved front segments. In [SSFS06] this procedure is used during remeshing and an error metric can be used based on the deviation of the new mesh from the original mesh. Since in our case there is no original mesh from which a deviation can be calculated, the fence height is chosen proportional to the average boundary edge length.

Once all new triangles and vertices are validated, they are committed and the advancing front is updated with the new boundary edges.

Edge Length Adjustment

Per default, the edges in the patching mesh that make up the connection to the hole boundary are chosen to have identical edge length as their associated edge on the surface boundary, propagating this edge length into the new mesh. Since in our application not a strictly watertight model is needed but gaps beneath a certain threshold can abide, the user can influence the target edge length. There are two options: One allows for adaptive mesh coarsening and is especially useful to have the perfect fit on the boundary but cutting down the number of triangles in the patching mesh, such as exemplified in Figure 2.31(a), which in turn reduces the time needed to mesh the hole. The second option forces the target edge length in relation to the average boundary edges for the whole mesh. Thus, a homogenic edge length matching the surrounding mesh elements can be achieved resulting in watertight patching of the mesh but, in contrast to the previous option, in a comparatively high number of



(a) Mesh with gradual adjustment of edge length

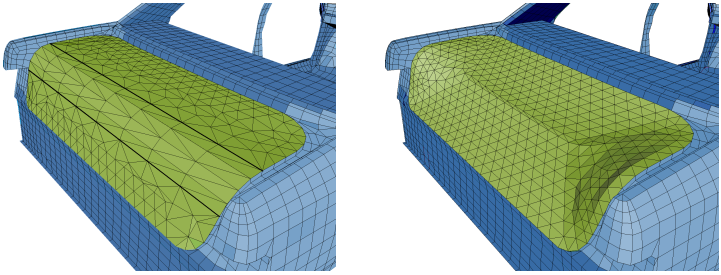
(b) Mesh with forced adjustment of edge length

Figure 2.31: Effect of edge length adjustment in mesh generation: (a) Gradual and (b) forced edge lengths.

triangles. On the other hand, when choosing the edge length longer than in the surrounding mesh, the triangle number can be further decreased but at the expense of gaps between the patching mesh and the surrounding surface. This is illustrated in Figure 2.31(b). But, as already mentioned, these gaps can be tolerated up to a certain size, if a volume element used in the subsequent simulation does not fit through.

2.4.5 Results on Filling Arbitrary Holes in FE Meshes

The approach described in this chapter is meanwhile part of the commercial FE analysis system *PERMAS*[3] and was tested with several CAE car models in the pre-processing of the dedicated scenario of acoustic simulation. The necessity of user interaction could be cut down significantly by the application of various filters classifying the detected holes for their size, their location and shape, i.e. the flatness of their boundary. By applying these filters the number of holes that need user-interaction could be reduced to less than 20% of the already filtered holes. Furthermore, the implemented interaction mechanisms provide the ability of interactively reviewing the holes to the user who can, thus, easily influence both the selection of holes to be meshed and the meshing itself. Since the meshing of even complex and large holes is accomplished within a few seconds the user can even vary the control parameters to gain the mesh that suits his needs best if the initially created mesh did not. For complex shaped holes it can be difficult to generate an appropriate mesh in one step, thus holes can be optionally subdivided into less complex ones that are closed subsequently as depicted in Figure 2.32.



(a) Manually segmented and subsequently meshed hole.

(b) Automatically closed car trunk.

Figure 2.32: Two versions of the same semantic hole of a car trunk closed: Smoothly meshed (a) by interactive manual definition of three holes meshed subsequently in order to achieve an optimal mesh. The automatic filling of the same hole is demonstrated in (b).

[3]PERMAS is a product of INTES. `http://www.intes.de/kategorie_permas/uebersicht`

Applying the presented process to contemporary models in real scenarios it turned out that the overall processing from hole detection over meshing to volume generation for the subsequent simulation can now be done in few hours instead of days as before. To give an example, in a plain car body model without chassis consisting of 463 independent parts 3359 holes have been detected. Due to the filtering methods previously mentioned, this number could be reduced to 248 since holes smaller than the volume mesh tetrahedra as well as the holes outside the region of interest can be ignored and can thus be excluded from the meshing process. The flatness criteria allowed the automatic closing of 201 of these holes resulting in only 47 holes to be inspected interactively by the user before being closed – two of them were partitioned as exemplified for the semantic hole of a car trunk in Figure 2.32. An example of a car part with both automatically meshed holes and holes closed after user interaction for ideal meshing is given in Figure 2.33.

Beside the filtering method and the automated closure of the vast majority of holes, the interactive definition and the resulting ability of closing semantic holes plays a decisive role for the gained speed-up. Figure 2.34 gives two examples of such semantic holes bounded by several independently meshed parts. Both holes were manually defined and then closed automatically.

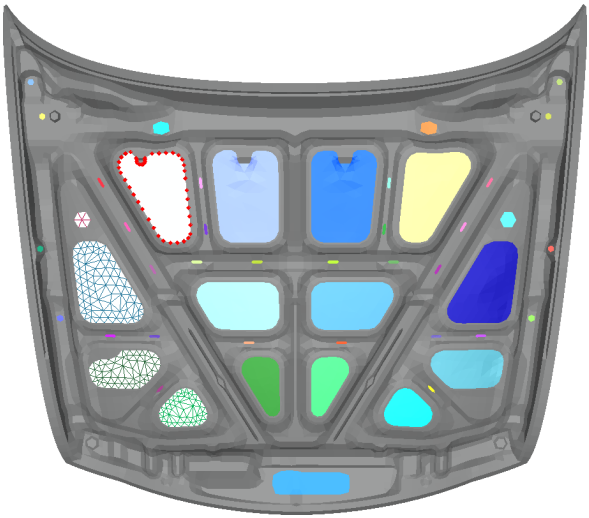
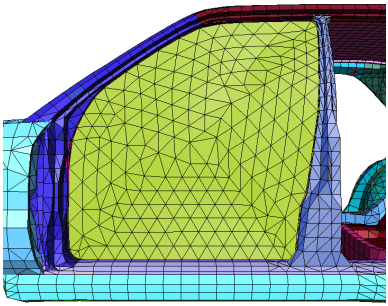
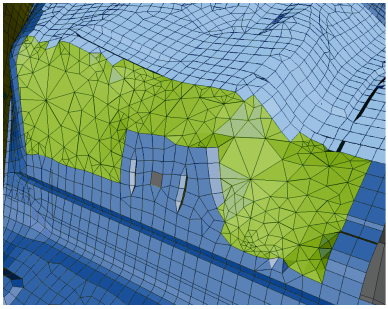


Figure 2.33: The flatness classification criteria of holes allows for automatically meshing the vast majority of all holes. In the given case only one of 48 holes inside the engine hood required user interaction (red dotted boundary).



(a) Automatically meshed semantic hole previously defined in Figure 2.23(b).



(b) Meshed hole between engine compartment and centre console.

Figure 2.34: Examples of automatically filled semantic holes that were defined manually.

Summing up, the presented process of detecting, defining and closing arbitrarily shaped holes in FE meshes enables the user to handle the vast majority of holes automatically, still having the possibility to influence the overall process if desired. Semantic holes usually cannot be detected automatically as their boundary spans over several separate meshes, and therefore ignored by other approaches, can also be defined and closed by the process presented – either automatically or more specifically according to the user's intention, whereas the algorithm still keeps in mind the special requirements of meshes in CAE models.

The methods presented in this chapter emerged to valuable tools in preprocessing, speeding up the development process and providing fast and intuitive interaction methods to the user. Hence, they found their way to productive use in the application field.

3 Beyond Meshes – Applications in Biochemistry and Live Science

The previous chapter covered the mesh-based visualization of geometries rooted in the structural and, in parts, topological properties of the models. By contrast, this chapter deals with models and geometries that, due to their structure, do not compellingly demand mesh-based representations. The applications examined here are based on derived or abstract geometric properties instead of intrinsically given geometric properties of the visualised data.

Visualization plays an increasingly crucial role in the research area of molecular dynamics. The properties and functionality of proteins is largely investigated performing molecular dynamic (MD) simulations. Appropriate visualization, and especially, analysis methods are needed to gain deeper insight into the resulting data. Hence, this also leads to a very active research field of molecular visualization over the past decade.

One part of this research focuses on the direct visualization of the molecules in various representations – chosen based on the particular task in the application field. After a short investigation of the basic properties of protein-solvent systems and related work on this field, these representations will be detailed in the following of this chapter, in Section 3.3. There, the focus lies on visualization methods supporting the dynamic properties of the data and still providing interactive representations.

Another part of molecular visualization is the investigation of new analysis methods rooted in visualization techniques. Especially the interplay of the protein with the surrounding solvent is a challenging task to enquire. In Section 3.5 a method for haptically exploring the flexibility of proteins is presented. The solvent around the protein is the subject of Section 3.6 with techniques for the analysis of its dynamic properties.

The final section of this chapter, Section 3.7 constitutes a further step towards medical visualization as well as abstraction of geometry and data, where, in the context of diffusion tensor imaging, hyperstreamlines are introduced to represent the diffusion tensor, first showing all three parameters of the tensor in a continuous representation.

3.1 Introduction to Protein-Solvent Systems

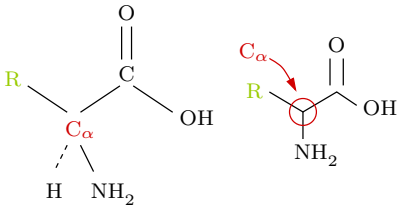
The behaviour and functionality of proteins is highly influenced by the surrounding solvent. Therefore, not only the proteins themselves are subject of the MD simulations, but a system of one or several proteins embedded in a box of solvent molecules. This solvent can either be constituted of a single molecule type – often water – or it can be a mixture of several types of molecules. The aim of these simulations is a deeper insight into the protein functionality, especially important, e.g. for the better understanding and prevention of diseases and, thus, in pharmaceutical research. The details of the MD simulations are given in Section 3.4 before focusing on the dynamic characteristics of the simulation data. The impact of visualization in this context is, on one hand, the direct representation of the molecules for the post-simulation analysis and, on the other hand, providing specialised methods for the simulation data analysis itself. Both parts will be discussed in the following sections.

Proteins are, by structural means, the most complex and finest adjusted molecules [AJL+04]. Thus, their representations have to support several levels of abstraction to focus on various characteristics. For a better understanding of these representations, the protein structure is detailed in the following.

Protein Structure

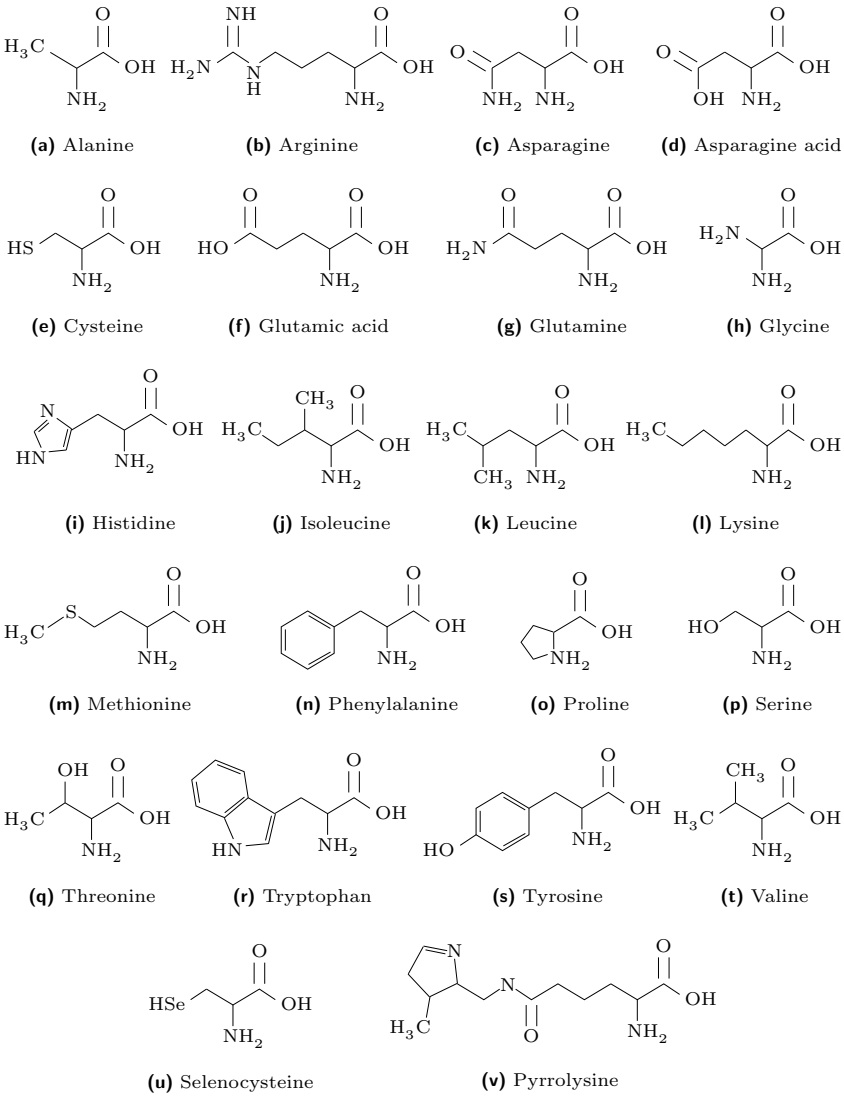
Proteins are linear, usually long-chained, macromolecules composed of a special group of amino acids called *in the alpha position* or *proteinogenic* amino acids. The chain length can vary from very few amino acids up to several thousands, with an average of about 460. The smaller proteins with up to about 20 amino acids are called *peptides*, longer chained ones are denoted as *polypeptides* [Lod04].

There are 20 different proteinogenic amino acids (sometimes two further ones are added [Poe05]), illustrated in Figure 3.2, which all have a common characteristic structure: A central carbon atom called C_α bonded to four different chemical groups, namely an amino group ($-NH_2$), a carboxyl group (–COOH), a hydrogen atom (-H) and one group called *side chain* or *R group*. Together, the three former groups form the part which is identical to all amino acids, whereas the side chain varies from amino acid to amino acid. This basic structure is depicted in Figure 3.1.



(a) Classical view **(b)** Compact view

Figure 3.1: Layouts of the chemical structure of proteinogenic amino acids.



(a) Alanine **(b)** Arginine **(c)** Asparagine **(d)** Asparagine acid

(e) Cysteine **(f)** Glutamic acid **(g)** Glutamine **(h)** Glycine

(i) Histidine **(j)** Isoleucine **(k)** Leucine **(l)** Lysine

(m) Methionine **(n)** Phenylalanine **(o)** Proline **(p)** Serine

(q) Threonine **(r)** Tryptophan **(s)** Tyrosine **(t)** Valine

(u) Selenocysteine **(v)** Pyrrolysine

Figure 3.2: Chemical structures of the proteinogenic amino acids. Amino acids (a)–(t) are usually referred to as the standard amino acids, however in some organisms (u) and (v) are included [Iup99, AG02].

In the following only proteinogenic amino acids will be considered and, thus, to simplify matters, will be referred to as *amino acids* only.

Within the protein, the amino acids are joined together by the *peptide bonds* between the carboxyl and amino groups of adjacent amino acid residues, illustrated in Figure 3.3, leading to a repeated sequence of amide (N), α carbon (C_α) and carbonyl (C) atoms. This sequence forms the *backbone* of the protein molecule to which the various side chains are attached. Consequently, one end of each protein has an unlinked amino group, the other end an unlinked carboxyl group. The side chains, which are not part of the protein bond, define the individual characteristic of the corresponding amino acid by variation in size, shape, charge, hydrophobicity, reactivity and polarity [Lod04].

The atoms along the protein chain are connected by the C_α atoms of the residues and are fixed in a plane per residue, with the bond lengths and bond angles being virtually the same all along the protein chain [BT98, AJL+04].

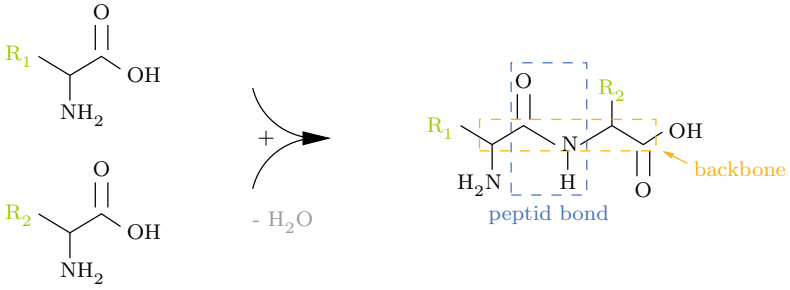


Figure 3.3: Basic structure of proteins exemplified by the peptide bond between two amino acids. The backbone is framed in orange.

Due to their strict construction and the resulting stabilising bonds and forces the protein structure can be categorised hierarchically, detailed in the remaining of this section.

Hierarchical Protein Structure

Primary Structure There are thousands of different proteins known, altogether carrying out an incredible array of diverse tasks, and where each protein has a unique and characteristic amino acid sequence. This sequence is called the *primary structure* of the protein and includes the steric configuration of the amino acids [LSK05]. The spatial organization of the peptide chain, called the *conformation*, specifies the protein function and is organised hierarchically by the *secondary, tertiary* and *quaternary structure* of the protein [LSK05, Lod04].

Secondary Structure The local spatial arrangement of the backbone which can be formed without the influence of the side chains is called the *secondary structure*. A single polypeptide can feature multiple secondary structure types depending on its amino acid sequence. In absence of noncovalent bonds the structure is called *random coil*. On the other hand, parts of the backbone fold into well-defined periodic structures – called the α *helix*, the β *sheet* and a U-shaped *turn* – when stabilising hydrogen bonds are formed between particular residues. The ratio of random coil and turn in an average protein sum up to about 60%.

The secondary structure of proteins was primarily predicted and categorised by Pauling and Corey [PCB51, PC51]. Levitt and Greer later presented a method of automatic assignments of secondary structure elements in [LG77]. Richardson [Ric81] structured and illustrated the prior found characteristics as ribbons and sheets, being common practice today. A good overview of the theory of the secondary structure of proteins is given in [KS83]. These elements of the secondary structure are composed as follows.

The α Helix: The backbone is folded into a right-handed helix which is stabilised by hydrogen bonds between the carbonyl oxygen atom and the amide hydrogen atom of the amino acid four residues further along the polypeptide chain. This periodic arrangement of bonds implicates a directionality on the helix as depicted in Figure 3.4 and causes the outer surface of the helix to be covered by the side chain groups.

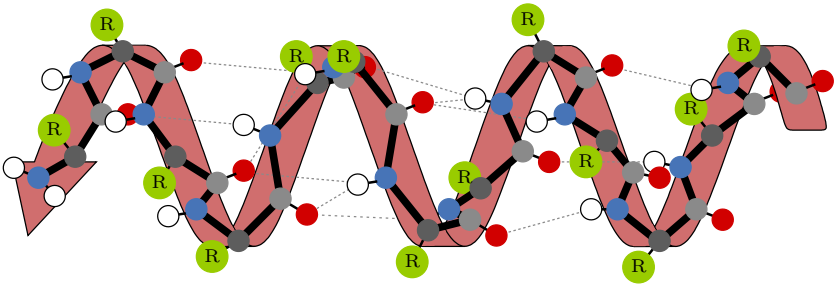


Figure 3.4: The α helix, a secondary structure element stabilised by hydrogen bonds (dotted lines). The protein backbone (thick black lines) curls to a helical structure, emphasised by the reddish arrow headed ribbon. The involved atom types are colour coded: Oxygen red, hydrogen white, nitrogen blue and carbon grey, the green circles represent the amino acid side chains.

The β Sheet: This type of secondary structure consists of parallel arranged β strands, where each strand is a short and nearly full extended polypeptide segment. The β sheet is then formed by hydrogen bonds between

the backbone atoms in adjacent strands – either within the same protein chain or between different ones. Since the peptide bonds are planar, the β sheets are wavy or pleated. Like in α helices, the orientation of the peptide bonds forces the β sheet to be directional: Adjacent β strands can either be oriented in the same, thus parallel, or the opposite, thus antiparallel, direction as depicted in Figure 3.5. In both cases the side chains stick out from both faces of the resulting sheet.

Turns: These short, U-shaped secondary structures are composed of three or four residues and are located on the surface of the protein. Their sharp bends redirect the backbone towards the interior [Lod04]. Turns are stabilised by hydrogen bonds between their end residues and allow large proteins to fold into highly compact structures.

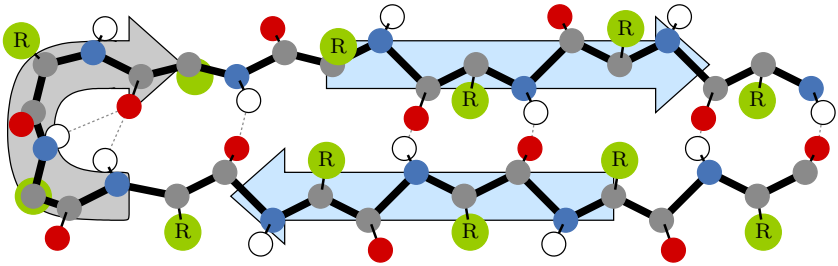


Figure 3.5: Parallel arranged strands form a β sheet, here a simple two-stranded anti-parallel sheet (blue arrows). The strands are connected by a short turn (grey arrow).

Tertiary Structure The *tertiary structure* includes the spatial arrangement of the protein backbone together with the side chains, and, thus, refers to the overall conformation of the protein chain. The structure is – in contrast with the secondary structure – primarily stabilised by hydrophobic interactions between the nonpolar side chains, hydrogen bonds between polar side chains, and peptide bonds, which keep the secondary structure elements compactly together. However, the tertiary structure is not rigidly fixed but subject to conformational changes, since the stabilising interactions are weak. This behaviour has an important influence on the protein function and, thus, is an important subject of molecular dynamic simulations in order to analyse and understand the protein function.

Quaternary Structure The *quaternary structure* refers to the composition of several protein chains into a multimeric protein [BT98, AJL$^+$04, Lod04].

3.2 Related Work on Visualization of Protein-Solvent Systems

The visualization of molecules, and proteins in particular, is a busy research area. Starting from early works about the definition of molecular surfaces [LR71, Ric77] and their efficient implementation [Con83, VBW94], current research mainly deals with the interactive high-quality representation of large molecular structures – often exploiting the programmability of modern GPUs. Just to mention a few of them, Lampe et al. [LVRH07] presented an approach for visualising proteins using a point-based method. For a better understanding of the three-dimensional structure, Tarini et al. [TCM06] presented a GPU-accelerated method for applying ambient occlusion to the real-time visualization of the space-filling representation. Other approaches are more focused on visualising the conformational changes of the molecules (e.g. [SEBH02]).

Additionally, there are many different visualization tools and frameworks for protein and protein-solvent data gained from MD simulations. The probably most popular ones are PyMOL [DeL02] and VMD [HDS96], but there are many more such as Chimera [PGH+04], MoilView [SEZ95], MD Display [CSL96] and commercial tools like Amira [Ami]. Most of them are being developed over a long period of time and hence offer a wide range of functionality for analysing the protein but usually lack special features for processing the solvent. On the other hand there is a growing number of GPU-optimised viewers like the BioBrowser [HOF05] and TexMol [BDST04], which feature good performance and rendering quality but lack support for time-dependent datasets such as offered by VMD and Chimera. Unfortunately, none of these solutions takes special care of the solvent surrounding the protein. This special task will be discussed in Section 3.6. The details about the various protein representations will be further treated in the following sections.

3.3 Protein Representations

The differentiation between atoms and molecules dates back to the early 19th century. The first graphical representations of the molecules arose in the 1860s when Loschmidt published his booklet *Chemische Studien I* [Los61] which also features two-dimensional sketches of molecules with atoms represented by circles in different sizes. Nowadays, the information about the basic 3D configuration of proteins is gathered from X-ray crystallography at high resolution.

Molecular structures, and thus proteins, do not have an inherent representation. Nevertheless, there are several well-established representations on different abstraction levels. They can be classified as follows: *Atom-based* representations, where the individual atoms and connections are represented, and *secondary-structure-based* illustrations showing a higher abstraction level. The latter ones are preferred when analysing properties depending on the semantic

structure of the molecule, further described in Section 3.3.2. In contrast, the former ones are used where the exact position or connection information of the individual atoms is needed. The third category is provided by surface representations that give the correct position of the outer atoms. They neglect the inner structure of the protein but show the protein as it is "seen" from another molecule.

Even though these representations are well-established they still pose a challenge in some ways. As in other areas, on the one hand the visualizations are expected to get more and more aesthetically pleasing and geometrically accurate, and the size of the datasets being explored are increasing considerably. On the other hand, the visualization should provide interactive frame rates to guarantee convenient and comfortable analysis and exploration of the data. Especially when handling large time-based datasets the compliance with all these requisites is not achievable in the first place, since the protein is usually visualised together with other structures, such as the numerous surrounding solvent molecules.

3.3.1 Atom-Based Representations

In atom-based molecular representations the individual atoms are visualised, commonly by spheres. Depending on the task, different variants, mainly depending on the sphere radius, are used. Another option is the visualization of the atom connections instead of or alongside with the spheres.

The probably first molecular model was pioneered by Hofmann in 1865 on a discourse at The Royal Institution where he presented a methane model with croquet balls of different colours representing the different atom types connected by thin sticks [Oll72]. The model was not yet realistic since it showed the molecule being flat-shaped. But still these models were the invention of the nowadays called *ball-and-stick* representation which is widely used. The spatial arrangement had been recognised some years later and presented by van't Hoff in 1875. The first paper was a short publication in Dutch which was shortly followed by longer versions in French (under the name *La chimie dans l'espace*) and in German [vH77].

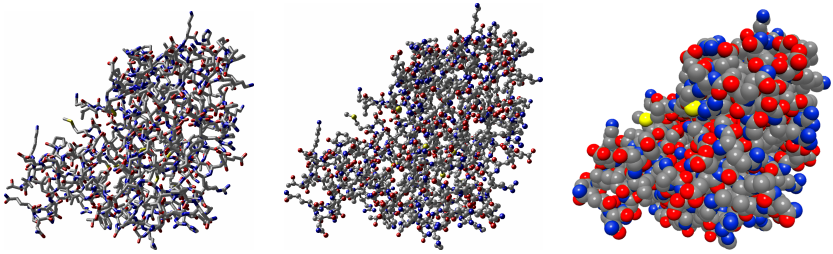
Derived from these early works several atom-based molecular representations are still in use – detailed in the following and exemplified in Figure 3.6. Besides these representations, the colour-coding chosen by Hofmann is still popular as well. In the commonly used CPK colour model – designed by chemists Robert Corey and Linus Pauling and improved by Walter Koltun, hence the name – nitrogen is represented in blue, oxygen in red, carbon in grey or black, sulphur is coloured yellow and hydrogen white or light grey – if represented at all.

Stick: In this representation shown in Figure 3.6(a) only the atom bonds are visualised using small tube segments or sticks – as the name indicates.

For a better visual impression the tube ends are usually furnished with spherical caps. The atoms and their position are given only implicitly by the bifurcation points and tube segment ends.

Ball and Stick: In the ball-and-stick representation, the atoms themselves are directly shown as spheres, as exemplified in Figure 3.6(b), usually with arbitrary radius. The bonds, as in the stick representation, are represented by small sticks or tubes between these spheres.

Space-filling: A third atom-based representation is the space-filling representation depicted in Figure 3.6(c) where the bonds are neglected but the atoms are visualised by their van der Waals radius which is a measure of the size of the electron cloud surrounding the atom. Thus, each atom type has a characteristic van der Waals radius.



(a) Stick representation

(b) Ball-and-stick representation

(c) Space-filling representation

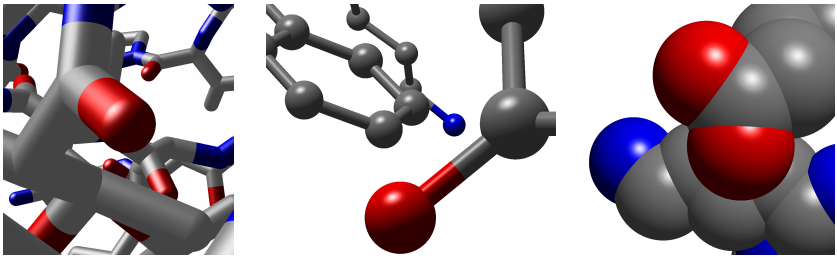
Figure 3.6: Common atom-based representations of the same protein. Close-ups of the representations can be seen in Figure 3.7

Since in the space-filling representation the size of the outer spheres prevents a look *into* the molecule, this representation can also be ranked among the surface representations discussed in Section 3.3.3. But since the visualization methodology complies with the stick and ball-and-stick representations, the space-filling representation is discussed in this section.

The previously mentioned representations only consist of two primitives: Cylinders and spheres. Due to the age of the common tools mentioned in Section 3.2, these primitives are still often tessellated, which is both an outdated technique for these simple shapes and for geometrical purposes, only an inappropriate approximation of the shapes. As mentioned above, the molecules do not have an inherent representation, so the visualizations are always based on abstraction – but the primitives used in these abstractions do, especially in the case of the atom-based molecular representations.

Besides that, an efficient point-based implementation of both primitives, based on the raycasting technique previously explained in Section 1.1, was available

in the institute group [RE05], which was adapted to the present protein data. As a consequence, not only a smooth and visually pleasing visualization depicted in Figure 3.7 could be achieved but also high frame rates compliant with the requisite of visualising large time-dependent datasets.



(a) Stick representation

(b) Ball-and-stick representation

(c) Space-filling representation

Figure 3.7: Close-ups of the atom-based representations.

3.3.2 Secondary-Structure-Based Representation

The secondary structure is visualised by the *cartoon* representation. The different kinds of secondary structure elements, detailed in Section 3.1, have a special graphical representation each, exemplified in Figure 3.8 on a small peptide. Random coil as well as the turns are represented by tubes, the α helix is depicted by a broad ribbon that coils up the turns of the helix defined by the backbone. The β sheet is visualised by broad ribbon-shaped arrows that follow the course of the backbone and point towards the carboxyl group at the end of each β strand.

Apart from the definitions mentioned above, no exact definition of the visual representation of the secondary structure elements is given. Thus, the cartoon representation looks slightly different in all viewers. We chose to follow a common convention using round-edged ribbons for the helices and sharp-edged ones with arrow heads for the sheets. The turns and random coil are represented by thin tubes as depicted in Figure 3.8. Since the cartoon representation is an abstract representation of the protein, the course of the secondary structure elements, in general, does not interpolate the positions of the backbone atoms but approximates them smoothly. However, the orientation of the ribbons illustrating the helices and sheets is a significant feature of the representation. The ribbon is always depicted lying in one plane with the hydrogen bonds stabilising the secondary structure element, which also corresponds to the peptide bond plane, both previously detailed in Section 3.1.

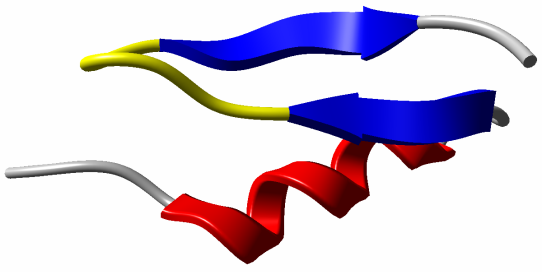


Figure 3.8: Protein in cartoon representation, coloured by secondary structure elements: The α helix is depicted red, the β sheets represented by blue ribbon-shaped arrows, the turn by yellow tubes and the random coil is coloured grey.

Since the cartoon representation only visualises the structure of the protein, most common colouring methods, like atom-based or amino-acid-based, are not suitable. Instead, the secondary structure elements are rather coloured by the element type where α helices are usually coloured red and the strands of β sheets blue. Another common colour scheme is the *rainbow* colouring where a colour gradient is applied along the polypeptide chain. This does not describe any biochemical attribute but enables a human observer to easily perceive the distance between particular regions in the chain.

The work presented in this Section was mainly implemented by Michael Krone as part of a project thesis and is published in [KBE08].

Geometric properties of the Cartoon Rendering

The curve defining the course of the secondary structure elements was chosen to be represented by cubic B-splines, previously discussed in Section 1.2.1, following the suggestion of Carson et al. [Car91]. Thus, a satisfying compromise between the calculation time and the output quality can be achieved and a smooth curve is computed that sufficiently approximates the characteristics of the backbone. The C_α atoms of four successive amino acids constitute the control points of a spline section. In order to create smoothly merging spline sections the last three coordinates of the preceding section together with the coordinates of the subsequent C_α atoms are used. Thus, each spline section forms the curve between two C_α atoms. These splines are then discretised for visualization where the number of line segments approximating the curve progression for each section is controlled by a parameter.

For the computation of the ribbon alignment, the direction of the bond between the backbone carbon and the oxygen atom is utilised instead of the correct

direction of the hydrogen bond. This applied direction is virtually equal to the hydrogen bond's direction and can be computed easily from the backbone atoms of one amino acid [CB86]. The major advantage in this is that no further information about the hydrogen bonds stabilising the secondary structure elements is needed. Hence, the orientation of the ribbons is determined by the generation of a secondary spline where the control points of the primary spline are shifted along the atom bond between the carbon atom and the oxygen atom. Prior to the computation of the second B-spline the direction of the oxygen atom must be determined because it may flip. If the angle between two consecutive directions is wider than $\pi/2$, the latter must be flipped, i.e. multiplied with (-1). The scheme of the two splines that determine the ribbon orientation is depicted in Figure 3.9.

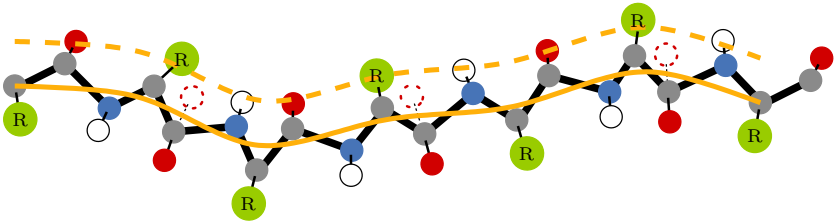


Figure 3.9: Schematic drawing of the basic B-spline (orange) and the secondary one (dashed orange curve) used for alignment of secondary structure elements. Dashed oxygen atoms denote flipped directions.

For each line segment the geometric primitives forming the cartoon representation are computed. According to the schematic view in Figure 3.10, six vertices $p_0, ..., p_5$ are needed to compute the geometry, namely the end points of the current line $\overline{p_1p_3}$ as well as the preceding line segment $\overline{p_0p_1}$ and the succeeding one $\overline{p_3p_5}$. Additionally the end points p_2 and p_4 of the connections from the current line segment to the corresponding one in the second B-spline are needed.

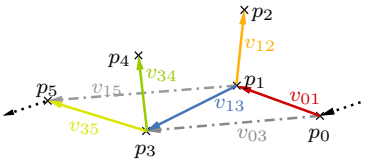


Figure 3.10: Vertices p_i and corresponding vectors v_{ij} used to compute the geometry for the current line segment v_{13}. The vertices p_0, p_1, p_3 and p_5 represent the C_α atoms whereas p_2 and p_4 are the oxygen atoms used for the ribbon alignment.

Once the necessary vertices and vectors are determined, the geometric representation of the cartoon elements are generated. If the current line belongs to a β sheet, a hexahedron is created that surrounds the line, according to Figure 3.11. The front face is a rect-

angle spanned by the vectors v_{34} and $v_{34} \times v_{15}$ which are scaled to fit the desired size of the resulting ribbon segment. Respectively, the back face is a rectangle spanned by the scaled vectors v_{12} and $v_{12} \times v_{03}$. Thus, the back face of a segment and the front face of a preceding one are generated from the same vectors. By applying this creation scheme continuous joints without gaps or crossovers are attained. The arrow heads that indicate the head of the β strands are generated by scaling the font and back faces of the line segment geometry along the last curve section of the strand.

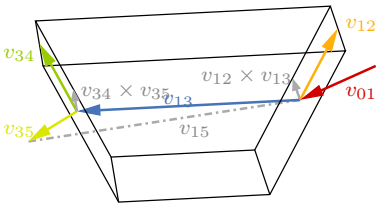


Figure 3.11: Geometry for a ribbon line segment (β sheet and α helix).

The helix ribbons are computed analogous except that they do not have arrowheads and that their lateral edges are bevelled which results in the desired rounded look. The normals are set in such a way that the lighting amplifies this effect.

The geometry representing the random coil and turns looks quite different. Corresponding to Figure 3.12 the front and back faces of the tubes that describe these cartoon elements are convex, regular polygons. These polygons are created by rotating the vector v_{34} n-times by a uniform angle of $2\pi/n$ around the axis defined by the vector v_{15} in case of the front face. For back faces the vector v_{12} is rotated around v_{03} respectively. As with the sheet and helix geometries the back and front faces of consecutive line segments thus connect without gap between these segments.

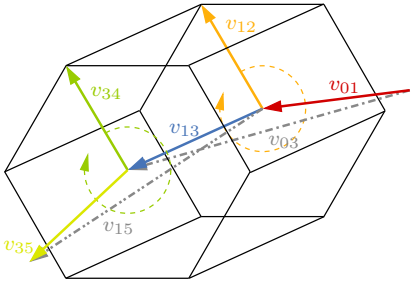


Figure 3.12: Geometry for a line segment of random coil or turn.

Three methods for rendering the cartoon representation were implemented: a CPU implementation, a plain GPU implementation and a hybrid implementation. The CPU and the hybrid implementation create the same graphical output. The GPU implementation is a testing implementation to analyse the abilities and limitations of current high-end consumer graphics hardware. Therefore, its graphical output is slightly differing from the other two. Further details about the different implementations are discussed in [KBE08].

Results on Cartoon Representation

The visual quality of the cartoon representation highly depends on the tessellation. Since all constraints for the appearance are parametrised, the user can adjust the graphical output according to his needs and the hardware limitations by configuring the number of geometric primitives.

Approximating each curve section with 12 line segments and rendering the tubes of random coil and turns with 12-sided front and back faces, a visually satisfying representation can be achieved whose quality is close to raycasting. The cartoon representation can be rendered at interactive frame rates with per-pixel lighting and high tessellation, even for large proteins as 1AON depicted in Figure 3.13.

For example a dataset with the trajectory of a molecular dynamics simulation containing 50,000 time steps can reach a size of about 17 GB. With such large datasets, pre-computation time and especially additional storage demands must be minimised to enable interactive rendering. Since the hybrid implementation only pre-computes and stores the vertices and parameters of

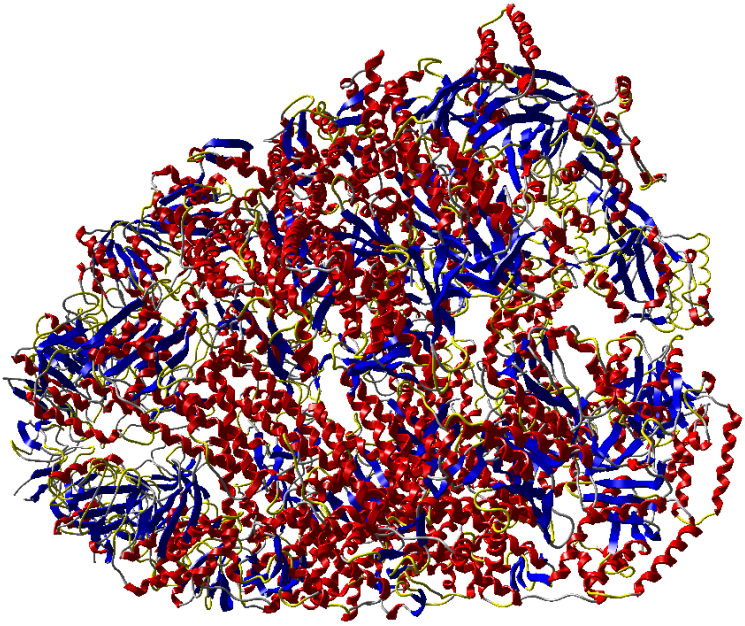


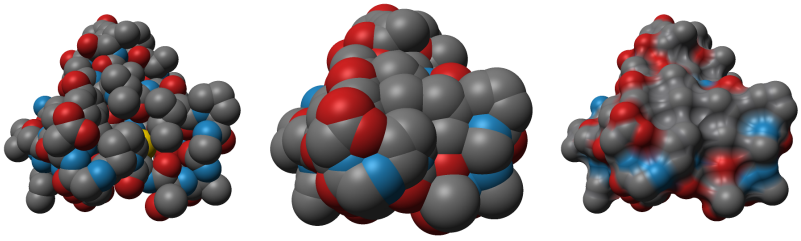
Figure 3.13: One of the largest protein structures known: 1AON consisting of about 8,000 amino acids in 21 protein chains, here in cartoon representation.

the two splines instead of all the vertices for the geometry, not only the precomputation time for each time step is significantly lower but also the memory requirements are reduced at least by half versus the hybrid implementation. Thus, rendering at interactive frame rates can be achieved.

3.3.3 Surface Representations

The previously described molecular representations focus on the representation of the protein structure, either the primary or the secondary structure, but they give only little information about the outer boundary of a molecule such as it is accessible or "seen" by the surrounding solvent or other molecules. This property is considered by the surface representations. When dealing with protein-solvent systems the surface representation is of special interest, since the protein-solvent interaction can be efficiently explored. There, surface representations are particularly helpful to identify cavities or channels accessible by the solvent or other specific molecules, having an impact on the protein characteristics. Additional typical applications are docking – the analysis of protein-protein or protein-ligand interactions and assemblies – and the exploration of specific phenomena which occur at the surface, like binding sites or hydrophobic and hydrophilic regions.

There are three common types of molecular surface representations exemplified in Figure 3.14. The simplest one is the space-filling representation, previously presented in Section 3.3.1, where each atom is represented by a sphere with the radius according to the atom's van der Waals radius and, thus, also referred to as *van der Waals surface*. An example is depicted in Figure 3.14(a). A momentous drawback of the van der Waals surface is that it has no correlation with the surrounding medium. This problem is coped with by the two



(a) Van der Waals surface **(b)** Solvent Accessible Surface **(c)** Solvent Excluded Surface

Figure 3.14: Comparison between various surface representations of the same peptide. Characteristically, crevices in the van der Waals surface are closed in the solvent accessible and the solvent excluded surface.

other surface representations, since they are defined from a different point of view – from the solvent's point of view. Both surfaces, the *solvent accessible surface*, shown in Figure 3.14(b), and the *solvent excluded surface*, given in Figure 3.14(c), are defined by a spherical probe rolling over the van der Waals surface. The definition of these surfaces will be detailed in the following.

The solvent accessible surface, first presented by Lee and Richards [LR71] in 1971, is defined by the centre of this spherical probe which rolls over the van der Waals surface of a molecule and depicts the surface that is directly accessible for solvent atoms of a particular type. The radius of the probe equals the van der Waals radius of the designated solvent atom type. Like the van der Waals surface, the solvent accessible surface geometrically consists of spherical surface patches bounded by small circle arcs. In Figure 3.15, the definition of the solvent accessible surface is illustrated schematically and coloured in blue. In other words, the solvent accessible surface corresponds to the space-filling representation with enlarged atom radii, namely the probe radius added to the van der Waals radius of each atom. Due to this inflation, minor gaps and crevices in the van der Waals surface which are not reachable by solvent atoms are closed in the solvent accessible surface as can be observed in Figure 3.14(a) and (b) respectively. Thanks to the close relation with the previously described space-filling representation, in terms of geometry, the solvent accessible surface can be rendered in the same way as the space-filling one: By GPU raycasting of the corresponding spheres.

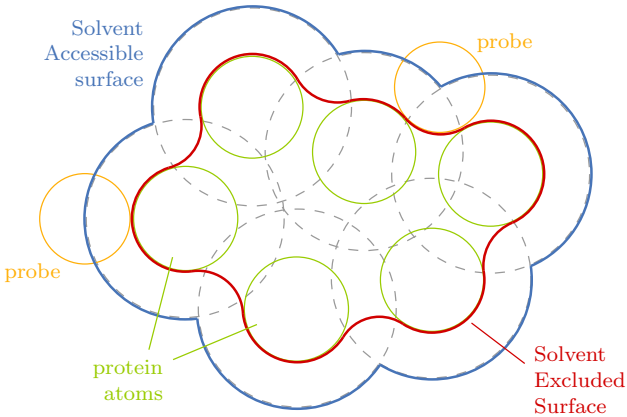


Figure 3.15: Sectional drawing of the solvent accessible surface (blue) and the solvent excluded surface (red), both defined by a probe (depicted in two sample positions and coloured in orange) rolling over the molecule atoms sized according to their van der Waals radius (green). The hypothetical path of the probe centre is traced in dashed lines per protein atom.

In 1977 Richards [Ric77] proposed an additional molecular surface which he simply called the *molecular surface* or the *smooth molecular surface*. It is defined by the union of two kinds of spherical patches: The *contact patches* are those parts of the van der Waals surface that can be actually in contact with the probe and the *reentrant patches* are the interior-facing part of the probe surface when it is simultaneously in contact with two or three molecular atoms. Together these patches form a continuous surface, the molecular surface, which is, thus, composed of both convex and concave elements. This surface was redefined by Greet and Bush [GB78] to be the surface composed of points at which the probe sphere approaches closest possible to protein atoms without intersecting them. Hence they called their surface the *solvent excluded surface* and additionally proposed an algorithm for the surface calculation. The resulting surface equals Richards' smooth molecular surface.

In other words, the solvent excluded surface as well is defined by a probe sphere rolling over the van der Waals surface of the molecule, but not the probe centre traces out the surface but the surface of the probe does, as illustrated by the red curve in Figure 3.15.

Properties of the Solvent Excluded Surface

The solvent excluded surface finally consists of three different types of surface patches and, thus, geometric primitives, which are marked in Figure 3.16 and are formed as follows: Concave spherical patches (red) bounded by small circle arcs, saddle-shaped toroidal patches (white) and concave spherical triangles (blue).

Spherical patches They occur when the probe is rolling over the surface of a single atom and has no contact with any other atom (two degrees of

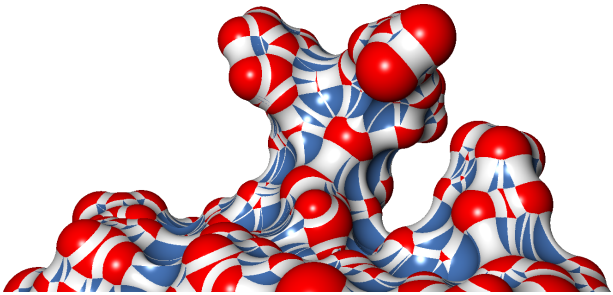


Figure 3.16: Solvent excluded surface with surface patch types coloured individually. Spherical patches being part of molecular atom surfaces are coloured red, the toroidal patches white and the spherical triangles are coloured blue.

freedom). All areas of the atom surface which can be in contact with the probe surface will be part of the solvent excluded surface. Spherical patches are coloured red in Figure 3.16.

Toroidal patches These patches are formed when the probe has simultaneous contact to two atoms and rotates around the axis connecting the atom centres (single degree of freedom). The probe then traces out a torus whose internal surface area belongs to the solvent excluded surface, depicted white in Figure 3.16. The contact points of the rolling probe and the two atoms are forming small circle arcs on the atom surfaces bounding the toroidal patch.

Spherical triangles These triangles, coloured blue in Figure 3.16, are generated by the probe surface when the probe is in simultaneous contact with three atoms. Hence, the probe cannot roll any further without losing contact to at least one of the atoms (zero degrees of freedom). Thus, spherical triangles adjoin toroidal patches along all three edges.

These different geometric primitives lead to a much smoother visual impression of the surface compared to the space-filling and solvent accessible surfaces but, on the other hand, make the geometric description of the surface and, thus, the rendering much more complicated.

Like the solvent accessible surface, the solvent excluded surface closes small gaps and crevices but has the advantage that it retains the basic shape and extent of the van der Waals surface. This can be seen in Figure 3.14(c) where the solvent excluded surface is exemplified for a small protein. The solvent excluded surface is, thus, suitable for applications like docking and the analysis of protein-solvent systems since it does not inflate the atom radii, which would cause overlapping of neighbouring surfaces.

Since molecular dynamic simulation trajectories can contain up to several ten thousand frames, pre-computation of the solvent excluded surface for all frames, like it is traditionally done in available molecular viewers, is not feasible. In order to enable the rendering of these trajectories at interactive frame rates, a method which maintains the surface out-of-core is essential, which was a previously not satisfactorily resolved challenge.

Thus, a new approach for the interactive visualization of molecular surface dynamics is presented in this section, especially focusing on large MD simulation trajectories. Providing an out-of-core rendering solution, the intended real time analysis of arbitrary long trajectories can be enabled. By applying GPU raycasting to the implicit mathematical description of the *Solvent Excluded Surface*, the memory consumption was considerably reduced – compared to available molecular viewers using triangulation – thereby increasing the scalability of the method, along with gaining superior rendering quality.

The advantage of the typically applied triangulation of the solvent excluded surface is that modern graphics hardware is designed for fast rendering of triangles and the adaptive tessellation can be applied for an optimal ratio between visual quality and rendering speed. The major drawbacks of polygonal graphics is the high amount of storage needed for the triangulated surface, as well as the visual artefacts which arise in close-up views even for very fine tessellation, and the imperfections when using fast hardware-accelerated per-vertex lighting. Thus, we decided to employ a GPU raycasting approach, which is particularly feasible since the geometric primitives of the solvent excluded surface can be described by implicit functions, further detailed in the following. As described previously, raycasting of surfaces has the benefit that, compared to triangulated surfaces, only a small amount of data must be stored and transferred from CPU to GPU, since not all vertices representing an object must be passed to the shader, but only the specific values for the coefficients of the implicit function. By circumventing this traditional bottleneck of polygonal computer graphics, the higher costs for ray intersection tests are partially compensated, in some cases to the point of raycasting reaching even higher frame rates than polygonal graphics. Another advantage of raycasting is the high rendering quality, which is completely independent of the viewport since the exact mathematical description of the surface is used for image synthesis, though, inducing expensive intersection tests for each fragment of the resulting image.

Related Work on the Solvent Excluded Surface Rendering

The first algorithm to compute the solvent accessible surface analytically was presented by Connolly [Con83], therefore this surface is also known as *Connolly Surface*. Based on his work several methods to accelerate the analytical computation were introduced: Perrot et al. [PCG+92] presented improvements to Connolly's method by avoiding computations related to internal atoms which do not contribute to the molecular surface. Sanner [San92] developed the *Reduced Surface* as a basis for the generation of the patches. He subsequently improved the method in [SOS96]. Edelsbrunner and Mücke [EM94] introduced the *α-shape* which was later expanded to the *β-shape* by Ryu et al. [RPK07]. Varshney et al. [VBW94] described a parallel algorithm and Totrov and Abagyan [TA95] proposed a contour-buildup algorithm.

Of all approaches mentioned above, the *Reduced Surface* is especially interesting when dealing with dynamic data, since it can be efficiently updated piecewise as proposed in [SO97]. Nevertheless, more recent work like [ZXB07] mostly deals with high quality triangulations of the solvent excluded surface, not with the underlying computations.

With the increasing computing power of modern GPUs and the availability of high-level shader languages like Cg, GLSL or HLSL, raycasting on the GPU is

becoming a serious alternative to traditional polygonal rendering. Good results have been previously obtained for raycasting large numbers of quadrics on the GPU, e.g. [Gum03, KE04, dTL04, RE05, SWBG06]. Toledo et al. [dTLP07] stated in 2007 that iterative methods for root finding are more efficient than analytic methods when raycasting higher order surfaces like cubics or quartics. They also compared several iterative methods to the solution of Loop and Blinn [LB06] who used Bézier tetrahedra to define algebraic surfaces and an analytic root solver and attained up to 50 fps for 16,000 tori using the Newton-Raphson method. Singh and Narayanan [SN07], on the contrary, showed that modern GPUs are able to achieve comparable frame rates using analytic root solvers. An updated version was recently published [SN09].

In the meantime, a new approach for the visualization of dynamic molecular surfaces, based on contour build-up, was presented in [LBPH10].

Due to the requisites for time-dependent data and the fact that the geometry patches in the solvent excluded surface representation are all defined by quartic implicit functions, we finally decided for an algorithm using the reduced surface and doing GPU raycasting with an analytic root solver, both detailed in the following.

The work presented in this section was mainly implemented by Michael Krone as part of a diploma thesis and is published in [KBE09].

Molecular Surface Computation

As mentioned previously in this chapter, the analytical computation of the solvent excluded surface requires time-consuming computations – a naïve implementation would have a runtime complexity of $O(n^3)$ – hence, an acceleration method is crucial for fast construction. We chose Sanner's reduced surface (RS) [San92] as it combines fast and straightforward computation and provides the ability to update the surface piecewise. The definition of the reduced surface is directly connected with the definition of the solvent excluded surface stated in the previous section: When the probe is in a fixed position, i.e. in contact with three or more atoms, all these atoms are forming a polygon which is called a *face of the reduced surface* (RS-face). The edges of this polygon are called *edges of the reduced surface* (RS-edges), while the centre points of the involved atoms are called *vertices of the reduced surface* (RS-vertices). Under the assumption that a probe has simultaneous contact with at most three atoms, all RS-faces are triangles. This assumption can be made without loss of generality since all polygons can be subdivided into triangles. An example of the reduced surface of a small peptide is depicted in Figure 3.17. The reduced surface is only valid for a specific probe radius r_p, so if the radius is changed the reduced surface has to be recomputed. A more detailed description of the reduced surface can be found in [SOS96].

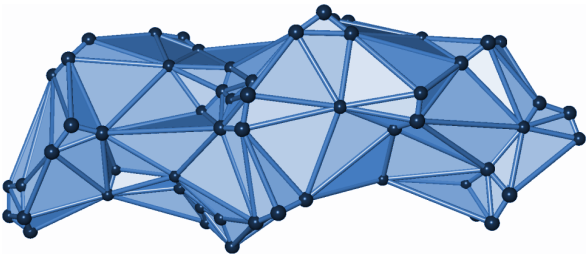


Figure 3.17: *Reduced Surface* of a tiny protein for a probe radius of 1.4 Å corresponding to water.

The primitives of the reduced surface (vertices, edges and faces) are directly related to the primitives of the solvent excluded surface, as already indicated by their definition. For each RS-face, a concave spherical triangle is generated, the RS-edges indicate toroidal patches and the RS-vertices represent the convex spherical patches. The equations for computing the parameters of these primitives of the solvent excluded surface are detailed in the original work by Connolly [Con83].

The reduced surface can be computed using the following algorithm outlined by Sanner [SOS96]: The first step is to find an initial RS-face, where the probe is in contact with three atoms and does not intersect with any other atom. Such a triple of atoms can be found by searching the leftmost atom $a_{min(x)}$ (i.e. the RS-vertex with the minimum x-coordinate) and computing the vicinity $V(a_i)$ of $a_{min(x)}$. The vicinity of an atom a_i is defined as all atoms with which a probe can theoretically be in simultaneous touch together with a_i. This includes all atoms which are inside a sphere with centre a_i and radius r_v:

$$r_v = r_{a_i} + r_p + max\left(r_{a_j}\right) \tag{3.1}$$

where r_{a_i} is the radius of the atom a_i, r_p is the radius of the probe, and $max\left(r_{a_j}\right)$ is the maximum atom radius.

For all triples $a_{min(x)}$, a_i, a_j with $a_i, a_j \in V(a_{min(x)})$, a potential RS-face is computed and the probe position for this RS-face is checked against intersections with all other atoms in the vicinity $a_k \in V(a_{min(x)})$. If no intersection occurs, an initial RS-face was found.

For each edge of the initial RS-face, an adjacent face can be computed by checking all potential adjacent RS-faces. Therefore, the probe defining the initial RS-face is rotated around the edge until it hits another atom which implies that the probe is in a fixed position again. In practice this can be done by computing the vicinity of the RS-edge $V(t_{ij})$, where t_{ij} is the centre of the torus being the trace of the probe rotating around the atoms a_i and a_j

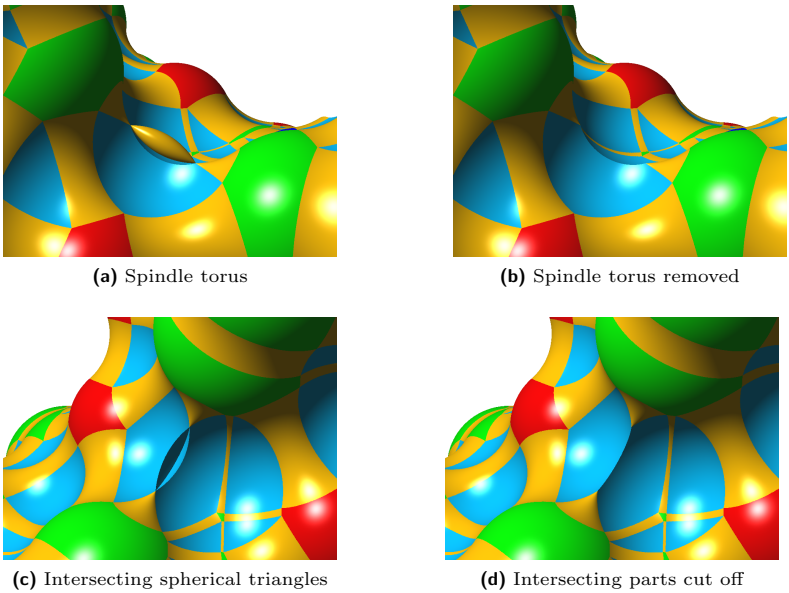
delimiting the RS-edge. The radius r_v of the vicinity for an RS-edge can be computed using Equation (3.2), where r_t is the outer radius of the torus:

$$r_v = r_t + max\left(r_{a_j}\right) \tag{3.2}$$

For each potential adjacent face a_i, a_j, a_k with $a_k \in V(t_{ij})$ the angle between the probe position $p_{ijk'}$ of the initial RS-face and the new probe position p_{ijk} is calculated. The wanted adjacent face for the RS-edge between a_i and a_j then is the potential face with the smallest angle and no further intersection tests have to be done for the probe of the newly generated RS-face. This operation is called the *treatment of an RS-edge* [SOS96].

The complete reduced surface is constructed incrementally by treating all new edges of each face as described above. The algorithm stops when all RS-edges are treated, i.e. each RS-edge is adjacent to exactly two RS-faces. By definition, the reduced surface is a closed surface.

Since by definition of the surface patches the solvent excluded surface can suffer from undesirable self-intersections shown in Figure 3.18, these *singularities* must be identified and taken account of in a subsequent processing step to ensure the correct display of the molecular surface. These singularities arise



(a) Spindle torus

(b) Spindle torus removed

(c) Intersecting spherical triangles

(d) Intersecting parts cut off

Figure 3.18: Undesired singularities due to self-intersections of the molecular surface and their removal.

either if the surface of a primitive intersects itself or if a primitive is intersected by another one. The first case occurs at toroidal patches if the distance between the torus centre t_{ij} and the centre of the torus tube (*major radius* R of the torus, i.e. the probe radius) is smaller than the tube (or *minor*) radius $r \equiv r_p$, which results in a spindle torus. The second case occurs if a spherical triangle is intersected by another spherical triangle. While the first case can be identified by a single comparison of the two radii, the second case requires a more intricate treatment. Thus, for each RS-edge, all probes in fixed positions cutting this edge must be detected and stored. Note that this test is only necessary for RS-edges where the first type of singularity (a spindle torus) was detected. Additionally, if an RS-face with the same RS-vertices exists, the probes of these two RS-faces must be checked against intersection from the other side, since intersections also occur if the probe can access a position where it is in simultaneous contact with the same three atoms from different directions (e.g. thin layers of atoms accessible from both sides).

The fast identification of the vicinity of a RS-vertex or -edge is a crucial factor for the speed of the aforementioned algorithms. Sanner proposed to use a *Binary Spatial Division* (BSP) tree [SOS96]. By contrast, we apply a uniform spatial subdivision. The RS-vertices are sorted into cubic voxels with a lateral length of $2r_{max} + 2r_p$ where r_{max} is the maximum atom radius and r_p the probe radius. The vicinity can then be obtained by only considering the RS-vertices inside a block of 3×3×3 voxels. The maximum number of vertices per voxel is bounded above and does not depend on the size of the molecule but on the probe radius [VBW94]. As the probe radius is fixed during computation, this number can be seen as a constant, therefore the vicinity of an RS-vertex can be obtained in $O(1)$, i.e. constant time.

Rendering the Molecular Surface

For the rendering of the solvent excluded surface, raycasting applies extremely well since the surface is composed of the small patches which can be described by implicit functions. Thus, the three types of primitives are generated by raycasting using GLSL shaders. The extensive root finding for the implicit functions has to be done per fragment, so that the number of fragments for which ray intersection tests have to be calculated are crucial for the resulting rendering times. Therefore, a minimal proxy geometry has to be applied for fast raycasting. Since all primitives can be easily circumscribed by relatively tight fitting spheres, a point-based approach was used. Additionally, as many computations as possible are done in the vertex shader in order to reduce the workload of the fragment shader.

The spherical patches are part of the van der Waals spheres that embody the atoms and can be delimited by a nearly arbitrary number of small circle

arcs, up to the maximum number of vicinity atoms. To simplicity matters the whole spheres can be rendered without impairing visual appearance if no transparency is applied to the surface because those parts of the van der Waals spheres that are not part of the solvent excluded surface are hidden by it and are thus invisible.

The rendering of the toroidal patches is more complicated. The surface of a torus with major radius R and minor radius r can be described by the quartic implicit function

$$\left(R - \sqrt{x^2 - y^2}\right)^2 + z^2 = r^2. \tag{3.3}$$

Fourth order polynomials can be solved iteratively as well as analytically. Since we need the inner surface of the torus and, thus, the second and fourth solution to Equation (3.3) instead of the first one, we decided for an analytical solver. Furthermore, iterative solvers hold the risk of divergence and suitable initial values have to be determined in order to end up with the wanted solution.

For the toroidal surface patches, we evaluated several analytical methods for quartic root finding – namely the Descartes, Neumark and Ferrari (stabilised and unstabilised, as proposed in [HE95]) algorithms – with the result that only the stabilized Ferrari algorithm was able to compute an exact representation of a torus on the GPU. Thus, we implemented the stabilized *Ferrari-Lagrange* method described by Herbison-Evans [HE95], which was also successfully used by [SN07], in GLSL. The numerical accuracy was enhanced by a translation of the base of the viewing ray towards the torus. According to the case of the spherical patches, the part of the torus which lies inside the molecule with respect to the solvent excluded surface is not clipped in order to reduce computation time. The toroidal patch that contributes to the solvent excluded surface is the inner part of the torus (represented orange in Figure 3.19) located between the two atom spheres. This part of the torus can be circumscribed by a sphere whose centre lies on the edge connecting the two atom centres. We call this sphere, depicted blue in Figure 3.19, the *visibility sphere*.

The radius r_{vs} and the centre c of the visibility sphere, with respect to the torus centre t_{ij}, are computed using Equations (3.4), where p is an arbitrary position of the probe while tracing out the torus, a_i and a_j the atom positions and r_i and r_j the atom radii respectively.

$$x = \frac{p - a_i}{|p - a_i|} \cdot r_i, \qquad r_{vs} = |x - c|\,, \qquad c = (c' + a_i) - t_{ij}, \tag{3.4}$$

$$\text{where} \quad c' = \frac{|p - a_i|}{|p - a_j| + |p - a_i|} \cdot (a_j - a_i)\,.$$

Only the section of the torus that lies inside the visibility sphere has to be rendered, the exterior parts are neglected. This is accomplished by testing if the second intersection between the ray and the torus lies within the visibility

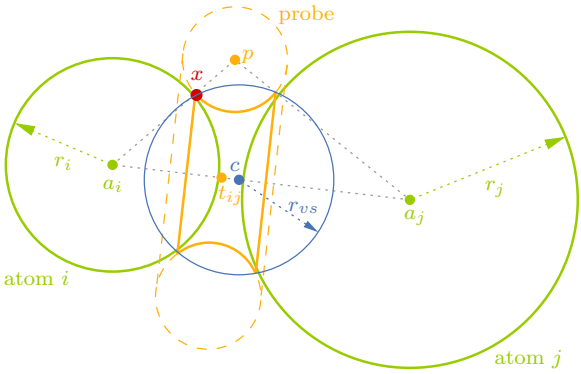


Figure 3.19: Sectional drawing of the visibility sphere with respect to the points x (exemplified in red) along the contact curve between the probe (orange dashed circle) and the protein atoms (green). The resulting visibility sphere is coloured blue.

sphere. In case of a spindle torus, the singularities previously mentioned and previously exemplified in Figure 3.18 must be handled as follows to obtain the visual result presented in Fig. 3.18(b). The spindle is enclosed by a sphere with radius $r_S = \sqrt{r^2 - R^2}$ and whose centre equals the torus centre t_{ij}. In case the intersection lies within this sphere, the fourth intersection with the torus which is located behind the spindle is used if this one also lies within the visibility sphere, otherwise the current fragment is discarded.

The third type of surface patch, the spherical triangle can be traced back to the spherical patches since raycasting a spherical triangle is basically equivalent to raycasting a sphere. For a concave spherical triangle the second intersection is used to get the inner part of the sphere instead the usually wanted outer one. The three great circles that trace out the spherical triangle define a plane, each. The sphere is, thus, cut with these three planes. In order to enable singularity handling, the centres of all probes cutting an RS-edge are written to a *singularity texture*. In the fragment shader, the spherical triangle is cut with the cutting probes of all three bordering RS-edges resulting in Figure 3.18(d).

For the rendering of each primitive, the centre point and the additional parameters needed for raycasting are sent to the vertex shader, where calculations identical for all fragments, like correct minimum point size, are performed. For an average sized protein, this results in a total of about 1 MB of data transferred to the GPU, compared to about 2 MB when using a low tessellation. Intersection tests and shading are subsequently performed in the fragment

shader. Thus, even for the largest proteins interactive frame rates of over 10 fps are reached for static data sets. Further details are given in the results section on page 115.

Dynamic Data Processing

The reduced surface is only valid for a particular configuration of atom positions. When dealing with dynamic data, the recomputation of the reduced surface may be necessary for each frame if one or more atoms are moving. If only a subset of the atoms is moving between two consecutive frames, the reduced surface can be updated piecewise. In our implementation we basically use the algorithm presented by Sanner in [SO97] for moving fragments of the protein (at most 100 moving atoms). Since we do visualise trajectories of full molecular dynamic simulations, the number of moving atoms is typically considerably higher. In order to accelerate the update of the reduced surface, we offer the option to filter the atomic motion. Positional changes below a user defined threshold are ignored and, consequently, movements where an atom oscillates around a fixed position (e.g. because of Brownian motion) but does not undergo essential positional changes, are ignored. Filtering can be disabled by setting the threshold to zero.

The algorithm for updating the reduced surface can be outlined as follows: All effective positional changes of atoms are registered and all RS-edges and -faces connected to these atoms are deleted along with the RS-faces whose probes intersected with one of the moved atoms. Afterwards the resulting holes in the reduced surface are closed by treating the RS-edges according to the algorithm previously presented. The worst case, apparently, eventuates if all atoms have moved and the complete reduced surface has to be reconstructed.

As mentioned above, the memory consumption is too high to precompute the solvent excluded surface for each frame of the whole trajectory in order to achieve fast rendering. Therefore, we decided to use a real time data streaming approach and handle the data on the fly (out-of-core rendering). The reduced surface, thus, has to be updated potentially in every render pass as described above. Low computation times are thus crucial to our approach.

Protein Alignment

Even with the methods described above, rendering large trajectories still poses a challenge if the protein is highly flexible, i.e. many atoms change their position per time step, or for very large proteins. Therefore, the protein alignment can be applied to overcome this problem. This technique treats the motion of the protein as a whole which often occurs during the MD simulation, but has no effect – or even a mistakable effect – on the subsequent analysis of the protein flexibility.

During the MD simulation, the protein may get translated or rotated within its bounding box. This motion could result in misinterpretation of the protein flexibility and unnecessarily increases the positional changes in the protein data. Thus, this unwanted motion can be filtered from the trajectory in a pre-processing step, leaving the internal motion of the protein untouched. A method meanwhile widely used was introduced by Kabsch [Kab76, Kab78] and bases on the minimization of the root mean square deviation (RMSD) between each conformation in the trajectory and a given reference conformation of the protein and, thus, leading to a superimposition of the trajectory conformations.

Results on Protein Surface Rendering

The major benefit of raycasting is the compact and efficient representation of the primitives. Each primitive is described by its centre and few parameters, which are already calculated during the reduced surface computation. Therefore, no further precomputation is needed for raycasting the solvent excluded surface – except the probe positions needed for the singularity handling of the spherical triangles. Their number easily exceeds the quantity of available variables for passing information to the fragment shader and, therefore, are written to the singularity texture. Thus, only the associated texture coordinates have to be passed to the fragment shader. By contrast, extensive computations are necessary if triangulation is used, especially when aiming at higher quality tessellations. This increase of computational time is particularly not acceptable since it not only constrains the interactive visualization of dynamic data, but also the visual quality of triangulation is inferior to raycasting since the geometry of the primitives is only approximated. By raycasting the implicit representation of the solvent excluded surface, an exact and crisp rendering is achieved, as shown in Figure 3.20.

Recently, a GPU raycasting of the *Molecular Skin Surface* (MSS) [Ede99] called *MetaMol*, was presented by Chavent et al. [CLM08]. Unlike the solvent excluded surface, the MSS is not defined by a rolling probe and only composed of quadric patches. But still the visual quality is analogous to our implementation. In the current version, MetaMol offers no specific support for dynamic data and is therefore unfit for the use with trajectories. Performance details on our method can be found in [KBE09] – also in comparison to the method of [CLM08].

We compared our approach to VMD [HDS96], Chimera [PGH$^+$04] and PyMOL, which use the program MSMS by Sanner [SOS96] to compute the solvent excluded surface, like almost all available molecular viewers. The advantage of MSMS is that it is quite fast and offers a high-quality triangulation with arbitrary tessellation. However, the drawback is, that it cannot update the solvent excluded surface but recomputes it completely for each frame. Furthermore,

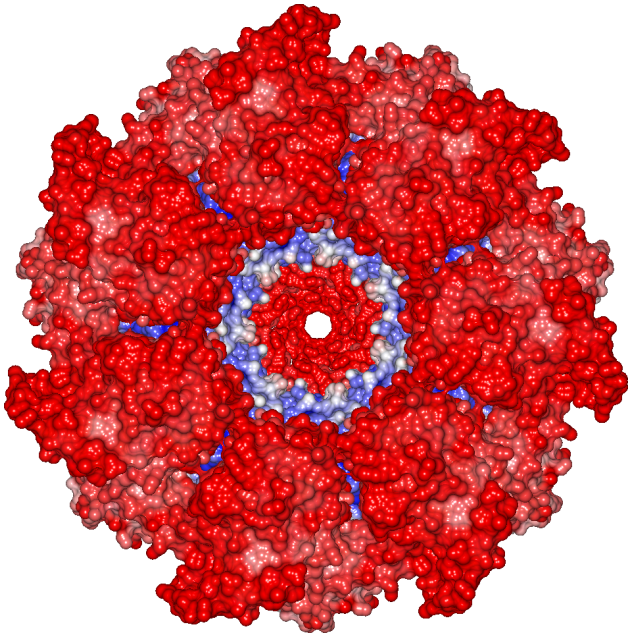


Figure 3.20: Solvent excluded surface of one of the largest proteins known (1AON) – previously shown in cartoon representation in Figure 3.13.

even when using the standard parameters resulting in a low tessellation the triangulation generated by MSMS consumes twice the memory compared to the implicit representation used in our work, and roughly doubles the computation time. VMD additionally offers the possibility to use the program SURF by Varshney [VBW94], but it also cannot update the solvent excluded surface. Because PyMOL attempts to precompute the geometry of the solvent excluded surface for every frame of the whole trajectory, it is virtually not possible to view larger trajectories at all. Even when selecting very few frames (<100), memory consumption and precomputational times are extremely high. VMD and Chimera, by contrast, compute the solvent excluded surface on the fly for each frame. Hence, both viewers take several seconds for rendering each frame, thus completely impeding trajectory analysis. A further disadvantage when using VMD is that the whole trajectory is initially loaded to the memory (like in PyMOL), which is infeasible when dealing with trajectories containing more than 1,000 frames. Our implementation, on the contrary, visualizes the solvent excluded surface of the trajectory out-of-core, thereby enabling the analysis of trajectories even if interactive frame rates are not always assured.

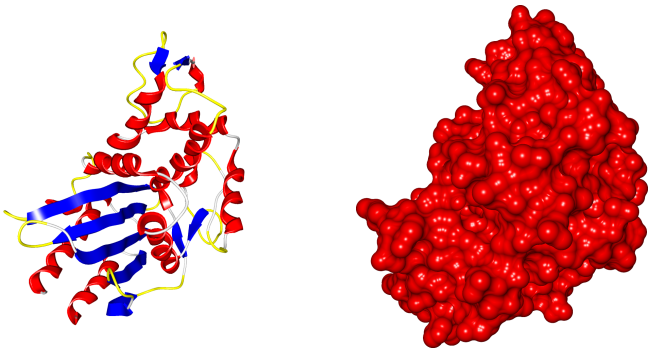
3.4 Dynamic Data for Time-Based Molecular Visualization Methods

The work presented in the subsequent two sections, the *Time-Based Haptic Analysis of Protein Dynamics* detailed in Section 3.5 and the *Visual Abstractions of Solvent Pathlines near Protein Cavities* presented in Section 3.6.2, both base on the same kind of molecular dynamic simulation of the same protein, a TEM-*1 β-lactamase*. The data is provided by researchers from the bio-chemistry field and the visualization methods described in the aforesaid sections are used by them for data analysis accordingly. In order to provide the appropriate background for these sections, the data used when developing these visualizations is explained here first.

TEM β-lactamases are enzymes that hydrolyse lactam antibiotics and, thus, transfer antibiotic resistance to pathogenic organisms. They are monomeric enzymes with a length of 263 amino acids. Figure 3.21 shows the protein that was analysed using the methods described in the subsequent sections in two different representations. TEM β-lactamases and their different phenotypes as well as their role in antibiotic resistance has been reviewed in detail in [Bra01].

To understand the function of this protein on a molecular basis and investigate the impact of protein flexibility thereon, a 3 ns MD simulation of the TEM-1 β-lactamase was performed.

A highly resolved structure at a resolution of 0.85 Å [MWS02] was used as a starting point. The protein was immersed in a 10 Å layer truncated octahedron periodic water box of 3-site water models (TIP3P waters) as depicted in



(a) Cartoon representation **(b)** Surface representation

Figure 3.21: Structure of the TEM-1 β-lactamase in cartoon and surface representation.

Figure 3.22. The system was subjected to energy minimization by steepest descent to remove initial close van der Waals contacts, followed by energy minimization by conjugate gradient. After energy minimization, the simulation was performed with periodic boundary conditions at a constant temperature of 300 K. After the equilibration of the system a production run of 1.0 ns was performed with a time step of 1 fs, whereas snapshots were saved every 250 fs for later analysis. Simulation and primary data analysis were performed using the AMBER7 software package [CPC+02].

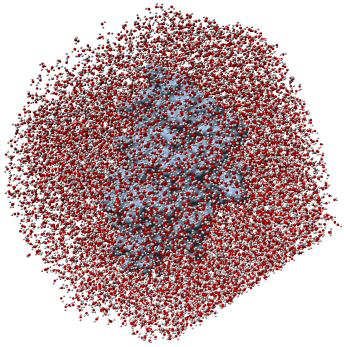


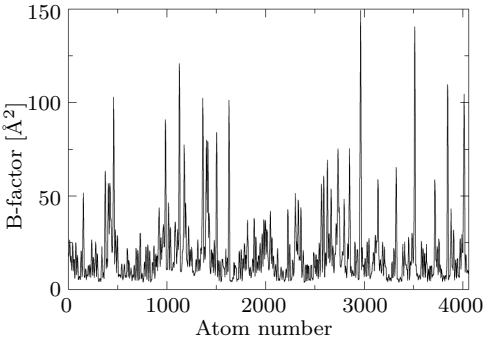
Figure 3.22: TEM β-lactamase in the periodic water box.

The resulting trajectory was stable and the backbone atoms deviated by only 0.9–1.1 Å from the experimental X-ray structure. In order to assess the protein flexibility, the root mean square fluctuations (in terms of B-factors) for all protein atoms were calculated. All coordinate frames from the last nanosecond of the trajectory were superimposed on the average conformation to remove overall translation and rotation.

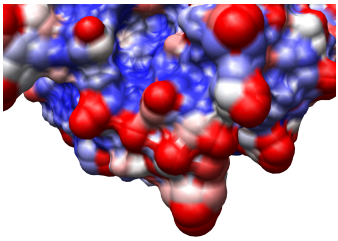
3.5 Time-Based Haptic Analysis of Protein Dynamics

Visualization of protein flexibility data from MD simulations usually involves plotting the flexibility data against the atom number as illustrated in Figure 3.23(a), static pictures of the protein structure with the degree of flexibility colour-coded and mapped onto the individual atoms as depicted in Figure 3.23(b) for an active site of a TEM1 β-lactamase, or representing the global correlated motions as arrows in a static picture. Another approach is the representation of the spatial probability density. But a direct time-dependent analysis of the flexibility other than looking at individual atoms or group of atoms in three-dimensional space has not been possible for a long time.

Therefore, we developed a new approach for evaluating a protein's simulated trajectory via time-based haptic feedback published in [BRB+07]. Molecular-scale trajectories are highly dynamic and, thus, pose new demands on haptic interaction. Using our haptic approach for tracking the atoms gives the user additional information and a more intuitive feedback than numbers or even plain graphics. We combined this tracking with touchable protein and atom surfaces together with data-driven surface properties as well as stereo representation and thus present a method for haptic interaction with the protein to give extended insight into the anisotropic motion of the protein.



(a) Classical diagram of B-factors per atom



(b) B-factors per atom colour-coded from blue (low) to red (high B-factor) and mapped onto the molecular surface.

Figure 3.23: Different visualization methods of protein flexibility via B-factors at the example of a TEM-1 β-lactamase.

We chose to couple a PHANTOM *Desktop Haptic Device* [Pha] to the open source molecular visualization tool VMD [HDS96] by using the SensAble OpenHaptics toolkit [IHZ05] whose HLAPI allows the programmer to directly set a virtual proxy position, as well as surface constraints, as defined in [RKK97]. The chosen haptic device, which is depicted in Figure 3.24, offers six degrees of freedom and the forces are applied to a stylus that the user holds like a pen. This stylus provides a button with hold and click events that are passed to the HLAPI. So, the user can touch the protein according to various representations together with different surface characteristics and can interactively select the atom for flexibility exploration. Locking the instrument tip to the selected atom, the user's hand is moved in accordance with the atom's conformational path. The user gets direct haptic feedback and thus insight into the anisotropic motion.

Figure 3.24: PHANTOM Desktop haptic device

3.5.1 Related Work on Haptics for Protein Dynamics

As mentioned before, beside VMD, there are various tools for protein visualization such as PyMol [DeL02], Amira [Ami], MoilView [SEZ95], MD Display [CSL96], and Chimera [PGH+04], but VMD is widely used and fitted our requisites best. All of these tools offer a wide range of representations, some of them structure-based, others based on the calculated surface of the proteins. For some representations, e.g. the solvent excluded surface detailed in Section 3.3.3, it is sensible to implement the option for exploring the protein surface haptically. Hence, there are several publications about haptic interaction with such surfaces, as detailed below. In principle, VMD does support haptic devices, but neither collision detection between cursor and molecule nor inter-molecular collision detection without resorting to a simulation backend.

A prototype Molecular Visualiser (MV) application, based on Web3D standards and extended by support of haptic interaction, such as frictional surfaces, was developed by Davies et al. [DJMH05]. To combine haptics with 3D molecular visualization, a Reachin Display was used. This tool is made for teaching and research alike, but might suffer from performance bottlenecks for large molecules, due to the use of relatively inefficient data formats and platforms.

Sankaranarayanan et al. [SWS+03] created multi-modality enhancements of tangible models by superimposing graphical information on top of existing physical models and by providing haptic feedback for the superimposed structures using a PHANTOM device. In our application scenario, however, no physical models are available, so we did not look into employing mixed-reality techniques.

In [MRF+96], Mark et al. integrated force feedback into a real-time virtual environment. Particular problems such as decoupling of the haptic servo loop from the main application loop for high-quality forces, benefits including haptic-textured surfaces, and device independence are addressed. Most of these issues are meanwhile solved by the OpenHaptics toolkit.

A lot of work in the field of haptics was done by Křenek et al.: In [KvK99] they focus on mapping quantities of chemical meaning to force fields. The special case of haptically mapping the required energy for switching between different conformations was studied in [KK99]. In contrast to that, we do not interact with the simulation itself, but only observe the dynamics of the complex system of a protein in solvent. A closely related work was [Kře00] where the haptic rendering of molecular flexibility was investigated, but bond flexibility was used in the model instead of real pre-computed conformational paths. Moreover, binding the haptics frequency to that of graphics limited feedback by computational power was addressed. In ensuing work [Kře01, PK05] improvements on conformational paths and offline calculation were made to enhance

the model's interactivity. But as in the earlier work, not the properties of a special trajectory were focused as in this work, but the switching between conformations.

Lundin et al. [LYG02, LBY05] presented a proxy-based approach to volume haptics instead of the isovalue based approaches. So they were able to add properties like friction and stiffness to the material, similarly to our approach. Likewise, volume haptics is the subject of [MCET05], where the docking of two proteins was focused and haptic rendering was combined with stereoscopic vision – but with only moderate success. In contrast to most of these papers, our approach aims at highlighting the dynamic properties of individual atoms in a single protein instead of investigating the interaction between several macromolecules.

The integration of haptic devices into molecular dynamics visualization tools is by far not limited to time-dependent analysis of protein flexibility. In recent years the applications of haptic devices in the field of molecular dynamics especially evolved into the fields of drug design [ZGAB08] and molecular docking [LYL06], as expected. But also the haptic interaction with molecular surfaces is still an active research field [SHL09].

3.5.2 Haptic Interaction with Protein Trajectory

We have extended VMD for our prototypical implementation. VMD already supports haptic devices, however the forces generated by the user are used as input for a simulation back-end (NAMD [PBW+05]), enabling the user to influence the simulation parameters. Feedback is the result of forces generated by the simulation influencing the selection of atoms the user is grabbing. No collision feedback is given with the standard tools. A powerful cluster or SMP machine is needed to ensure that the simulation can be updated in real time. We wanted the user to benefit from haptic feedback even when the visualization is decoupled from the simulation, and trajectory files are loaded off-line. Additionally, the user should be able to feel the surface of the different graphical representations of a molecule. Therefore, we modified the rendering subsystem of VMD to interface with the SenSable OpenHaptics toolkit directly. To locate the tip of the haptic device, we added a virtual representation – a small sphere – of the stylus position to the visualization. Since it has been shown that haptic interaction is much more effective when the visualization is stereo-enabled [FI04], we also make use of stereo rendering. Fortunately, VMD does support stereo rendering out of the box, however we had to implement a stencil stereo mode which is column-interleaved to be able to support our SeeReal autostereoscopic LCD [See].

Surface Interaction

For the navigation in the 3D space around a molecule and the selection of specific atoms it is very helpful that the user feels the molecular surface. Even with depth perception through stereo visualization, the actual distance from the cursor to the molecule surface is difficult to estimate precisely. However, if the user gets additional feedback from touch events, it makes the localization of the 3D cursor in relation to the atoms much easier and adds a certain consistency between the visualization and the user's input, since tactile feedback is known from daily life.

In order to give the user access to additional properties of a protein, we experimented with different surface properties as can be defined with the OpenHaptics toolkit. When setting the friction based on the type of the atom that contributed to the generation of a certain surface patch, the user can, for example, visualise another attribute on the surface and still find out where the atom boundaries on the surface are because of the perceived roughness changes. The charge of amino acids can also be mapped for a defined charge of the cursor: If the charges are inverted, the surface is sticky (high friction), while a collision event with the same polarity will result in a force applied along the surface normal to repel the cursor.

Time-based Feedback

The main contribution of the presented work is the time-dependent force feedback caused by the motion of selected atoms in the displayed molecule. Using the button on the stylus, the user can attach himself to the position of a chosen atom, which means that for static datasets the stylus of the haptic device will just be locked in place. When trajectory playback is activated, the user's hand will be dragged around in space according to the motion the selected atom performs over time. Even at the highest playback speed, this motion is still a bit discontinuous, as the different discrete positions are just handed over to the haptic device during the redraw of the scene. In order to achieve a more pleasant experience, the haptic rendering needs to be much faster than the visualization [Kře03], so we added interpolated position reporting at a fixed rate, regardless of the animation update speed. In practice 100Hz seemed adequate, as no discrete positional jumps could be felt in the datasets we have tested. We are currently using linear interpolation, but higher order interpolation could be easily implemented as well.

A drawback of this smoother movement – which also made us avoid higher order interpolation for now – is that even drastic directional changes are not perceived very clearly anymore. To overcome this limitation, we augmented the feedback magnitude by scaling the movement by a fixed factor (currently 4.0). This scaling is calculated relatively to the position p_g of an atom when it

is first grabbed, that means that all movement of this atom occurs in a subspace surrounding this position and is perceived relatively to it. To emphasise that movement, we applied the scaling also relatively to the position p_g, effectively scaling the traversed subspace. The applied scaling is unaffected by zooming, as it is calculated in the local molecule's coordinate system. The user can feel no discrepancy in distances since it is very difficult to mentally map the motion on screen absolutely to the motion in the real world, however the perception of small and smooth movements is facilitated.

3.5.3 Results on Haptics Analysis of Protein Dynamics

The integration of the haptic device into the molecular viewer VMD allows the user to accomplish two important kinds of analysis tasks in the field of molecular modelling and simulation.

First, mapping molecular properties onto the surface and coupling them to haptic feedback makes them directly tangible for the user. It is possible to intuitively explore the active site and get a sensory impression of e.g. the atomic charge distribution.

Second, tracking of motion allows the analysis of anisotropic protein dynamics at different levels, something that is not covered in standard analysis of MD trajectories. The usage of single-atom tracking with the PHANTOM is very intuitive. The user can directly feel motion in all three dimensions without perceptual issues as there would be for stereoscopic vision. Haptic feedback also does not suffer from occlusion or cluttering like visualization. The visualization would need selection-sensitive transparency to make the tracking of the focus (selection) in the context (protein) easier, whereas we can just use the visualization as context while the user feels what the focus is doing.

It is quite simple to distinguish areas of high motion from those with low activity when grabbing the atoms. The difference between the relatively rigid atoms of the active site and the more flexible atoms that constitute to the considered less rigid loop becomes evident to the user. However, the perception of the absolute quantity of movement is difficult. It is also unclear yet how many different magnitudes of motion the user can absolutely tell apart.

To filter out high-frequency movements and just pass the large changes to the user, we can make use of the smoothing option of VMD which also directly affects our haptic rendering. Using trajectory smoothing therefore allows the user to filter out the fast uncorrelated motions of individual atoms and amino acids and to directly focus on slow correlated motions between different parts of the protein or protein domains.

In summary, the presented method for time-based haptic rendering of molecular dynamics using a wide-spread application and commodity hardware allows

the user to compare the flexibility of various parts of a protein. We combined this with haptic rendering of protein attributes as surface properties to allow for better insight through a higher dimensionality in output. This may involve molecular attributes like atomic charges or hydrophobicity or any other one-dimensional property that can be calculated and mapped onto a per-atom or per-residue basis.

3.6 Solvent Visualization

The visualization of macromolecules and, thus, proteins is a busy research area for decades and a lot of methods, representations and specializations have been achieved and established. Unlike the protein visualization, the processing and visualization of the solvent in the protein-solvent systems is an often neglected task – notwithstanding the fact that the solvent plays a crucial role in protein structure, flexibility and activity and, thus, in many biological processes. According to this, there is a growing scientific interest in the protein's interaction with the solvent.

Below a certain threshold of hydration level, enzymes are inactive and non-functional [RC91]. Water mediates protein-protein and protein-ligand contacts by forming hydrogen bonds to side chain and backbone atoms of proteins [PS05] and to polar atoms of substrates and ligands [LWYW07]. Therefore, protein-solvent systems are being studied using various experimental and theoretical methods to gain insight into the interactions of solvent molecules, in particular water, with protein structure and dynamics – for reviews see [Mat02, Des05, PS06, Hel07].

X-ray crystallography has long been used to analyse water at protein surfaces, since crystal structures determined at high resolution provide a detailed picture of protein hydration [Nak04]. Analysis of multiple crystal structures of a protein or a protein family showed that certain positions on the protein surface and particularly in the interior of the protein are repeatedly occupied by conserved water molecules [BWT06]. Molecular dynamics (MD) simulations meanwhile provide a detailed atomic description of biomolecules enclosed by a shell of solvent molecules but still especially post-processing tools are lacking, that provide methods to focus on the solvent behaviour while analysing long-time molecular dynamics simulations.

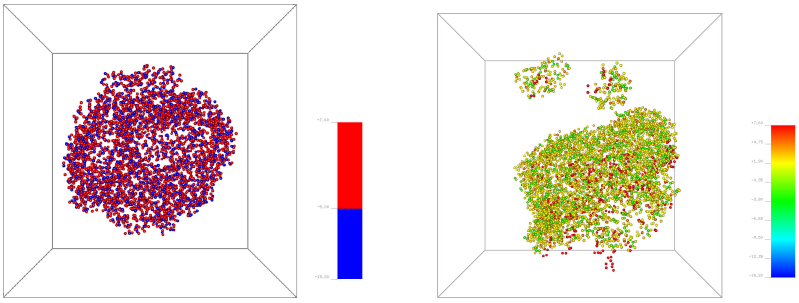
The protein-solvent interaction mainly takes place within a thin layer of solvent molecules, close to the protein surface. Thus, various filter mechanisms are a first step towards solvent analysis. The presented filters got implemented as part of a project thesis by Michael Ott.

3.6.1 Filtering of Solvent Molecules

The basic behaviour is common to all the described filters. The filters can be switched on and off using the graphical user interface and they can be combined with each other by boolean operators.

The most simple filter mechanism is filtering by atom radius. For this purpose an interval can be defined within which the atom radius has to range for the atom to be displayed. This can be especially useful if the visualization of certain solvent molecules shall be reduced to a special atom type for simplification and reduction of visual clutter. An example is the simplification of water by only showing the oxygen atoms.

Another simple but valuable filter method is the filtering by charge. According to the radius filter an interval can be chosen by the user and only atoms with charges within this range are displayed. In both cases the property by which the solvent is filtered can be additionally mapped by colour onto the remaining atoms. Either the interval is partitioned into two ranges and the atoms are coloured with two colours only – one for each subrange, or a colour gradient is used and the atoms are coloured according to their property value within this colour gradient. The two colouring styles are exemplified in Figure 3.25 in combination with filtering by atom charge.



(a) Binary colouring by charge **(b)** Gradient colouring by charge

Figure 3.25: Solvent filtered by atom charge in combination with two different colouring schemes.

A more sophisticated filter, in terms of supporting the interactive visualization of large time-dependent datasets, is the filtering by distance to the protein surface. Here the user can define a maximum distance up to which the solvent atoms are to be displayed whereas further away located solvent is cut off from representation as exemplified in Figure 3.26. While the previously presented

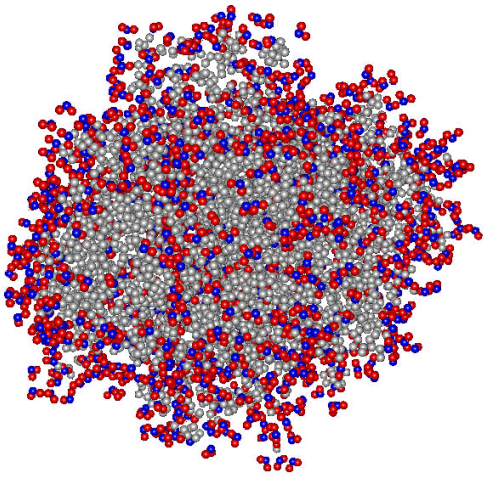


Figure 3.26: Solvent filtered by distance (3Å) and solvent atoms coloured by charge.

filters solely depend on the properties of the solvent atoms and every solvent atom has to be checked once for representation, the filtering by distance additionally depends on the protein atoms and, thus, not only the solvent atoms have to be checked but the protein atoms as well. Using the brute-force approach every solvent atom is checked against each protein molecule for distance measurement. As expected, this method does not reach update rates suitable for interactive visualization and time-dependent data. Thus, a voxel-based approach was chosen to cut down the computational effort and speed up the filtering. The data domain is voxelised according to the chosen distance for filtering and each atom (solvent as well as protein atoms) is assigned to the according voxel by its index. For each solvent atom only the protein atoms in the same voxel and in adjacent ones have to be considered for distance calculation. For further speed-up the voxels are flagged containing either solely solvent, solely protein atoms or atoms of both kinds, so that all adjacent voxels containing solvent atoms only can be skipped, too.

For large distances the voxelization results in few but huge voxels which cancels out the effect intended by introducing the voxel-based distance measurement. This is avoided by hierarchically subdividing the voxels and accordingly expanding the neighbourhood of voxels to be considered for calculating the distance to the protein surface.

The presented filters additionally can be combined with each other by the binary operators *and* and *or* leading to more specific filter options.

Although these filter options already offer very good support for data analysis additional filters are planned for future work. One of these additional filters is the filtering by motion. There, the motion speed of a solvent molecule has to be computed by evaluating the adjacent time steps in the trajectory. Since the fast moving solvent molecules are mainly uninteresting for the analysis of protein-solvent interaction this is an important option to reduce the subsequent data analysis to the potentially interesting subset of molecules.

The data reduction and analysis based on the motion of solvent is further investigated in the following section in a more specialised task for which main paths of solvent molecules are analysed.

3.6.2 Visual Abstractions of Solvent Pathlines

In contrast to the previously described filter methods, the method presented in this section and initially published in [BGB+08] focuses on visualising major paths of solvent molecules when analysing long trajectories and a large number of simulations such as a 50 ns (50,000 frames) trajectory with the protein surrounded by 25,000 water molecules as depicted in Figure 3.22. The benefit of the method in the analysis of long-time scale molecular dynamics simulations is shown in the application of water molecules entering and leaving cavities of two mutants of a protein which not only confirmed conjectures based on previous manual observations made by chance, but also led to novel insights into the dynamical and structural role of water molecules and their interplay with protein structure.

The method primarily allows to study the route and dynamics of the exchange of tightly bound internal water molecules with the bulk solvent. The proposed visualization represents the extracted paths of the solvent together with its dynamic properties such as direction and velocity in a static frame whereas these properties are analysed for the whole trajectory and subsequently mapped into a single frame. Especially by clustering similar pathlines with respect to designated properties, the visualization can be abstracted to extract and represent principal paths of solvent molecules in and through a pre-defined region of interest within the protein-solvent system.

Water molecules in the interior of a protein form an integral part of its structure and do also exchange with the external bulk solvent as a result of protein conformational fluctuations [GH00]. While the surface bound water was found to have an experimental mean residence time in the sub-nanosecond time scale, interior water molecules, in contrast, exchange slower within the nanosecond to millisecond time scale [OLW91].

Protein-solvent interactions have been studied for the TEM β-lactamase previously described in Section 3.4 using long-time scale MD simulations. An available protein-centric algorithm was applied to identify tightly bound wa-

ter molecules which form hydrogen bonds to two or more protein atoms of different amino acid residues [San04]. An interesting aspect of this interplay are the so-called water bridges. Our approach resulted in the discovery of water molecules located inside a cavity and involved in water bridges connecting a functionally relevant loop and the protein core. Being among the most highly-occupied water bridges throughout the whole simulation, they play a possible role in the stabilization of this loop. The question arose, how these water molecules, if at all, exchange with the bulk solvent.

Conventional tools like VMD [HDS96] can visualise the positions of the water molecules inside the cavity but, however, as the identification of the water bridges is a time-averaged process, all information about the motion and movement of the individual water molecules is lost. From plain visualization and animation of the trajectory, even with water molecules filtered by distance from the protein surface, it became clear that the amount of remaining water and its high fluctuation prevents further visual investigation. However, it was observed that water molecules enter or leave the cavity via at least two distinct routes, though, directly visualising the water molecules' pathlines in existing viewers was not feasible, as the resulting picture was cluttered by the unsteady pathlines of the water molecules.

The analysis of these solvent paths is enabled by the presented new abstract way to identify and visualise the pathways of solvent molecules, namely water in the simulation examined, in and around protein cavities from MD simulation trajectories. The visualization retains the directions of how the water molecules enter and leave the cavity and can give information on the velocities and residence times of water molecules following these routes.

Related Work on Solvent Paths

The visualization tools and frameworks previously discussed in Section 3.2 usually lack special features for processing the solvent and/or do not support time-dependent datasets. Unfortunately none of these solutions provides special analysis methods for the solvent surrounding the protein.

On the other hand, the trajectory analysis tools bundled with popular molecular dynamics packages like AMBER or GROMACS [BvdSVD95, HKvdSL08] per se only provide the possibility to extract the n-closest water molecules to a defined region (mostly a selection of one or more protein residues). However, renumbering of the water molecules in each frame prevents the identification of solvent molecules passing through one or more well-defined ROIs over the whole trajectory.

So far, the water distribution was represented either using a clustering or density-based approach. The clustering approach identifies water molecules occupying the same position in all trajectory frames, resulting in distinct cluster

points that represent water molecules tightly bound to the protein [MB04]. Density-based approaches voxelise the protein and represent the histogram of water molecules for each voxel. This leads to a density distribution of the solvent around the protein with high density peaks constituting areas of tightly bound water molecules [PMA98].

As both approaches require an initial superposition of the trajectory frames onto a reference frame, thus treating the protein essentially as inflexible, the cluster-based approaches fail to identify water molecules associated with flexible protein residues as they move along with the residues over the course of the simulation. In the density-based approaches this results in a smeared-out density distribution around the flexible protein residues. To circumvent these problems, methods have been employed which use a protein- or water-centric coordinate system instead of a global one and calculate water binding relative to the protein atoms [San04, HM02].

Although these representations of solvent distribution mentioned above only show a static view of the water distribution, they may be sufficient for many purposes. They allow for identifying water-filled cavities on the surface and deeply buried structural water molecules. In [BvG02] a tight hull volume of the cavity is identified and water molecules inside are detected. Similar to our approach, statistics on which exit these water molecules take when exchanging with the bulk are made, but these water molecules are neither tracked to examine and visualise the pathways inside the cavity nor the information if water molecules enter and leave the cavity by the same exit is provided. Besides the information about the exchange with the bulk, a visualization preserving the data's dynamics is desired, such as the routes water molecules take into and out of the cavity – a detail that cannot be seen in a cluster or density representation as described above.

VMD, after all, provides the functionality of tracking individual atoms over the whole trajectory, but which is not suitable for large trajectories with numerous atoms to track. For the task described here especially long trajectories with tens of thousands of time steps are processed, containing easily over 50,000 atoms in the solvent per frame. Already tracking a single atom over such a trajectory usually results in erratic pathlines extending throughout the whole simulation domain. However, this is useful to select atoms being in the cavity at a chosen frame and to determine whether this molecule leaves the cavity at all. To explore the more general behaviour of solvent molecules entering and leaving the ROI an approach providing a more specific interaction support is necessary.

In order to handle the numerous pathlines resulting from a large trajectory, we decided for an abstract representation of the main routes, clustering pathlines and visualising them as tubes, similar to the approach in [TvW99], where pathlines in vector fields are simplified for flow visualization. Nevertheless,

the wide range of techniques commonly used in flow visualization do not apply to our task, as we do not have a vector field and our attention is turned to explore the trajectory as a whole in a single frame.

Pathlines

The calculation and processing of the solvent molecules' pathlines was implemented as a pre-processing step in an independent tool decoupled from the viewer in order to keep the application most flexible and the resulting pathlines are stored in a file. Additionally, some statistical details are calculated by user's choice and written to an output file. This data includes information about the lengths of the extracted pathlines together with the average pathlengths and the exits taken when entering and leaving the cavity.

Doing so, at least the basic part of the tool can be used by a wider range of users, by loading the extracted pathline coordinates into another MD viewer, e.g. PyMOL, VMD or Chimera, as they are extensible by scripts or plugins. A python script for PyMOL is provided with the tool.

Since it should be possible to adjust the clustering parameters to the particular dataset during visualization, the clustering of the pathlines together with their representation described in section 3.6.2, as well as the protein's visualization can be chosen and changed interactively in the viewer, which will be detailed in the following sections.

Before filtering solvent molecules of the trajectory, a superposition of each frame in the trajectory onto a common reference frame, e.g. the first frame, is applied using the standard analysis tools provided by the AMBER molecular dynamics package [CCD+05]. The superposition bases only on the protein backbone atoms and, thus, removes the overall translational and rotational motion of the protein whereas the protein's flexibility is preserved.

Visualising the paths of all solvent molecules would result in a box stuffed with pathlines. The dynamic behaviour of solvent molecules close to the protein surface and in protein cavities is very different from that of bulk solvent molecules. The latter ones have no impact on the behaviour we want to explore. Hence, we want to visualise only likely relevant solvent molecules and the user can define a region of interest (ROI) within which the solvent molecules are tracked. Depending on the particular application scheme this region of interest can either be represented by simple geometric shapes like spheres, cylinders or boxes, or the geometry of the cavity is extracted by the external tool *PocketPicker* [WPS07], which is well-established in the application domain. PocketPicker detects cavities in proteins based on voxelization and subsequently filling up these voxel positions with solvent molecules. Finally, the geometry of a cavity is represented by specific solvent molecule positions along the voxels defining the cavity volume. The output of this tool can be

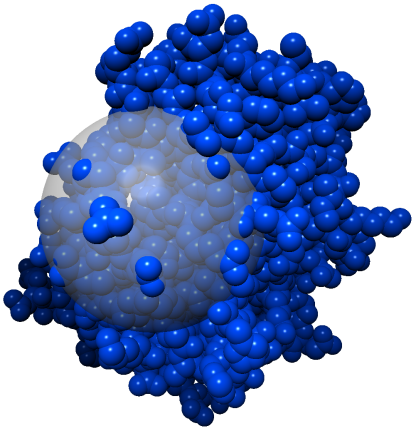


Figure 3.27: The protein illustrated in spacefill representation. The grey sphere marks the region of interest enclosing the cavity. Solvent molecules outside are neglected in further processing.

read in to our programme as the region of interest for the pathline detection. In the following examples, the ROI is defined by a bounding sphere with its centre S inside the protein's cavity and an appropriate radius to surround the cavity together with its possible exits as illustrated in Figure 3.27.

In order to get an initial set of pathlines in each frame of the trajectory the solvent molecules located inside the ROI are extracted. Reading the whole trajectory, we get all positions of solvent molecules within the selected area. This also includes molecules of the surrounding bulk solvent passing the ROI with high velocity and being of no interest for the phenomenon to explore. Therefore, the initially extracted paths are filtered and only pathlines persistent over a user-chosen minimum of frames are kept.

To ensure the visibility of contextual information, that is, water entering and leaving the cavity or translocating a protein channel, every pathline may be extended up to a user-chosen distance (either in Ångströms or number of frames) beyond the selection taking into account adjoining pathlines of the same solvent molecule as well as the trajectory's boundaries.

Rendering

Even the filtering of pathlines as described above does not prevent the view from being cluttered with too many individual pathlines. Therefore, additional processing is applied in order to get a more comprehensible abstract

representation – reduced to the principal paths traversed by the solvent molecules entering and leaving the protein region in question. However, creating an abstract representation always implies the risk of discarding or distorting relevant information and so leading to wrong conclusions. We therefore also implemented the direct visualization using pathlines for the filtered solvent molecules, both for reference and as a base for the abstract representations.

We implemented a direct and straight-forward rendering of all pathlines within the selected area of interest not only as a reference visualization for checking the plausibility of the abstract representations, as mentioned above, but also for the analysis of small datasets. When the number of relevant solvent molecules can be reduced to just a few, the direct visualization can be sufficient or even more appropriate to gain the required insight.

Since the extracted pathlines represent a static view of the trajectory, the dynamic properties have to be mapped to supplementary graphical attributes. Thus, the three most interesting aspects are encoded: Direction of the movement, velocity and the time frame. However, it is not necessary to represent the exact values but a qualitative impression is sufficient (e.g. whether a given position is at the beginning or at the end of the trajectory). The position in time is mapped to a colour gradient from red, over yellow and cyan, to blue. Figure 3.28 shows that this allows to roughly compare the time frames of neighboured positions. The velocity is mapped to the saturation. Since we are interested in slow movements, low velocity is mapped to high saturation,

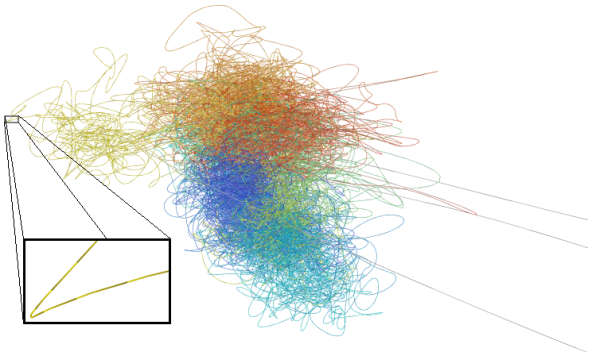


Figure 3.28: Properties mapped to the pathlines: The time within the trajectory is represented by a colour gradient from red (start of the trajectory) over yellow and cyan, to blue (end of the trajectory). An additional luminance ramp encodes the molecule's direction of motion as shown in the cut-out. The velocity is encoded in the saturation resulting in fast pathlines (right side) coloured grey.

whereas paths of high velocity are represented in grey. The velocity as well as the direction of motion is also shown by a small, repeated, and animated luminance gradient pattern.The pattern length is chosen in proportion to the local velocity in order to emphasise the mapping.

As a first processing step on the pathlines the chaotic small-scale motion of the molecules obscuring the principal movements is filtered for better perceptibility. Therefore, a rather simple but effective smoothing operator is applied to the pathlines with a filter kernel based on $\cos^2$ in the interval $[0, \frac{\pi}{2}]$ used because of its compact support. Although this does not reduce the total number of pathlines visualised, it simplifies the representation and, thus, helps to understand the structure of the dataset as can be seen in Figure 3.29 for a small region of the dataset.

(a) Original molecular paths **(b)** Smoothed pathlines **(c)** 50 ns trajectory with pathlines

Figure 3.29: The original pathlines (left) are smoothed (centre) to emphasise the principal directions of motion normally obscured by the molecule's chaotic small-scale fluctuations. The right image shows the examined protein region together with the resulting pathlines of a 50 ns trajectory.

For large datasets there are still too many pathlines for analysis as exemplified in Figure 3.29(c). Thus, the number of paths are reduced to prevent cluttering. Therefore the major paths are extracted by clustering adjacent pathlines with similar dynamic properties. This results in an abstract representation that provides the ability to gain deeper insight into the characteristics of the data. But still the clustering parameters are interactively adjustable.

The interesting parts of the dataset can be classified in two groups: Positions of conserved water molecules which are the areas of low velocity of the solvent molecules and the principal paths of the solvent molecules between these positions. So we extract the positions of low velocity as nodes of a graph and build edges using the pathlines' positions in between. As a first step our clustering method detects these positions of low velocity.

According to Figure 3.30 each sequence of successive solvent molecule positions (*pathline vertices*) w_i of a pathline p_i with velocity values below a defined

threshold forms a cluster $c_{i,n}$ of its own, where n reflects the fact that on each pathline several clusters can emerge. Spheres are chosen to represent the cluster bases. The positions of the spheres are given by the mean positions of all clustered pathline vertices and the sphere radius $r_{c_{i,n}}$ is chosen for the sphere to cover 90% of all of these vertices.

The second step of our clustering method merges overlapping spheres to new clusters c'_k, coloured green in Figure 3.30. All pathline segments connecting two clusters c'_n and c'_m form a directed edge between them. Finally, all edges between a pair of clusters are grouped into a single edge of the abstract representation. The number of pathline segments forming this final edge is defined to be the weight of this edge.

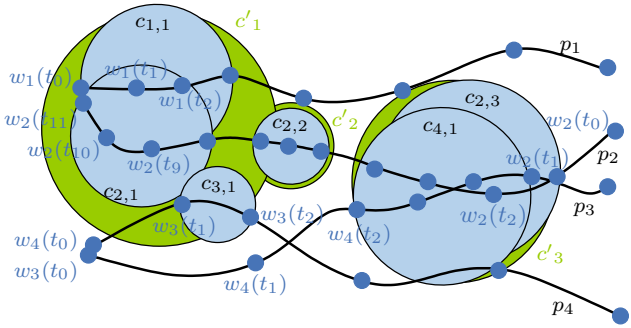
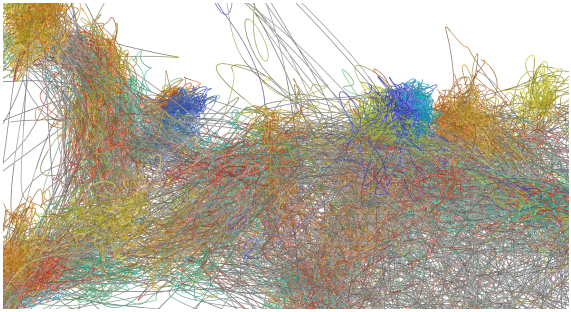


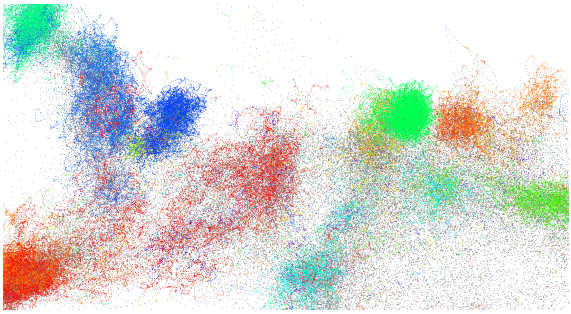
Figure 3.30: Pathline clustering: Vertices w_i (blue) along a pathline p_i with sufficiently low velocity values form a spherical cluster base $c_{i,n}$ (marked in light blue). Overlapping cluster bases define a new cluster c'_k (green) connecting these pathlines.

Additionally, helper clusters are defined at each end of all pathlines. These allow for creating edges between the previously detected clusters at the conserved water positions and the bulk solvent at the border of the selected area of interest. These helper clusters are merged together based on their distance using a threshold. To further reduce the clutter of the images, edges with a weight of only one are (optionally) removed, since these paths are usually not of further interest in data analysis. Figure 3.31 exemplifies the approach and shows the individual pathlines and the resulting abstract representation together with the helper clusters of the same dataset.

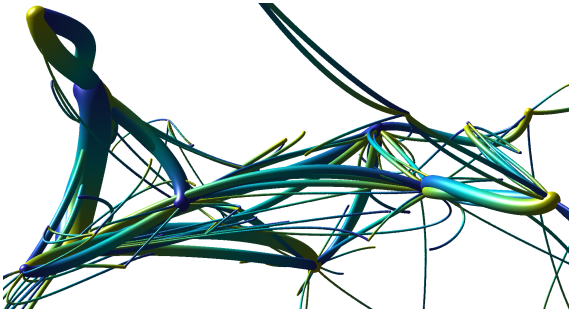
Straight lines connecting the clusters would produce unintentional overlaps since edges are rendered as tubes with the edge weight encoded as tube radius. Additionally, edges with a smaller weight could be completely occluded by thicker ones. This problem is overcome by the use of cubic Bézier curves



(a) Smoothed original pathlines



(b) Detected clusters

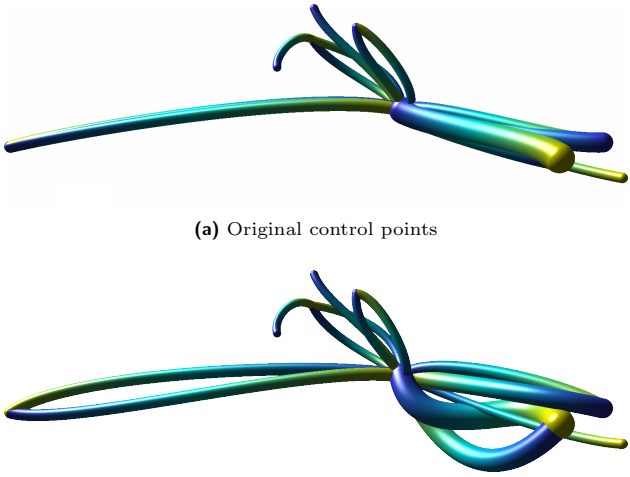


(c) Abstract path representation

Figure 3.31: Abstraction of pathlines. Figure (a) shows all pathlines, (b) the vertices of all pathlines, coloured according to the detected clusters of slower motion. Positions of conserved water correspond to the larger clusters. The lower figure (c) represents the extracted principal paths.

to represent the edges. Each curve is fitted through the vertices of all pathline segments forming this edge and therefore is a good trade-off between the original trajectory and the principal path of the solvent molecules. In case two clusters are connected by two edges of opposite direction the two tubes representing these edges would still overlap. So, the inner control points of the Bézier curves are varied depending on distance and edge weight. For each of these control points the scaled distance vectors are accumulated to all line segments between all inner control points of all other curves. The distance vectors, pointing from the line segment to the control point, are being normalised before they are scaled, depending on the length of the distance vectors and the weights of the corresponding edges. The smaller the distance is, the larger the scaling factor is chosen, so that edges close together affect each other more than edges which have a large distance to each other. The impact of this method is depicted in Figure 3.32. The direction of the edges is shown using a yellow to blue colour gradient.

The graphical primitives used for the abstract representation of the pathlines, namely spheres and cylinders, were created by point-based GPU raycasting for optimal visual quality and performance. These are used for the abstract representation as well as for some elements of the protein rendering, like in the stick, ball-and-stick and spacefill representation discussed in Section 3.3.1.



(a) Original control points

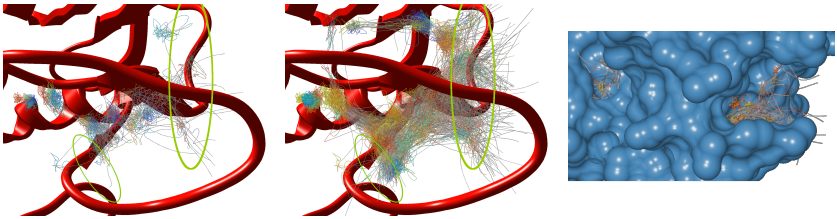
(b) Inner control points varied

Figure 3.32: Variation of inner control points of Bézier curves to prevent intersections and enclosures.

Results on Solvent Paths

The pathline extraction and visualization method proposed in this section has been applied to study the interaction of water molecules with the protein TEM β-lactamase, which plays a major role in the antibiotic resistance of gram-negative bacteria. According to Section 3.4 the main datasets used to evaluate our approach are 50 ns MD simulation trajectories of two variants of this protein surrounded by approximately 25,000 water molecules. A special scientific focus lies on the interplay of water molecules with parts of the protein structure.

Our novel approach to identifying and visualising solvent pathlines was applied to the MD trajectories, so that all pathlines of water molecules that pass through the region of interest (the water-filled cavity) during the simulation were identified. Figure 3.33 shows that the direct rendering of all pathlines within the selected area confirmed the prior observation that two distinct exits (marked in green) exist through which water molecules can enter or leave the cavity. One route is the obvious opening of the cavity (on the right side of the images) that was also visible in the initial crystal structure. The additional route (on the lower left side of the images) proceeds through a dynamically formed opening in the middle of the loop and was not evident in the initial crystal structure. From the direct rendering of all pathlines and the statistical output of the pre-processor it is evident that fewer water molecules follow this route. But to gain a better understanding of the flow inside and through the cavity a more abstract representation is needed.

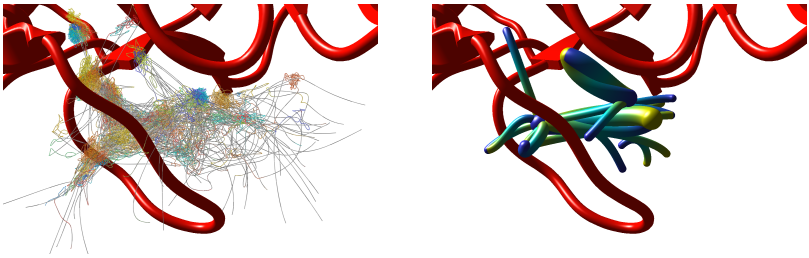


(a) 5 ns trajectory (5000 frames)

(b) 50 ns trajectory (50,000 frames)

(c) 5 ns trajectory with surface representation

Figure 3.33: Two trajectories of the same dataset. The shorter 5 ns trajectory (a) in the upper image could tediously be evaluated manually using existing tools. However, results gained on such short datasets must be judged with care, since interesting values depend on the frequency of the water exchange. The 50 ns trajectory (b), on the other hand is far too cluttered and too complex to be efficiently evaluated with existing tools. (c) shows the solvent paths together with the solvent excluded surface of the protein.



(a) Individual pathlines **(b)** Individual pathlines

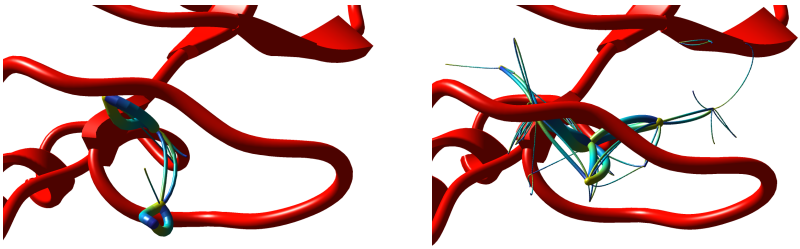
Figure 3.34: Structure extraction of solvent paths. (a) shows all solvent molecule pathlines, (b) the extracted main solvent paths. In both representations the same structure of solvent motion can be detected.

Figure 3.34 shows that this abstract visualization (b), where the sequences of tubes illustrate the principal paths of solvent motion, represents the real distribution of pathlines very well, as the structure of the tubes can also be observed in the image showing all pathlines (a).

The visualization of the clusters of slower motion (coloured separately in the previous Figure 3.31(b)) concur with the positions of the conserved water bridges detected in the previous analysis, and was therefore a confirmation of the chosen clustering approach. The tubes representing the main paths of faster motion between these clusters help to follow and comprehend the movement of water molecules between conserved positions inside the cavity. Even though not all individual paths are shown, the principal structure of the motion of the solvent is represented (Figures 3.31 and 3.34) and allows interpretation of the data. It becomes evident that there is no straight route for a water molecule once it has entered the cavity, rather it seems that water molecules quickly exchange between their relatively long stays at conserved water bridge positions. Another interesting fact is that the solvent molecules' motion inside the cavity is rather structured compared to the chaotic motion in the bulk solvent area.

Detailed performance results especially for rendering can be found in the original paper [BGB+08]

The presented approach, thus, allows to visually analyse the solvent molecules' pathlines on an abstract level, supporting long trajectories with tens of thousands of time frames. Different levels of abstraction are provided facilitating the exploration of solvent-protein-interaction on various levels of detail. The clusters observed in the abstract representation of solvent pathlines within the region of interest confirm the positions of conserved water found with other



(a) Original protein **(b)** Mutant

Figure 3.35: Solvent paths of 50 ns trajectories of the original protein and its mutant in comparison. The different solvent motion behaviour can be easily seen in the abstract representation of the solvent paths.

approaches but additional information about the dynamic behaviour of the solvent molecules is provided. In the given scenario, the analysis of functionality and protein-solvent interaction of a protein and its mutant, geometry processing played a key role in data analysis. Due to the presented visual abstraction of solvent paths the different structure of the solvent motion within the region of interest at the original protein compared to its mutant became obvious – depicted in Figure 3.35 – and the transient forming of a second entry to the cavity could be observed. Thus, we were able to substantiate conjectures based on previous observations on particular solvent molecules in solvent-protein interactions as well as to get deeper understanding of the protein-solvent interaction. The further insight into the simulation results on the part of bio-chemistry is detailed in [BP09].

Although the examples in the presented approach are focused on analysing the motion of solvent molecules in a protein cavity, it was also successfully applied to the analysis of solvent motion between several protein molecules within the same dataset as well as the solvent motion in a partially blocked protein channels.

3.7 Hyperstreamlines for Diffusion Tensor Imaging

A further step towards abstraction of geometries in visualization applications is the representation of diffusion tensors, an important technique in medical visualization. Diffusion tensor imaging (DTI) is a magnetic resonance imaging (MRI) technique with increasing popularity especially in neurosurgery. This method provides information about the diffusion properties of molecules in tissue instead of information about quantity and linkage of hydrogen. This is of

special interest since the underlying cell structure influences these properties in a way such that strongly aligned cells restrict the diffusion to an anisotropic behaviour. As important neuronal pathways feature these cell characteristics, one can infer about the occurrence of such major white matter tracts based on DTI data. From the medical point of view, this is particularly interesting since the protection of such major white matter tracts is of utmost importance during intervention. Therefore, visualization of the tracts got increasingly relevant to diagnosis and planning over the last years. However, many visualization methods are not capable of representing all tensor information in a comprehensive manner. This is disadvantageous since diagnosis is improved by the availability of additional information.

Our approach published in [RBE+06] uses the diffusion tensor eigenvectors to generate oriented ellipses at the sampled points, which are then linearly interpolated to form tubes connecting the sampling points. Together, the tube sections form hyperstreamlines as shown in [DH93]. Instead of creating a mesh from these tubes on the CPU, we just upload the orientation and size parameters of each tubelet to the graphics card and use sphere tracing [Har96] to generate the surface on the GPU. Since hyperstreamlines in medical visualization are most useful when displayed in the context of a volume representation of the surrounding tissue, we want to minimize data transfer as much as possible. That way, the system bus bandwidth can be used to upload volume data in case of time-dependent or bricked datasets. Since our approach does not access textures at all, the volume visualization can use all of the available bandwidth on the graphics card to keep performance as high as possible. On the other hand, our approach to the rendering of hyperstreamlines relies on the computational power that is available on current GPUs but is not subject to heavy load from direct volume rendering approaches.

3.7.1 Related Work on Diffusion Tensor Imaging

Several strategies have been applied to the field of DTI. The mapping of tensors to scalar values and displaying them in a separate slice is a well-known method for diagnosis purposes [ERMB00, PAB02]. For a more comprehensive visualization of DTI data, volume rendering was proposed by Kindlmann *et al.* [KW99, KWH00] who utilised textures and transfer functions to represent the tensor properties. To investigate the features of the tensor per voxel directly, glyph-based approaches have been employed [WMM+02, Kin04]. They represent each tensor independently by shape, size, colour, and other attributes of a glyph placed at the corresponding voxel position. These direct visualization techniques are useful for a detailed inspection of the DTI dataset. However, their weakness is the missing connectivity of the data which prevents the analysis of complete pathways.

To overcome this problem, streamline tracking techniques were adapted to DTI processing [BPP+00, LAS+02]. Additional improvements have been achieved by the use of streamtubes [ZDL03] and hyperstreamlines [DH93, WL01] since streamline techniques are limited to vector fields which can be partly compensated by the use of three-dimensional objects. Finally, neuronal pathways can be extracted and visualised as surfaces which support an intuitive understanding of the data [ESM+05].

Generating parametric surfaces on the GPU has been investigated in several publications, be it for explicitly generating geometry in vertex buffers [MP03, LH04], be it for raycasting the surfaces directly on the GPU, as has been done for ellipsoids [KE04, Gum03] and application-specific glyphs [RE05], to mention just a few. Sphere tracing [Har96] has been reproposed for displacement mapping on graphics hardware [Don05].

An approach very closely related to our own was proposed in [SGS05], however the tubes generated by their method are limited to circular cross-sections, as opposed to our elliptical cross-sections. This trade-off allows for the rendering of the resulting tubes using splatting techniques that yield much higher frame rates than our approach at the cost of a lower information density.

3.7.2 DTI Data

For the computation of diffusion tensor data, six diffusion-weighted images with different gradient directions were measured in combination with a reference image, measured without any gradient. Based on this set of images, the real-valued symmetric second-order tensor can be determined [WMM+02]. This tensor represents the diffusion characteristics of hydrogen averaged over the volume of the corresponding voxel. The eigensystem of this tensor can be evaluated, resulting in three real eigenvalues and the corresponding orthogonal eigenvectors.

The first step towards hyperstreamlines is the tracking of the related streamlines. Classical integration schemes such as Euler or Runge-Kutta, discussed in Section 1.2.2, are applied to determine streamlines through a vector field. The required input vector field is derived by a simple reduction of the tensor to its principal eigenvector. The loss of information is partly compensated by introducing a threshold for tracking. A possible parameter is fractional anisotropy (FA) which is a measurement for anisotropy [BMP+01]. It serves as stop criterion to terminate tracking when running into areas of reduced anisotropy with low probability for neuronal pathways. In our approach all voxels above the specified FA threshold were used as seed regions. As soon as the tracking enters a voxel with an FA value below the threshold it stops. A resulting whole-brain tracking can be seen in Figure 3.36. Afterwards, subsets of fibres were selected using manually defined regions of interest.

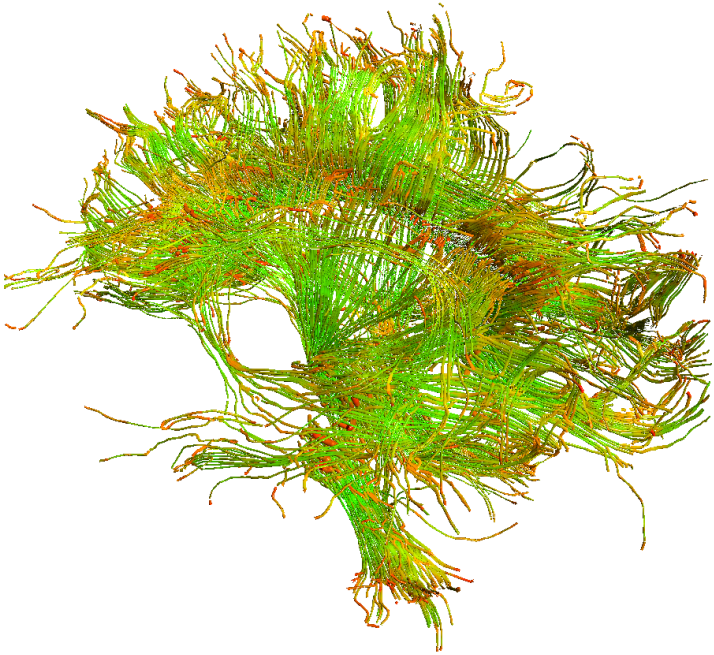


Figure 3.36: Hyperstreamline visualization of a whole-brain tracking. Green segments indicate tracking in reliable regions with high anisotropy, yellow tube sections suggest fibre crossings. The absence of red segments in the central parts of the hyperstreamlines documents the correctness of the fibre tracking algorithm.

The drawback of streamlines in the context of DTI is certainly the restriction to a vector field. Features of the tensor field like torsion or the minor eigenvalues cannot be displayed with this method. To overcome this problem, hyperstreamlines are applied which are capable of visualising such attributes. The eigenvalues and eigenvectors of the tensor at each base point of the streamline serve as basis for the twist and the semi-major axes of the segments of the hyperstreamline.

3.7.3 Tubelets

In order to visualise the tensor field, the data is mapped to a special kind of tubelet with a shape defined by ellipses within a slice plane at each end. These ellipses are determined by their semi-major and semi-minor axis and

their rotation around the x axis. The shape along the tubelet is determined by linear interpolation as further described in this section on page 143.

Taking into account the special needs of our given data, a local coordinate system is introduced, and some more complicated customizations on distance measuring and rotation interpolation are required, both detailed in the following.

Definition of the Local Coordinate System

For each tubelet a local coordinate system is defined, depending on the two ellipses that define the tubelet's shape. The ellipses are defined by an eigensystem each, where the normalised eigenvectors e_i are sorted by their corresponding eigenvalues v_i in descending order: $v_1 > v_2 > v_3$. As e_1 is the hyperstreamline's direction, the two minor eigenvalues define the length of the ellipse's semi axes and the corresponding eigenvectors define the corresponding direction of each semi axis. The signs of the eigenvectors are chosen in a way such that the eigensystem is a right-handed orthonormal basis (as mentioned in Section 3.7.2) that can be used directly to define the tubelet's local coordinate system.

The local x-axis is given by the tubelet's axis x_t, connecting the midpoints p_l and p_r of the two defining ellipses with $x_t = (p_r - p_l ht)/\|p_r - p_l\|$, and the other two axes are derived from the left end ellipse as follows: The ellipse's first eigenvector e_1 is to point in the same direction as the tubelet axis. This is done by rotating the eigensystem around $e_1 \times x_t$ by an angle of $\arccos(e_1 \cdot x_t)$. Then the rotated eigenvectors e_2 and e_3 define the tubelet's local y-axis y_t and z-axis z_t respectively.

Geometrical Definition of the Tubelets' Shape

Geometrically defining the shape of a tubelet first requires some considerations about ellipses in the yz-plane: Assuming r_1 and r_2 to be the two semi axes, ellipses are usually described by the Cartesian equation

$$\frac{y^2}{{r_1}^2} + \frac{z^2}{{r_2}^2} = 1 \; . \tag{3.5}$$

Our ellipses are defined within the yz plane and may be rotated around the x axis. Therefore a representation of the ellipse corresponding to the polar coordinates of a circle, as illustrated in Figure 3.37, is easier to handle. Additionally, we need the surface normal later for correct lighting, which is also easier to calculate in polar coordinates.

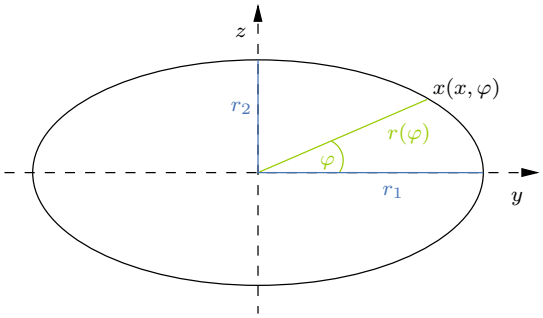


Figure 3.37: Ellipse in polar coordinates $x(x, \varphi)$.

Let φ be the angle to the y axis, then the ellipse can be given in polar coordinates by

$$x(x, \varphi) = \begin{pmatrix} x \\ r(\varphi)\,\cos\varphi \\ r(\varphi)\,\sin\varphi \end{pmatrix} \tag{3.6}$$

with

$$r(\varphi) = \frac{r_1\, r_2}{\sqrt{r_1\,\sin^2\varphi + r_2\,\cos^2\varphi}} \tag{3.7}$$

determined by plugging the coordinates (3.6) into the Cartesian equation (3.5).

Rotating the whole ellipse within a plane of constant x around the x-axis by an angle ρ changes (3.6) to

$$x(x, \varphi) = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x \\ r(\varphi)\,\cos(\rho + \varphi) \\ r(\varphi)\,\sin(\rho + \varphi) \end{pmatrix} . \tag{3.8}$$

The tubelets are defined by an ellipse at each end and are centred along the x-axis of their own local coordinate system. These ellipses may vary in the length of their semi axes and in the rotation along x, and they are located at $(-l/2, 0, 0)$ and $(l/2, 0, 0)$ respectively, where l is the length of the tubelet as depicted in Figure 3.38.

For each x value along the tubelet the semi axes and the rotation angle ρ of the ellipse are calculated by linear interpolation from the properties of the ones at the tubelet's ends.

The tubelets are connected to each other in order to get longer tubes, so the cutting planes E_l and E_r at the ends of each tubelet have to be specified. In order to approximate curved tubes, these planes may be rotated arbitrarily, as illustrated in Figure 3.38. We will go into more detail about this in the following, taking into account special needs arising with these properties.

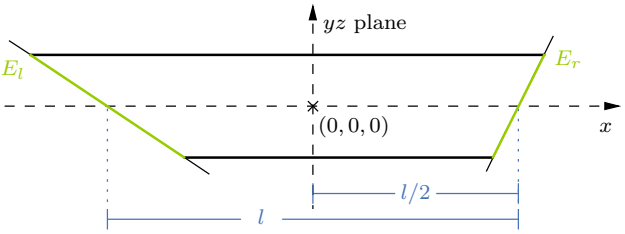


Figure 3.38: Definition of a tubelet along x, limited by an ellipse at each end: E_l and E_r.

Geometrical Background for Raycasting

In order to raycast each tubelet's surface we need the intersection point of the eye ray with the tubelet. As this intersection calculation leads to an equation of degree four we decided to adopt the sphere tracing algorithm presented in [Har96]. Starting at the eye's position we step along the eye ray, determine the current distance to the tubelet's surface, and take this distance as the next step size to close in onto the tubelet. This procedure is repeated until the distance falls below a specified threshold ω. In most cases this works fine but some rays require additional adjustments which will be explained in Section 3.7.4.

From the current position, the shortest distance to the tubelet has to be calculated in order to get the new step size. Due to the rotation of the tubelet along x, the correct shortest distance d_{corr} to the tubelet's surface – which is measured along the surface normal N – cannot be easily calculated as it would also lead to an iterative and time consuming way to find the solution. To avoid that, we decided to substitute the shortest distance by a close solution: We assume the x value to be the same as the one of our current position and calculate the distance d_{tmp} to the ellipse in this common plane normal to the x-axis and through the current position as depicted in Figure 3.39:

$$d_{tmp} = \sqrt{{p_y}^2 + {p_z}^2} - r(x, \varphi) \tag{3.9}$$

where p_y is the y-component of p and p_z the z-component respectively, and with r satisfying (3.7) within the current plane of constant x value. In the next step we take the point d_{tmp} units further along the ray as our new current position. In case $d_{tmp} < 0$ we already intersected the surface and have to step backwards, which is equivalent to walking along the ray in negative direction and needs no special considerations.

In order to get a longer tube which is more flexible than a single tubelet we connect neighbouring tubelets at their cutting planes E_l and E_r. As men-

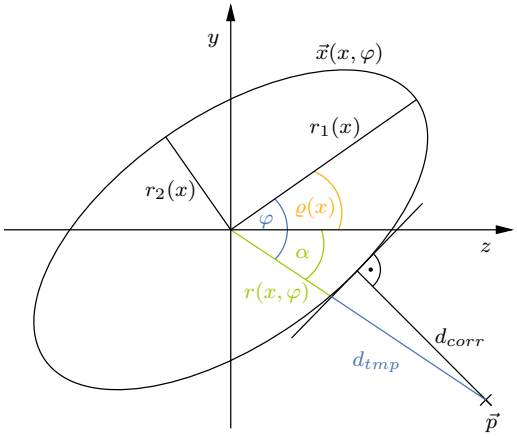


Figure 3.39: Distance measurement from the current position p to the rotated ellipse $x(x, \varphi)$ within a plane of constant x.

tioned before, the planes are not necessarily normal to the local x axis when approximating a curved tube. These sloped tubelet ends cause further considerations about the tubelet's properties: Merging the tubelets to form larger tubes also raises the problem of rotation consistency at the interconnections. Consistent properties of the ellipses are only guaranteed in slices perpendicular to x_t that do not contain parts of a neighbouring tubelet, because otherwise there are two inconsistent interpolation parameters belonging to the two tubelets respectively. So we have to restrict the interpolation up to the point q as exemplified in Figure 3.40 for the tubelet's left end.

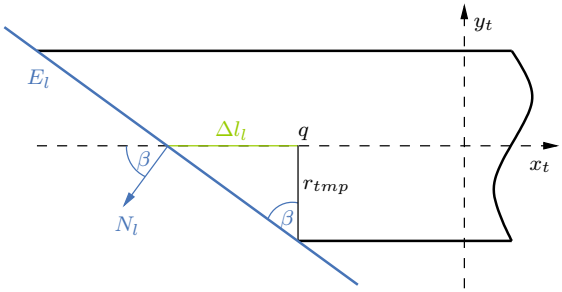


Figure 3.40: Correction of distance and interpolation range. E_l is the left cutting plane, N_l the corresponding normal vector.

To restrict the interpolation to the correct range of x values we need to calculate the distance $\Delta l = r_{tmp} \tan \beta$ with β being the angle between the tubelet's local z-axis z_t and the intersection line of the cutting plane and the plane normal to the local x-axis x_t, which is $\beta = 0$ for the left end and $\beta = \rho_r$ for the right end – due to the definition of the local coordinate system as described in 3.7.3 – and r_{tmp} satisfying (3.7) with $\varphi = \beta - \rho(x)$ leading to

$$r_{tmp} = \begin{cases} r_{1\,l} r_{2\,l} / \sqrt{r_{2\,l}} & \text{(left end)} \\ r_{1\,r} r_{2\,r} / \sqrt{r_{1\,r} \sin^2 \rho_r + r_{2\,r} (\cos^2 \rho_r)} & \text{(right end).} \end{cases}$$

Taking this fact into account for the computation of the tubelet's total length as drafted in Figure 3.38 we get

$$l_{total} = l + \Delta l_l + \Delta l_r \tag{3.10}$$

with $\Delta l_l \, , \Delta l_r \geq 0$.

To enhance the impression of the tubelet's surface shape, the correct surface normals are needed for correct lighting. Using the derivatives of (3.8) we get the surface normal N' by evaluating

$$N' = \frac{\partial x}{\partial \varphi} \times \frac{\partial x}{\partial x} \, . \tag{3.11}$$

We will go into more detail about lighting in the following section.

3.7.4 Sphere Tracing on the GPU

In order to generate tubelets on the GPU, we reduce the needed surface to its parameters as described in Section 3.7.3. The vertex programme calculates a bounding cuboid to scale the rendering primitive (a point) accordingly. Most of the values that are constant for a whole tubelet are computed from the parameters as well. This additional data is then passed to a fragment programme along with the parameters so it can find the proper surface intersection with one ray cast per pixel, add Phong shading and correct the depth value to fit the geometry.

For the GPU-based part of the algorithm we need the position, two colours, orientation (as a quaternion), the four radii, the right end rotation angle, the total length and the normals of the two bounding planes. We can fit this data into five float quadruples (O_{local} and l, q, $r_{1l}, r_{2l}, r_{1r}, r_{2r}$, N_l, N_r and ρ_r, as ρ_l is always 0 due to the definition of the local coordinate system) plus two byte quadruples (colour), so we only need to upload 644 bytes per tubelet, which is less than would be needed for 10 triangles with normals and constant colour. For each tubelet we upload a single point (with the attributes as texture coordinates) to the graphics card, since a point is the smallest type of

billboard in terms of data size, and as bonus we do not even need to adjust the orientation to face the eye position p_e.

The vertex programme computes two orientation matrices. The first one (M_c) is obtained after combining the tubelet orientation quaternion with the camera orientation quaternion. It is used for orbiting the eye point around the tubelet to obtain the local coordinate system described in Section 3.7.3. The second matrix (M_o) is obtained from the tubelet orientation only and used to calculate a bounding cuboid from the worst-case dimensions of the tube, i.e. a cylinder with length l_{total} and radius $max(r_{1l}, r_{2l}, r_{1r}, r_{2r})$. This cuboid is projected onto the view plane to obtain the point's extents and centre. Since the light position is also constant for all pixels of one tubelet, we rotate the light vector of our single light source with M_c in order to always have a headlight-like illumination.

The parameters we have to pass to the fragment programme are the following: the eye position p_e relative to the tubelet centred at $(0, 0, 0)$, the rotation matrix from the combined quaternions M_c, the transformed light vector, the two bounding planes and the radii and length. Furthermore, we only need to calculate Δl_l and Δl_r once for each tubelet so they are computed in the vertex programme and passed to the fragment shader. Together with the re-oriented z vector o' we use up all available varying parameters shared between vertex and fragment programme.

The fragment programme first has to find the vector s which connects the eye to the current pixel starting from the x- and y-component of the fragment's *window position*. To account for the fact that the origin is at the tubelet and the eye point orbits around it, M_c has to be applied to the resulting normalised ray direction s as well. To speed up the iteration process, we use an approximation of the intersection point as our starting point: The tubelet's surface is enclosed between two conical frustums, interpolating between the major axes of both ellipses in the bigger frustum and interpolating between the two minor axes in the other one. We calculate the intersection of our ray with the two cones and use the middle point between both intersection points as the start point for raycasting.

We then walk along the ray with a step-size computed from the approximated distance as in (3.9) until either the distance is below a threshold ω or a maximum number of steps has been walked (see below). If the distance left after the last step is beyond this threshold or we are outside the two clipping planes E_l, E_r, the fragment is discarded.

This may lead to holes in the surface if the ray is approximately parallel to the tubelet's axis as the step-size is almost constant and often very small, but it might still be far from the intersection point with the surface. To enhance the results in that case, we use an adaptive step-size: If the angle between x_t and s is small, we scale the step-size according to the angle between the surface

normal and the ray direction by $(1 - \cos(N \cdot s)) + 1$ which avoids these holes and, thus, leads to much better results with less iteration steps.

In case the fragment is not discarded, the correct depth is calculated to ensure that tubelets intersect correctly with each other and the volume as well. Since the eye point is displaced from $(0, 0, 0)^T$ and the view direction is no longer $o = (0, 0, -1)^T$, the depth z' is the distance to a plane normal to the orientation transformed by $o' = M_c^T o$ (see Figure 3.41), so we can use the Hessian normal form to get the distance and then fit the result to the exponential OpenGL depth range:

$$\begin{aligned} z' &= o' \cdot (p - p_e) \\ z_{ogl} &= -\frac{z_F + z_N}{2(z_N - z_F)} + \frac{1}{2} + \frac{z_F z_N}{(z_N - z_F)z} \end{aligned} \tag{3.12}$$

The normal is calculated as defined in (3.11) and used for Phong shading of the surface.

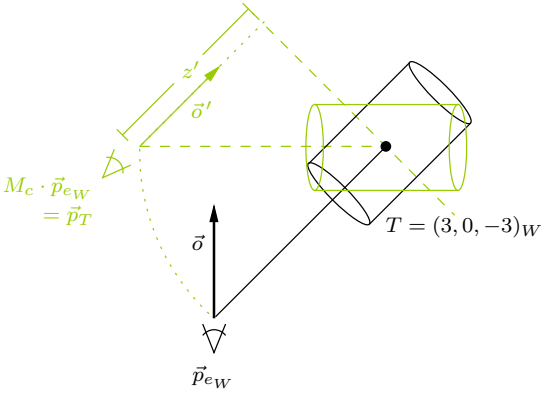


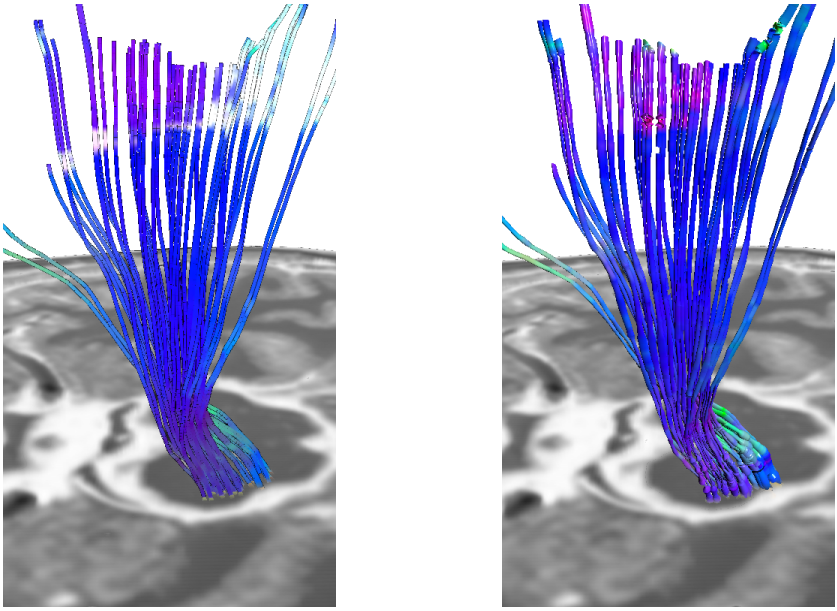
Figure 3.41: Local tubelet coordinates (green) in relation to world coordinates (black).

A drawback of this technique is that one cannot look inside the tubelets, since for rendering the back face we would have to start iterating on the other side of the local x-axis and, thus, require two times the performance we need now. However, there is no need to render the back face since users are usually not looking down the length of a tubelet because the relevant information (how the rotation and radii change over a certain distance) is visualised along the length of the tubelet and has to be interpreted in the context of the local x-coordinate.

3.7.5 Results on Hyperstreamlines

The advantage of DTI is its capability to provide the intrinsic diffusion properties of water within tissue. Due to the anatomical structure of neuronal pathways this diffusion is anisotropic in areas of major white matter tracts. Thus, DTI can reveal coherences in-vivo which are not visible in MRI_{T1} or MRI_{T2} datasets. An accepted method to access this information is to apply fibre tracking. However, since streamlines cannot convey tensor information their extension to hyperstreamlines is of certain value for detailed data analysis.

Our approach has been integrated into a framework for DTI visualization used at the Neurocentre, University of Erlangen-Nuremberg, for easy access to MRI data pre-processing and rendering of correct context information. Figure 3.42 (a) shows a line-rendering in comparison to hyperstreamlines (b) of a pyramidal tract combined with direct volume rendering of a MRI_{T1} dataset.



(a) Standard streamlines **(b)** Hyperstreamlines

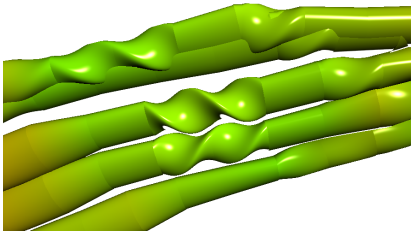
Figure 3.42: These two figures show the same pyramidal tract rendered with standard streamline (a) and with our method (b) in combination with direct volume rendering of a T1-weighted MRI dataset. For colouring the principal eigenvectors are mapped into RGB-colour space.

Using hyperstreamlines instead of simple lines or tubes enables the analysis of the whole tensor data. Areas with large eigenvalues will result in larger diameters of the hyperstreamline. This allows users to make conclusions about the underlying tissue. For a hyperstreamline showing a larger diameter it is very likely that it is not aligned with a neuronal pathway since white matter restricts the diffusion perpendicular to the cell orientation. Therefore, such expansions are an indication for an uncertainty in the fibre tracking.

The analysis regarding uncertainties can be further improved by the application of special colour schemes. Instead of utilising the standard RGB-scheme, where the principal eigenvector is used as colour vector, the colour can be selected by a scheme based on an approach presented by Westin et al. [WMK+99]. Thereby, areas with high anisotropy are depicted green while areas with planar diffusion are coloured yellow and isotropic areas appear red. Accordingly, red tube segments do have a higher uncertainty. Figure 3.36 previously showed a whole brain tracking. It can be seen that especially in the end segments the colour changes from green to red. This is plausible since the fibre tracking algorithm stops when reaching a voxel with a sufficiently low anisotropy which is depicted red.

Segments which appear yellow correspond to regions of planar diffusion. This occurrence is generally considered to be a potential fibre crossing which cannot be treated adequately with current fibre tracking algorithms. Therefore, such segments are of special interest and supportive rendering is desired. For the actual analysis of such regions hyperstreamlines are superior to common streamlines and -tubes. They provide information about the spatial orientation of the diffusion (Figure 3.43(a)). However, hyperstreamlines suffer from the same problem of restricted data accuracy as standard streamlines. Since DTI

(a) Hyperstreamlines traversing a region of planar diffusion



(b) Extreme torsion in hyperstreamlines

Figure 3.43: Figure (a) shows a bundle of hyperstreamlines traversing a region of planar diffusion leading to yellow colouring and a flattening of the tube. Some segments showing extreme torsion are depicted in (b).

data does not provide better resolution than the current 2 mm voxel spacing, all tracked lines can only be considered averaged models for the underlying tissue structure.

Another feature which can be represented by hyperstreamlines is torsion as depicted in Figure 3.43(b). The displayed extreme torsions are added manually as a proof of concept by switching off angle correction, thus allowing rotations larger than 90 degrees between consecutive ellipses. While strong torsion is not necessarily required for DTI visualization it is extremely useful for other data, namely technical simulation data.

In summary, this method demonstrates an alternative way to visualise hyperstreamlines relying on GPU-based iterative raycasting. This method allows us to calculate intersections with surfaces that cannot be implicitly ray cast. The main advantage of this approach is its suitability for large datasets and its inherent scalability with the evolution of graphics processors and its suitability for parallel rendering methods as SLI or graphics clusters.

In the context of medical visualization the enhanced geometry allows the user to benefit from both the connectivity information – usually available from streamlines – and the orientation information – usually available at discrete points by using glyphs – which is visualised as the semi axes of our ellipse tube section.

4 Discussion

The rampant amount of simulation data, resulting from the continuous increase of computational power and the pursuant fact that more and more complex and extensive simulation runs and set-ups are affordable, makes powerful but specialised analysis tools and visualizations increasingly crucial. Since in many application fields, geometry plays a crucial role in analysis, this thesis has focused on some geometric aspects of visualization methods.

Commonly, geometry, especially surfaces, are represented by meshes in computer graphics and, thus, in visualization. In recent years, alternative rendering methods based on mesh-free geometry representations emerged. Both approaches have their pros and cons, and therefore both have their qualification – depending on the application field or task as shown in this thesis. The methods introduced in the two described research fields, automotive prototyping and molecular dynamics of proteins, are tailored to the specific needs of the application issues and are developed in close collaboration with scientists in the dedicated research fields. However, the methods in both areas were designed to speed up the analysis process and to provide tools for new and further analysis prospects with geometry playing a key role.

In the application field of automotive pre-processing, the progress was mainly towards speeding up the whole car development process. This is crucial due to the continuously decreasing development times and narrowing development cycles in car industry. The provision of tools such as the optimization of FE meshes, presented in Section 2.3, and the filling of arbitrary holes in FE models, detailed in Section 2.4, – time-consuming tasks that were by then performed manually – not only account for the overall speed-up of automotive prototyping but also contribute to more dependable models and, thus, reliable simulations. On the other hand, developments such as the generation of mesh variants, described in Section 2.2, not directly account for the reliability of the models but allow for an intense speed-up of the simulation cycle as a whole which makes more simulation runs and, thus, the evaluation of more variants affordable. And finally, this higher number and variety of simulation set-ups leads to more sound models and results. However, the application of this method is not limited to the field of automotive prototyping but can be applied to various fields where FE meshes are used, and so the described method of generating mesh variants via volumetrical representation is meanwhile also used in productive simulations of steel casting.

In this scenario of the visualization and processing of finite element meshes, the role of geometry is directly bound to the models and thus the meshes, since the geometric aspects of the models are the main part of the simulation purpose. Therefore, geometric properties not only have to be kept in mind during the whole process but, as detailed in the approaches presented in this thesis, have to emphasised and, when being modified as part of the process cycle, special care and treatment has to be taken in order to guarantee reliable models – essential for trustworthy simulations.

By contrast, in the application field of molecular dynamics of protein-solvent systems the main problem is not to speed up an existing process. Protein structures are usually determined by X-ray crystallography or nuclear magnetic resonance spectroscopy (NMR). The increasing computational power, though, together with new methods meanwhile allow for simulations investigating this structural information based on physical aspects. However, a huge variety of simulation set-ups is needed. On the other hand, the functionality of proteins influenced or caused by separate parts within the protein (amino acids or even single atoms) as well as the interplay of proteins is investigated by computer simulations. Due to the slow dynamics of this application scenarios, long-term simulation runs are needed.

Since, as described in Section 3.2, most of the established post-processing and analysis tools for molecular dynamics data either lack the performance necessary to evaluate such huge amounts of data or lack support for time-based data at all, the primary issue in this context was to provide methods and tools to make the analysis of these extensive simulation runs affordable. Therefore, the methods presented in this thesis are all developed with time-based data – long trajectories of large data sets – in mind, and they all cope well with this task: The protein representations allow for efficient interactive rendering with superior visual quality, whereas the solvent visualization, especially the visual abstractions of solvent paths (Section 3.6.2), were developed in close collaboration with scientists in the application field and, thus, provide important analysis features that allowed for new insight but are neglected by the common available tools.

In contrast to the scenario of FE meshes, molecules do not have an intrinsic geometric representation and geometry is not the simulation purpose in molecular dynamics. Therefore, abstraction, and so geometry, can be intentionally employed to provide specialised visualizations, such as the cartoon representation of proteins detailed in Section 3.3.2, as well as novel analysis methods, i.e. the abstraction of solvent paths presented in Section 3.6.2, where geometry processing turned out to be the key for analysis.

Besides the differing aspects of both application areas described – in methodology as well as in visualization aims – some of the techniques can be applied to both kind of data. So, the mesh generation via volumetric representation can

be applied to proteins in order to create a meshed protein surface visualization of defined mesh size which can easily be textured. The volumetrical data can simply be generated based on the atom position coordinates together with the according van der Waals radii and subsequently meshed using the method described in Section 2.2. Due to the quad elements, textures can be applied very efficiently. The texturing of protein surfaces can be helpful to annotate protein areas during the evaluation process, e.g. with docking information or functionality details.

Additionally, the voxelization approach can be used with solvent-protein datasets in order to estimate volumes or surface areas of proteins as well as of their cavities. This is an important task in flexibility analysis, since these properties are considered one of the measure for protein flexibility, as well as in the investigation of the influence of protein cavities and tunnels on the functionality.

Bibliography

[ABE97] N. Amenta, M. Bern, and D. Eppstein. Optimal Point Placement for Mesh Smoothing. In *Proceedings of the Eighth Annual* ACM-SIAM *Symposium on Discrete Algorithms* (SODA'97), pages 528–537, 1997.

[AG02] J. F. Atkins and R. Gesteland. The 22nd Amino Acid. *Science*, 296(5572):1409–1410, 2002.

[AJL+04] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molekularbiologie der Zelle (Molecular biology of the cell)*. Wiley, 4th edition, 2004.

[AMHH08] T. Akenine-Möller, E. Haines, and N. Hoffman. *Real-Time Rendering*. A. K. Peters, Ltd., 3rd edition, 2008.

[Ami] Amira – Advanced 3D Visualization and Volume Modeling. http://www.amiravis.com/.

[Arg54] J. Argyris. Energy Theorems and Structural Analysis. Part I: General Theory. *Aircraft Engineering and Aerospace Technology*, 26:347–356, 383–387, 394, 1954.

[Arg55] J. Argyris. Energy Theorems and Structural Analysis. Part I: General Theory (contd.). *Aircraft Engineering and Aerospace Technology*, 27:42–58, 80–94, 125–134, 145–158, 1955.

[Arg71] Argyris, J.H. *Energy Theorems and Structural Analysis*. Butterworths, 1971.

[BDST04] C. Bajaj, P. Djeu, V. Siddavanahalli, and A. Thane. TexMol: Interactive Visual Exploration of Large Flexible Multi-Component Molecular Complexes. In VIS'04: *Proceedings of* IEEE *Visualization 2004*, pages 243–250, 2004.

[BE05] K. Bidmon and T. Ertl. Generation of Mesh Variants via Volumetrical Representation and Subsequent Mesh Optimisation . In *Proceedings 14th International Meshing Roundtable*, pages 275–286, 2005.

[BFG05] C.-S. Böttcher, S. Frik, and B. Gosolits. 20 Years of Crash Simulation at Opel – Experiences for Future Challenge. 4. LS-DYNA-Forum, 2005.

[BGB+08] K. Bidmon, S. Grottel, F. Bös, J. Pleiss, and T. Ertl. Visual Abstractions of Solvent Pathlines near Protein Cavities. *Computer Graphics Forum*, 27(3):935–942, 2008. Proceedings of EuroVis 2008.

[BK97] G. Barequet and S. Kumar. Repairing Cad Models. In Vis'97: *Proceedings of* Ieee *Visualization 1997*, pages 363–370, 1997.

[BL73] W. Blaschke and K. Leichtweiss. *Elementare Differentialgeometrie*. Die Grundlehren der mathematischen Wissenschaften, Band 1. Springer-Verlag, 1973.

[Blu01] R. Blumhardt. Fem – Crash Simulation and Optimization. *International Journal of Vehicle Design (Special Issue)*, 26(4):331–347, 2001.

[BMP+01] D. L. Bihan, J.-F. Mangin, C. Poupon, C. A. Clark, S. Pappata, N. Molko, and H. Chabriat. Diffusion Tensor Imaging: Concepts and Applications. *Journal of Magnetic Resonance Imaging*, 13:534–546, 2001.

[BP09] F. Bös and J. Pleiss. Multiple Molecular Dynamics Simulations of Tem β-Lactamase: Dynamics and Water Binding of the Ω-Loop. *Biophysical Journal*, 97(9):2550–2558, 2009.

[BPB06] J. Branch, F. Prieto, and P. Boulanger. Automatic Hole-Filling of Triangular Meshes using Local Radial Basis Function. In *3*Dpvt'06: *Proceedings of the Third International Symposium on 3D Data Processing, Visualization, and Transmission*, pages 727–734, 2006.

[BPK+08] M. Botsch, M. Pauly, L. Kobbelt, P. Alliez, B. Lévy, S. Bischoff, and C. Rössl. Geometric Modeling Based on Polygonal Meshes. In *Eurographics 2008 – Tutorials*, pages 1–181, 2008.

[BPP+00] P. J. Basser, S. Pajevic, C. Pierpaoli, J. Duda, and A. Aldroubi. In Vivo Fiber Tractography Using DT-Mri Data. *Magnetic Resonance in Medicine*, 44:625–632, 2000.

[Bra01] P. Bradford. Extended-Spectrum β-Lactamases in the 21st Century: Characterization, Epidemiology, and Detection of this Important Resistance Threat. *Clinical Microbiology Reviews*, 14(4):933–951, 2001.

[BRB+07] K. Bidmon, G. Reina, F. Bös, J. Pleiss, and T. Ertl. Time-Based Haptic Analysis of Protein Dynamics. In *World Haptics Conference (*Whc *2007): Second Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 537–542, 2007.

[BRE04] K. Bidmon, D. Rose, and T. Ertl. Intuitive, Interactive, and

Robust Modification and Optimization of Finite Element Models. In *Proceedings 13th International Meshing Roundtable*, pages 59–69, 2004.

[BS91] T. D. Blacker and M. B. Stephenson. Paving: A New Approach to Automated Quadrilateral Mesh Generation. *International Journal for Numerical Methods in Engineering*, 32:811–847, 1991.

[BT98] C. Branden and J. Tooze. *Introduction to Protein Structure*. Garland Publishing, 2nd edition, 1998.

[BvdSVD95] H. J. C. Berendsen, D. van der Spoel, and R. Van Drunen. GROMACS: A Message-Passing Parallel Molecular Dynamics Implementation. *Computer Physics Communications*, 91(1-3):43–56, 1995.

[BvG02] D. Bakowies and W. F. van Gunsteren. Water in Protein Cavities: A Procedure to Identify Internal Water and Exchange Pathways and Application to Fatty Acid-Binding Protein. *Proteins*, 47(4):534–545, 2002.

[BWT06] C. A. Bottoms, T. A. White, and J. J. Tanner. Exploring Structurally Conserved Solvent Sites in Protein Families. *Proteins*, 64(2):404–421, 2006.

[BZK09] D. Bommes, H. Zimmer, and L. Kobbelt. Mixed-Integer Quadrangulation. ACM *Transactions on Graphics*, 28(3):1–10, 2009. Proceedings of ACM SIGGRAPH 2009.

[Car91] M. Carson. Ribbons 2.0. *Journal of Applied Crystallography*, 24:958–961, 1991.

[CB86] M. Carson and C. E. Bugg. Algorithm for Ribbon Models of Proteins. *Journal of Molecular Graphics*, 4:121–122, 1986.

[CBC+01] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and Representation of 3D Objects with Radial Basis Functions. In SIGGRAPH'01: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 67–76, 2001.

[CC08] C.-Y. Chen and K.-Y. Cheng. A Sharpness-Dependent Filter for Recovering Sharp Features in Repaired 3D Mesh Models. IEEE *Transactions on Visualization and Computer Graphics*, 14(1):200–212, 2008.

[CCD+05] D. Case, T. Cheatham, T. Darden, H. Gohlke, R. Luo, K.M. Merz, Jr., A. Onufriev, C. Simmerling, B. Wang, and R. Woods. The Amber Biomolecular Simulation Programs. *Journal of Com-*

putational Chemistry, 26:1668–1688, 2005.

[Che95] E. V. Chernyaev. Marching Cubes 33: Construction of Topologically Correct Isosurfaces. Technical report, CERN CN 95–17, 1995.

[CLM08] M. Chavent, B. Lévy, and B. Maigret. MetaMol: High Quality Visualization of Molecular Skin Surface. *Journal of Molecular Graphics and Modelling*, 27(2):209–216, 2008.

[Clo60] R. W. Clough. The Finite Element Method in Plane Stress Analysis. *Proceedings of the* ASCE *2nd Conference on Electronic Computation*, 1960.

[COK95] D. Cohen-Or and A. Kaufman. Fundamentals of Surface Voxelization. *Graphical Models and Image Processing*, 57(6):453–461, 1995.

[Col55] L. Collatz. *Numerische Behandlung von Differentialgleichungen*. Springer, 1955. English edition *The numerical treatment of differential equations*, 1960.

[Con83] M. L. Connolly. Analytical Molecular Surface Calculation. *Journal of Applied Crystallography*, 16(5):548–558, 1983.

[Cou43] R. Courant. Variational Methods for the Solution of Problems of Equilibrium and Vibrations. *Bulletin of the American Mathematical Society*, 49(1):1–23, 1943.

[CPC+02] D. Case, D. Pearlman, J. Caldwell, T. Cheatham III, J. Wang, W. Ross, C. Simmerling, T. Darden, K. Merz, R. Stanton, A. Cheng, J. Vincent, M. Crowley, V. Tsui, H. Gohlke, R. Radmer, Y. Duan, J. Pitera, I. Massova, G. Seibel, U. Singh, P. Weiner, and P. Kollman. AMBER7. University of California, San Francisco, 2002. www.ambermd.org.

[CS92] X. Chen and F. Schmitt. Intrinsic Surface Properties from Surface Triangulation. In ECCV'92: *Proceedings of the Second European Conference on Computer Vision*, pages 739–743, 1992.

[CSL96] T. Callahan, E. Swanson, and T. Lybrand. MD Display: An Interactive Graphics Program for Visualization of Molecular Dynamics Trajectories. *Journal of Molecular Graphics*, 14:39–41, 1996.

[CTS98] S. A. Canann, J. R. Tristano, and M. L. Staten. An Approach to Combined Laplacian and Optimization-Based Smoothing for Triangular, Quadrilateral, and Quad-Dominant Meshes. In *Proceedings of 7th International Meshing Roundtable*, pages 479–494, 1998.

[CW99] R. W. Clough and E. Wilson. Early Finite Element Research at Berkeley. In USNCCM'99: *The Fifth US National Congress on Computational Mechanics*, 1999. Keynote Lecture.

[Dar87] G. Darboux. *Leçons sur la théorie générale des surfaces et les applications géométriques du calcul infinitesimal*, volume I–IV. Gauthier-Villars, 1887.

[dB01] C. de Boor. *A Practical Guide to Splines*. Applied Mathematical Sciences, Vol. 27. Springer, 2001.

[DBG+06] S. Dong, P.-T. Bremer, M. Garland, V. Pascucci, and J. C. Hart. Spectral Surface Quadrangulation. ACM *Transactions on Graphics*, 25(3):1057–1066, 2006. Proceedings of ACM SIGGRAPH 2006.

[dC76] M. P. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, 1976.

[DeL02] W. DeLano. PyMOL: An Open-Source Molecular Graphics Tool. CCP4 *Newsletter On Protein Crystallography*, 40, 2002. http://www.pymol.org.

[Des05] F. Despa. Biological Water: Its Vital Role in Macromolecular Structure and Function. *Annals of the New York Academy of Sciences*, 1066:1–11, 2005.

[DH93] T. Delmarcelle and L. Hesselink. Visualizing Second-Order Tensor Fields with Hyperstreamlines. IEEE *Computer Graphics and Applications*, 13(4):25–33, 1993.

[DJMH05] R. Davies, N. John, J. MacDonald, and K. Hughes. Visualization of Molecular Quantum Dynamics: A Molecular Visualization Tool with Integrated Web3D and Haptics. In *Proceedings of Web3D'05*, pages 143–150, 2005.

[DMGL02] J. Davis, S. R. Marschner, M. Garr, and M. Levoy. Filling Holes in Complex Surfaces Using Volumetric Diffusion. In *Proceedings of* 3DPVT2002: *The First International Symposium on 3D Data Processing, Visualization and Transmission*, pages 428–438, 2002.

[Don05] W. Donnelly. GPU *Gems 2*, chapter Per-Pixel Displacement Mapping with Distance Functions. Addison-Wesley, 2005.

[Dou31] J. Douglas. Solution of the Problem of Plateau. *Transactions of the American Mathematical Society*, 33(1):263–321, 1931.

[DSS+09] C. Dietrich, C. Scheidegger, J. Schreiner, J. Comba, L. Nedel, and C. Silva. Edge Transformations for Improving Mesh Quality of Marching Cubes. IEEE *Transactions on Visualization and Computer Graphics*, 15(1):150–159, 2009.

[dTL04] R. de Toledo and B. Lévy. Extending the Graphic Pipeline with New Gpu-Accelerated Primitives. Technical report, Inria-Alice, 2004.

[dTLP07] R. de Toledo, B. Lévy, and J.-C. Paul. Iterative Methods for Visualization of Implicit Surfaces on Gpu. In Isvc07: *International Symposium on Visual Computing*, pages 598–609, 2007.

[Ede99] H. Edelsbrunner. Deformable Smooth Surface Design. *Discrete & Computational Geometry*, 21(1):87–115, 1999.

[EM94] H. Edelsbrunner and E. P. Mücke. Three-Dimensional Alpha Shapes. Acm *Transactions on Graphics*, 13(1):43–72, 1994.

[ERMB00] L. J. Elias R. Melhema, Ryuta Itoha and P. B. Barkera. Diffusion Tensor MR Imaging of the Brain: Effect of Diffusion Weighting on Trace and Anisotropy Measurements. *American Journal of Neuroradiology*, 21:1813–1820, 2000.

[ESM+05] F. Enders, N. Sauber, D. Merhof, P. Hastreiter, C. Nimsky, and M. Stamminger. Visualization of White Matter Tracts with Wrapped Streamlines. In Vis'05: *Proceedings of* Ieee *Visualization 2005*, pages 51–58, 2005.

[Feh68] E. Fehlberg. Classical Fifth-, Sixth-, Seventh-, and Eighth-Order Runge-Kutta Formulas with Stepsize Control. Technical Report Nasa-TR-R-287, Nasa, 1968.

[FI04] M. Fujimoto and Y. Ishibashi. The Effect of Stereoscopic Viewing of a Virtual Space on a Networked Game Using Haptic Media. In Ace'04: *Proceedings of the 2004* Acm Sigchi *International Conference on Advances in Computer Entertainment Technology*, pages 317–320, 2004.

[Fre97] L. Freitag. On Combining Laplacian and Optimization-Based Mesh Smoothing Techniques. Amd *Trends in Unstructured Mesh Generation*, 220:37–43, 1997.

[FVF95] J. D. Foley, A. VanDam, and S. K. Feiner. *Computer Graphics: Principles and Practice*. Addison-Wesley, 2nd edition, 1995.

[Gau27] C. F. Gauss. *Disquisitiones Generales Circa Superficies Curvas*. Typis Dieterichianis, 1827. English edition: *General Investigations of Curved Surfaces*. Princeton University Library, 1902.

[GB78] J. Greer and B. L. Bush. Macromolecular Shape and Surface Maps by Solvent Exclusion. In *Proceedings of the National Academy of Science*, pages 303–307, 1978.

[GH00] A. E. García and G. Hummer. Water Penetration and Escape in Proteins. *Proteins*, 38(3):261–272, 2000.

[Gib98] S. F. F. Gibson. Using Distance Maps for Accurate Surface Representation in Sampled Volumes. In VVS'98: *Proceedings of the* IEEE *Symposium on Volume Visualization*, pages 23–30, 1998.

[GLM96] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A Hierarchical Structure for Rapid Interference Detection. In SIGGRAPH'96: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 171–180, 1996.

[Gum03] S. Gumhold. Splatting Illuminated Ellipsoids with Depth Correction. In VMV'03: *Proceedings of the Workshop on Vision, Modelling and Visualization*, pages 245–252, 2003.

[Ham93] B. Hamann. Curvature Approximation for Triangulated Surfaces. In *Geometric Modelling*, volume 8 of *Springer Computing Supplementum*, pages 139–153. Springer, 1993.

[Har96] J. C. Hart. Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces. *The Visual Computer*, 12(10):527–545, 1996.

[HDS96] W. Humphrey, A. Dalke, and K. Schulten. VMD – Visual Molecular Dynamics. *Journal of Molecular Graphics*, 14:33–38, 1996.

[HE95] D. Herbison-Evans. Solving Quartics and Cubics for Graphics. In A. W. Paeth, editor, *Graphics Gems V*, pages 3–15. Academic Press, 1st edition, 1995.

[Hel07] V. Helms. Protein Dynamics Tightly Connected to the Dynamics of Surrounding and Internal Water Molecules. *ChemPhysChem*, 8:23–33, 2007.

[Heu00] K. Heun. Neue Methode zur approximativen Integration der Differentialgleichungen mit einer unabhängigen Variablen. *Zeitschrift für Mathematik und Physik*, 45:23–38, 1900.

[HKvdSL08] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl. GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation. *Journal of Chemical Theory and Computation*, 4(3):435–447, 2008.

[HLC+01] J. Huang, Y. Li, R. Crawfis, S. C. Lu, and S. Y. Liou. A Complete Distance Field Representation. In VIS'01: *Proceedings of* IEEE *Visualization 2001*, pages 247–254, 2001.

[HM02] R. H. Henchman and J. A. McCammon. Extracting Hydration Sites Around Proteins from Explicit Water Simulations. *Journal of Computational Chemistry*, 23(9):861–869, 2002.

[HOF05] A. Halm, L. Offen, and D. Fellner. BioBrowser: A Framework for Fast Protein Visualization. In *Proceedings of EuroVis'05: Joint*

Eurographics - IEEE VGTC *Symposium on Visualization*, pages 287–294, 2005.

[Hre41] A. Hrennikoff. Solutions of Problems of Elasticity by the Framework Method. *Journal of Applied Mechanics*, 7:169–175, 1941.

[HYFK98] J. Huang, R. Yagel, V. Filippov, and Y. Kurzion. An Accurate Method for Voxelizing Polygon Meshes. In VVS'98: *Proceedings of the 1998* IEEE *Symposium on Volume Visualization*, pages 119–126, 1998.

[IHZ05] B. Itkowitz, J. Handley, and W. Zhu. The OpenHaptics Toolkit: A Library for Adding 3D Touch Navigation and Haptics to Graphics Applications. In *World Haptics Conference (*WHC *2005): First Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 590–591, 2005.

[Iup99] Iupac-IUBMB Joint Commission on Biochemical Nomenclature (JCBN) and Nomenclature Committee of IUBMB (NC-IUBMB). Newsletter 1999. *European Journal of Biochemistry*, 264(2):607–609, 1999.

[Kab76] W. Kabsch. A Solution for the Best Rotation to Relate Two Sets of Vectors. *Acta Crystallographica Section A*, 32(5):922–923, 1976.

[Kab78] W. Kabsch. A Discussion of the Solution for the Best Rotation to Relate Two Sets of Vectors. *Acta Crystallographica Section A*, 34(5):827–828, 1978.

[Kau87] A. Kaufman. Efficient Algorithms for 3D Scan-conversion of Parametric Curves, Surfaces, and Volumes. In SIGGRAPH'87: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, pages 171–179, 1987.

[KBB+00] L. Kobbelt, S. Bischoff, M. Botsch, K. Kähler, C. Rössl, R. Schneider, and J. Vorsatz. Geometric Modeling Based on Polygonal Meshes. In *Tutorials of the European Association for Computer Graphics 21st Annual Conference (Eurographics'00)*, pages 1–47, 2000.

[KBE08] M. Krone, K. Bidmon, and T. Ertl. GPU-Based Visualisation of Protein Secondary Structure. In *Proceedings of the Sixth Theory and Practice of Computer Graphics 2008 Conference (*TP.CG *2008)*, pages 115–122. EG, 2008.

[KBE09] M. Krone, K. Bidmon, and T. Ertl. Interactive Visualization of Molecular Surface Dynamics. IEEE *Transactions on Visualization and Computer Graphics*, 15(6):1391–1398, 2009.

[KBSS01] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel. Feature Sensitive Surface Extraction from Volume Data. In SIGGRAPH'01: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 57–66, 2001.

[KE04] T. Klein and T. Ertl. Illustrating Magnetic Field Lines Using a Discrete Particle Model. In VMV'04: *Proceedings of the Workshop on Vision, Modelling and Visualization*, pages 387–394, 2004.

[Kin04] G. Kindlmann. Superquadric Tensor Glyphs. In *Proceedings of* IEEE TVCG/*EG Symposium on Visualization 2004*, pages 147–154, 2004.

[KK99] A. Křenek and Z. Kabeláč. Studying Conformational Behaviour with PHANTOM Device(s). In *Proceedings of* PURS'99, pages 3–6, 1999.

[Kle07] B. Klein. FEM. Vieweg, 7th edition, 2007.

[Kli73] W. Klingenberg. *Eine Vorlesung über Differentialgeometrie*. Heidelberger Taschenbücher, Band 107. Springer-Verlag, 1973.

[KNP07] F. Kälberer, M. Nieser, and K. Polthier. QuadCover – Surface Parameterization using Branched Coverings. *Computer Graphics Forum*, 26(3):375–384, 2007.

[Kra06] F. Kramer. *Passive Sicherheit von Kraftfahrzeugen*, chapter 8. Rechnerische Simulation. Vieweg, 2006.

[Kre59] E. Kreyszig. *Differential Geometry*. University of Toronto Press, 1959.

[Kře00] A. Křenek. Haptic Rendering of Molecular Flexibility. In *Proceedings of* PURS2000, pages 19–26, 2000.

[Kře01] A. Křenek. Haptic Rendering of Molecular Conformations. In *Proceedings of EuroHaptics Conference 2001*, pages 142–145, 2001.

[Kře03] A. Křenek. Haptic Rendering of Complex Force Fields. In EGVE'03: *Proceedings of the Workshop on Virtual Environments 2003*, pages 231–239, 2003.

[KS83] W. Kabsch and C. Sander. Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22(12):2577–2637, 1983.

[Kut01] M. W. Kutta. Beitrag zur näherungsweisen Integration totaler Differentialgleichungen. *Zeitschrift für Mathematik und Physik*, 46:435–453, 1901.

[KvK99] A. Křenek, M. Černohorský, and Z. Kabeláč. Haptic Visualiz-

ation of Molecular Data. In WSCG'99 *Conference Proceedings*, 1999.

[KW99] G. Kindlmann and D. Weinstein. Hue-Balls and Lit-Tensors for Direct Volume Rendering of Diffusion Tensor Fields. In VIS'99: *Proceedings of* IEEE *Visualization 1999*, pages 183–189, 1999.

[KWH00] G. Kindlmann, D. Weinstein, and D. Hart. Strategies for Direct Volume Rendering of Diffusion Tensor Fields. IEEE *Transactions on Visualization and Computer Graphics*, 6(2):124–138, 2000.

[Lam73] J. Lambert. *Computational Methods in Ordinary Differential Equations*. John Wiley & Sons, 1973.

[LAS+02] N. Lori, E. Akbudak, J. Shimony, T. Cull, A. Snyder, R. Guillory, and T. Conturo. Diffusion Tensor Fiber Tracking of Human Brain Connectivity: Aquisition Methods, Reliability Analysis and Biological Results. NMR *in Biomedicine*, 15(7-8):494–515, 2002.

[Lau60] D. Laugwitz. *Differentialgeometrie*. Teubner, 1960.

[LB03] A. Lopes and K. Brodlie. Improving the Robustness and Accuracy of the Marching Cubes Algorithm for Isosurfacing. IEEE *Transactions on Visualization and Computer Graphics*, 9(1):16–29, 2003.

[LB06] C. Loop and J. Blinn. Real-Time GPU Rendering of Piecewise Algebraic Surfaces. ACM *Transactions on Graphics*, 25(3):664–670, 2006.

[LBPH10] N. Lindow, D. Baum, S. Prohaska, and H.-C. Hege. Accelerated Visualization of Dynamic Molecular Surfaces. *Computer Graphics Forum*, 29(3):943–952, 2010.

[LBY05] K. Lundin, G. Bjorn, and A. Ynnerman. General Proxy-Based Haptics for Volume Visualization. In *World Haptics Conference (*WHC *2005): First Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems.*, pages 557–560, 2005.

[LC87] W. E. Lorensen and H. E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In SIGGRAPH'87: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, pages 163–169, 1987.

[LG77] M. Levitt and J. Greer. Automatic Identification of Secondary Structure in Globular Proteins. *Journal of Molecular Biology*, 114:181–239, 1977.

[LH04] P. Lacz and J. C. Hart. Procedural Geometric Synthesis on the

GPU (Poster). In *Proceedings of the* ACM *Workshop on General Purpose Computing on Graphics Processors (GP^2)*, 2004.

[Lie03] P. Liepa. Filling Holes in Meshes. In SGP03: *Eurographics Symposium on Geometry Processing 2003*, pages 200–205, 2003.

[LKH08] Y.-K. Lai, L. Kobbelt, and S.-M. Hu. An Incremental Approach to Feature Aligned Quad Dominant Remeshing. In SPM'08: *Proceedings of the 2008* ACM *Symposium on Solid and Physical Modeling*, pages 137–145, 2008.

[LLVT03] T. Lewiner, H. Lopes, A. W. Vieira, and G. Tavares. Efficient Implementation of Marching Cubes´ Cases with Topological Guarantees. *Journal of Graphics Tools*, 8(2):1–15, 2003.

[LMZ86] R. Löhner, K. Morgan, and O. C. Zienkiewicz. Adaptive Grid Refinement for the Compressible Euler Equations. *Accurency Estimates and Adaptive Refinements in Finite Element Computations*, pages 281–297, 1986.

[Lod04] H. F. Lodish. *Molecular Cell Biology*. Freeman, 5th edition, 2004.

[Los61] J. Loschmidt. *Chemische Studien. Constitutions-Formeln der organischen Chemie in graphischer Darstellung*. 1861.

[LR71] B. Lee and F. M. Richards. The Interpretation of Protein Structures: Estimation of Static Accessibility. *Journal of Molecular Biology*, 55(3):379–400, 1971.

[LSK05] K. Lübke, E. Schröder, and G. Kloss. *Chemie und Biochemie der Aminosäuren, Peptide und Proteine II*. Thieme, 2005.

[LVRH07] O. D. Lampe, I. Viola, N. Reuter, and H. Hauser. Two-Level Approach to Efficient Visualization of Protein Dynamics. IEEE *Transactions on Visualization and Computer Graphics*, 13(6):1616–1623, 2007.

[LWYW07] Y. Lu, R. Wang, C. Y. Yang, and S. Wang. Analysis of Ligand-Bound Water Molecules in High-Resolution Crystal Structures of Protein-Ligand Complexes. *Journal of Chemical Information and Modeling*, 47(2):668–675, 2007.

[LYG02] K. Lundin, A. Ynnerman, and B. Gudmundsson. Proxy-based Haptic Feedback from Volumetric Density Data. In *Proceedings of the EuroHaptics Conference 2002*, pages 104–109, 2002.

[LYL06] S. K. Lai-Yuen and Y.-S. Lee. Energy-Field Optimization and Haptic-Based Molecular Docking and Assembly Search System for Computer-Aided Molecular Design (CAMD). In HAPTICS'06: *Proceedings of the 14th Symposium on Haptic Interfaces for Vir-*

tual Environment and Teleoperator Systems, page 34, 2006.

[Mat02] C. Mattos. Protein-Water Interactions in a Dynamic World. *Trends in Biochemical Sciences*, 27:203–208, 2002.

[MB04] G. Mustata and J. M. Briggs. Cluster Analysis of Water Molecules in Alanine Racemase and their Putative Structural Role. *Protein Engineering Design and Selection*, 17(3):223–234, 2004.

[MCET05] R. Maciejewski, S. Choi, D. Ebert, and H. Tan. Multi-Modal Perceptualization of Volumetric Data and Its Application to Molecular Docking. In *World Haptics Conference (*WHC *2005): First Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 511–514, 2005.

[MDSB03] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr. Discrete Differential-Geometry Operators for Triangulated 2-Manifolds. In H.-C. Hege and K. Polthier, editors, *Visualization and Mathematics III*, pages 35–57. Springer, 2003.

[Mee81] W. H. Meeks. A Survey of the Geometric Results in the Classical Theory of Minimal Surfaces. *Boletim da Sociedade Brasileira de Matemática*, 12(1):29–86, 1981.

[MP03] R. Měch and P. Prusinkiewicz. Generating Subdivision Curves with L-Systems on a GPU. In SIGGRAPH'03: ACM SIGGRAPH *2003 Sketches & Applications*, page 1, 2003.

[MRF+96] W. Mark, S. Randolph, M. Finch, J. Van Verth, and I. Taylor, R.M. Adding Force Feedback to Graphics Systems: Issues and Solutions. In SIGGRAPH'96: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 447–452, 1996.

[MSR07] E. Magid, O. Soldea, and E. Rivlin. A Comparison of Gaussian and Mean Curvature Estimation Methods on Triangular Meshes of Range Image Data. *Computer Vision and Image Understanding*, 107(3):139–159, 2007.

[MSS94] C. Montani, R. Scateni, and R. Scopigno. Discretized Marching Cubes. In VIS'94: *Proceedings of* IEEE *Visualization 1994*, pages 281–287, 1994.

[MT08] M. Meywerk and W. Tomaske. Crash ohne Krach. VDI *Mensch & Technik*, II/2008:16–17, 2008.

[MW00] D. Meek and D. Walton. On Surface Normal and Gaussian Curvature Approximations Given Data Sampled From a Smooth Surface. *Computer Aided Geometric Design*, 17(6):521–543, 2000.

[MWS02] G. Minasov, X. Wang, and B. Shoichet. An Ultrahigh Resolution Structure of TEM-1 Beta-Lactamase Suggests a Role for Glu166 as the General Base in Acylation. *Journal of the American Chemical Society*, 124(19):5333–5340, 2002.

[Nak04] M. Nakasako. Water-Protein Interactions from High-Resolution Protein Crystallography. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 359(1448):1191–1204, 2004.

[NH91] G. M. Nielson and B. Hamann. The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes. In VIS'91: *Proceedings of* IEEE *Visualization 1991*, pages 83–91, 1991.

[Nie03] G. M. Nielson. On Marching Cubes. IEEE *Transactions on Visualization and Computer Graphics*, 9(3):283–297, 2003.

[Nie04] G. M. Nielson. Dual Marching Cubes. In VIS'04: *Proceedings of* IEEE *Visualization 2004*, pages 489–496, 2004.

[Nit75] J. C. Nitsche. *Vorlesungen über Minimalflächen.* Die Grundlehren der mathematischen Wissenschaften, Band 199. Springer-Verlag, 1975.

[Ode87] T. Oden. Some Historic Comments on Finite Elements. In *Proceedings of the* ACM *Conference on History of Scientific and Numeric Computation*, pages 125–130, 1987.

[Oll72] W. Ollis. Models and Molecules. *Proceedings of the Royal Institution of Great Britain*, 45:1–31, 1972.

[OLW91] G. Otting, E. Liepinsh, and K. Wüthrich. Protein Hydration in Aqueous Solution. *Science*, 254(5034):974–980, 1991.

[O'N83] B. O'Neill. *Semi-Riemannian Geometry: With Applications to Relativity.* Academic Press, 1983.

[O'N97] B. O'Neill. *Elementary Differential Geometry.* Elsevier, 2nd edition, 1997.

[Oss67] R. Osserman. Minimal Surfaces. *Uspekhi Matematicheskikh Nauk*, 22(4):55–136, 1967.
Оссерман, Р., Минимальные поверхности, *Успехи математических наук.*

[Oss69] R. Osserman. *A Survey of Minimal Surfaces.* Number 25 in Van Nostrand Reinhold Mathematical Studies. Van Nostrand Reinhold Co, 1969.

[Oss70] R. Osserman. A Proof of the Regularity Everywhere of the Classical Solution to Plateau's Problem. *The Annals of Mathematics*, 91(2):550–569, 1970.

[PAB02] S. Pajevic, A. Aldroubi, and P. Basser. A Continuous Tensor

Field Approximation of Discrete DT-MRI Data for Extracting Microstructural and Architectural Features of Tissue. *Journal of Magnetic Resonance*, 154(1):85–100, 2002.

[Pam00] PAM System International. PAM-SCL *Theory Notes Manual*, 2000.

[PBW+05] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten. Scalable Molecular Dynamics with NAMD. *Journal of Computational Chemistry*, 26(16):1781–1802, 2005.

[PC51] L. Pauling and R. B. Corey. Configurations of Polypeptide Chains with Favored Orientations Around Single Bonds: Two New Pleated Sheets. *Proceedings of the National Academy of Sciences of the United States of America*, 37(11):729–740, 1951.

[PCB51] L. Pauling, R. B. Corey, and H. R. Branson. The Structure of Proteins: Two Hydrogen-Bonded Helical Configurations of the Polypeptide Chain. *Proceedings of the National Academy of Sciences of the United States of America*, 37(4):205–211, 1951.

[PCG+92] G. Perrot, B. Cheng, K. D. Gibson, J. Vila, K. A. Palmer, A. Nayeem, B. Maigret, and H. A. Scheraga. MSEED: A Program for the Rapid Analytical Determination of Accessible Surface Areas and Their Derivatives. *Journal of Computational Chemistry*, 13(1):1–11, 1992.

[Pet02] S. Petitjean. A Survey of Methods for Recovering Quadrics in Triangle Meshes. ACM *Computing Surveys*, 34(2):211–262, 2002.

[PGH+04] E. Pettersen, T. Goddard, C. Huang, G. Couch, D. Greenblatt, E. Meng, and T. Ferrin. UCSF Chimera - A Visualization System for Exploratory Research and Analysis. *Journal of Computational Chemistry*, 25(13):1605–1612, 2004.

[Pha] The PHANTOM Desktop device home page. http://www.sensable.com/haptic-phantom-desktop.htm.

[PK05] I. Peterlik and A. Křenek. Haptically Driven Travelling through Conformational Space. In *World Haptics Conference (*WHC *2005): First Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems.*, pages 342–347, 2005.

[PMA98] B. M. Pettitt, V. A. Makarov, and B. K. Andrews. Protein Hydration Density: Theory, Simulations and Crystallography. *Current Opinion in Structural Biology*, 8(2):218–221, 1998.

[PMV06] J.-P. Pernot, G. Moraru, and P. Véron. Filling Holes in Meshes Using a Mechanical Model to Simulate the Curvature Variation

Minimization. *Computers & Graphics*, 30(6):892–902, 2006.

[Poe05] G. Poeggel. *Kurzlehrbuch Biologie*. Thieme, 2005.

[PR05] J. Podolak and S. Rusinkiewicz. Atomic Volumes for Mesh Completion. In SGP05: *Eurographics Symposium on Geometry Processing 2005*, pages 33–41, 2005.

[PS05] S. Park and J. G. Saven. Statistical and Molecular Dynamics Studies of Buried Waters in Globular Proteins. *Proteins*, 60:450–463, 2005.

[PS06] N. Prabhu and K. Sharp. Protein-Solvent Interactions. *Chemical Reviews*, 106:1616–1623, 2006.

[Rad30] T. Radó. The Problem of the Least Area and the Problem of Plateau. *Mathematische Zeitschrift*, 32:763–796, 1930.

[RBE04] D. Rose, K. Bidmon, and T. Ertl. Intuitive and Interactive Modification of Large Finite Element Models. In VIS'04: *Proceedings of* IEEE *Visualization 2004*, pages 361–368, 2004.

[RBE+06] G. Reina, K. Bidmon, F. Enders, P. Hastreiter, and T. Ertl. GPU-Based Hyperstreamlines for Diffusion Tensor Imaging. In *Proceedings of EuroVis'06: Joint Eurographics* - IEEE VGTC *Symposium on Visualization*, pages 35–42, 2006.

[RC91] J. A. Rupley and G. Careri. Protein Hydration and Function. *Advances in Protein Chemistry*, 41:37–172, 1991.

[RE05] G. Reina and T. Ertl. Hardware-Accelerated Glyphs for Mono- and Dipoles in Molecular Dynamics Visualization. In *Proceedings of EuroVis'05: Joint Eurographics* - IEEE VGTC *Symposium on Visualization*, pages 177–182, 2005.

[Ric77] F. M. Richards. Areas, Volumes, Packing, and Protein Structure. *Annual Review of Biophysics and Bioengineering*, 6(1):151–176, 1977.

[Ric81] J. S. Richardson. The Anatomy and Taxonomy of Protein Structure. *Advances in Protein Chemistry*, 34:167–339, 1981.

[RKK97] D. C. Ruspini, K. Kolarov, and O. Khatib. The Haptic Display of Complex Graphical Environments. In SIGGRAPH'97: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pages 345–352, 1997.

[RPK07] J. Ryu, R. Park, and D.-S. Kim. Molecular Surfaces on Proteins Via Beta Shapes. *Computer-Aided Design*, 39(12):1042–1057, 2007.

[Run95] C. Runge. Über die numerische Auflösung von Differentialgleichungen. *Mathematische Annalen*, 46:167–178, 1895.

[SACO04] A. Sharf, M. Alexa, and D. Cohen-Or. Context-based Surface Completion. ACM *Transactions on Graphics*, 23(3):878–887, 2004. Proceedings of ACM SIGGRAPH 2004.

[San92] M. Sanner. *Sur la modélisation des surfaces moléculaires*. PhD thesis, Université de Haute-Alsace, Mulhouse, France, 1992.

[San04] B. Sanjeev. Ankush. Indian Institute of Science, 2004.

[SB78] J. Stoer and R. Bulirsch. *Einführung in die numerische Mathematik I*. Heidelberger Taschenbücher. Band 114. Springer-Verlag, 2nd edition, 1978.

[SBSE08] A. Schilling, K. Bidmon, O. Sommer, and T. Ertl. Filling Arbitrary Holes in Finite Element Models. In *Proceedings 17th International Meshing Roundtable*, pages 231–248, 2008.

[Sch51] K. Schellbach. Probleme der Variationsrechnung. *Journal für die reine und angewandte Mathematik*, 41:293–364, 1851.

[Sch91] H. R. Schwarz. *Methode der finiten Elemente*. B.G. Teubner Verlag, 3rd edition, 1991.

[SE00] O. Sommer and T. Ertl. Geometry and Rendering Optimization for the Interactive Visualization of Crash-Worthiness Simulations. In *Proceedings of the Visual Data Exploration and Analysis Conference in IT&T/*SPIE *Electronic Imaging*, pages 124–134, 2000.

[SE01] O. Sommer and T. Ertl. Comparative Visualization of Instabilities in Crash-Worthiness Simulations. In *Proceedings of EG/*IEEE TCVG *Symposium on Visualization VisSym'01*, pages 319–328,364, 2001.

[SEBH02] J. Schmidt-Ehrenberg, D. Baum, and H.-C. Hege. Visualizing Dynamic Molecular Conformations. In VIS'02: *Proceedings of* IEEE *Visualization 2002*, pages 235–242, 2002.

[See] (: SeeReal Technologies. http://www.seereal.com/.

[SEZ95] C. Simmerling, R. Elber, and J. Zhang. MOIL-View – A Program for Visualization of Structure and Dynamics of Biomolecules and STO – A Program for Computing Stochastic Paths. *Modelling of Biomolecular Structure and Mechanisms*, pages 241–265, 1995.

[SGS05] C. Stoll, S. Gumhold, and H. Seidel. Visualization with Stylized Line Primitives. In VIS'05: *Proceedings of* IEEE *Visualization 2005*, pages 695–702, 2005.

[SHL09] M. B. Stocks, S. Hayward, and S. D. Laycock. Interacting with the Biomolecular Solvent Accessible Surface Via a Haptic Feedback Device. BMC *Structural Biology*, 9(69), 2009.

[SK06] H. R. Schwarz and N. Köckler. *Numerische Mathematik*. B.G. Teubner Verlag, 6th edition, 2006.

[SMS+03] T. Surazhsky, E. Magid, O. Soldea, G. Elber, and E. Rivlin. A Comparison of Gaussian and Mean Curvatures Estimation Methods on Triangular Meshes. In IEEE *International Conference on Robotics and Automation (*ICRA*)*, pages 1021–1026, 2003.

[SN07] J. M. Singh and P. J. Narayanan. Real-Time Ray-Tracing of Implicit Surfaces on the GPU. Technical Report IIIT/TR/2007/72, International Institute of Information Technology, Hyderabad, India, 2007.

[SN09] J. M. Singh and P. Narayanan. Real-Time Ray Tracing of Implicit Surfaces on the GPU. IEEE *Transactions on Visualization and Computer Graphics*, 99(2):261–272, 2009.

[SO97] M. F. Sanner and A. J. Olson. Real Time Surface Reconstruction for Moving Molecular Fragments. In *Pacific Symposium on Biocomputing'97*, pages 385–396, 1997.

[SOS96] M. F. Sanner, A. J. Olson, and J.-C. Spehner. Reduced Surface: An Efficient Way to Compute Molecular Surfaces. *Biopolymers*, 38(3):305–320, 1996.

[Spi79] M. Spivak. *A Comprehensive Introduction to Differential Geometry*, volume I–V. Publish or Perish, 2nd edition, 1979.

[Sra01] M. Sramek. High Precision Non-Binary Voxelization of Geometric Objects. In SCCG'01: *Proceedings of the 17th Spring Conference on Computer Graphics*, page 220, 2001.

[SSFS06] J. Schreiner, C. E. Scheidegger, S. Fleishman, and C. T. Silva. Direct (Re)Meshing for Efficient Surface Processing. *Computer Graphics Forum*, 25(3):527–536, 2006. Proceedings of Eurographics 2006.

[Str69] K. Strubecker. *Differentialgeometrie III: Theorie der Flächenkrümmung*. Walter de Gruyter & Co, 1969.

[SWBG06] C. Sigg, T. Weyrich, M. Botsch, and M. Gross. GPU-Based Ray-Casting of Quadratic Surfaces. In *Eurographics Symposium on Point-Based Graphics*, pages 59–65, 2006.

[SWS+03] G. Sankaranarayanan, S. Weghorst, M. Sanner, A. Gillet, and A. Olson. Role of Haptics in Teaching Structural Molecular Biology. In HAPTICS'03*: Proceedings of the 11th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, page 363, 2003.

[TA95] M. Totrov and R. Abagyan. The Contour-Buildup Algorithm to

Calculate the Analytical Molecular Surface. *Journal of Structural Biology*, 116:138–143, 1995.

[TACSD06] Y. Tong, P. Alliez, D. Cohen-Steiner, and M. Desbrun. Designing Quadrangulations with Discrete Harmonic Forms. In SGP06: *Eurographics Symposium on Geometry Processing 2006*, pages 201–210, 2006.

[TC04] L. S. Tekumalla and E. Cohen. A Hole-Filling Algorithm for Triangular Meshes. Technical report, School of Computing, University of Utah, 2004.

[TCM06] M. Tarini, P. Cignoni, and C. Montani. Ambient Occlusion and Edge Cueing for Enhancing Real Time Molecular Visualization. IEEE *Transactions on Visualization and Computer Graphics*, 12(5):1237–1244, 2006.

[TMT56] R. W. Turner, M.and Clough, H. C. Martin, and L. J. Topp. Stiffness and Deflection Analysis of Complex Structures. *Journal of the Aeronautical Science*, 23(9):805–823, 1956.

[TvW99] A. Telea and J. J. van Wijk. Simplified Representation of Vector Fields. In VIS'99: *Proceedings of* IEEE *Visualization 1999*, pages 35–42, 1999.

[VBW94] A. Varshney, F. P. Brooks, and W. V. Wright. Linearly Scalable Computation of Smooth Molecular Surfaces. IEEE *Computer Graphics and Applications*, 14(5):19–25, 1994.

[vH77] J. H. van't Hoff. *Die Lagerung der Atome im Raume*. 1877.

[VKK+03] G. Varadhan, S. Krishnan, Y. J. Kim, S. Diggavi, and D. Manocha. Efficient Max-norm Distance Computation and Reliable Voxelization. In SGP03: *Eurographics Symposium on Geometry Processing 2003*, pages 116–126, 2003.

[Wat99] A. Watt. *3D Computer Graphics*. Addison-Wesley, 3rd edition, 1999.

[Wat09] D. S. Watt. *OpenGL Programming Guide*. Addison-Wesley, 7th edition, 2009.

[WL01] B. Wuensche and R. Lobb. The Visualization of Diffusion Tensor Fields in the Brain. In *Proceedings of the International Conference on Mathematics and Engineering Techniques in Medicine and Biological Science,* METMBS'01, pages 498–504, 2001.

[WMK+99] C.-F. Westin, S. E. Maier, B. Khidhir, P. Everett, F. A. Jolesz, and R. Kikinis. Image Processing for Diffusion Tensor Magnetic Resonance Imaging. In *Medical Image Computing and Computer-Assisted Intervention*, Lecture Notes in Computer Sci-

ence, pages 441–452, 1999.

[WMM+02] C.-F. Westin, S. E. Maier, H. Mamata, A. Nabavi, F. A. Jolesz, and R. Kikinis. Processing and Visualization of Diffusion Tensor Mri. *Medical Image Analysis*, 6(2):93–108, 2002.

[WO03] J. Wang and M. M. Oliveira. A Hole-Filling Strategy for Reconstruction of Smooth Surfaces in Range Images. In *Proceedings of* Sibgrapi, pages 11–18, 2003.

[WO07] J. Wang and M. Oliveira. Filling Holes on Locally Smooth Surfaces Reconstructed From Point Clouds. *Image and Vision Computing*, 25(1):103–113, 2007.

[WPS07] M. Weisel, E. Proschak, and G. Schneider. PocketPicker: Analysis of Ligand Binding-Sites with Shape Descriptors. *Chemistry Central Journal*, 1:7, 2007.

[Yoo07] D.-J. Yoo. Filling Holes in Large Polygon Models Using an Implicit Surface Scheme and the Domain Decomposition Method. *International Journal of Precision Engineering and Manufacturing*, 8(1):3–10, 2007.

[ZDL03] S. Zhang, C. Demiralp, and D. H. Laidlaw. Visualizing Diffusion Tensor MR Images Using Streamtubes and Streamsurfaces. Ieee *Transactions on Visualization and Computer Graphics*, 9(4):454–462, 2003.

[ZGAB08] N. Zonta1, I. J. Grimstead, N. J. Avis, and A. Brancale. Accessible Haptic Technology for Drug Design Applications. *Journal of Molecular Modeling*, 15(2):193–196, 2008.

[Zie67] O. C. Zienkiewicz. *The Finite Element Method in Structural and Continuum Mechanics*. McGraw-Hill, 1967.

[Zie70] O. C. Zienkiewicz. The Finite Element Method: From Intuition to Generality. *Applied Mechanics Reviews*, 23:249–256, 1970.

[ZXB07] W. Zhao, G. Xu, and C. Bajaj. An Algebraic Spline Model of Molecular Surfaces. In Spm'07: *Proceedings of the 2007* Acm *Symposium on Solid and Physical Modeling*, pages 297–302, 2007.

[ZZHW91] J. Z. Zhu, O. C. Zienkiewicz, E. Hinton, and J. Wu. A New Approach to the Development of Automatic Quadrilateral Mesh Generation. *International Journal for Numerical Methods in Engineering*, 32:849–866, 1991.