High Quality Surface Generation and Efficient Multiresolution Editing Based on Triangle Meshes

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der Rheinisch-Westfälischen Technischen Hochschule Aachen zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Mathematiker Mario Botsch
aus Bremen

Berichter: Prof. Dr. Leif Kobbelt
Prof. Dr. Peter Schröder

Tag der mündlichen Prüfung: 11. Juli 2005

Acknowledgments

First of all, I sincerely thank my advisor Leif Kobbelt for guiding and supporting me throughout all my studies in the last years, for motivating me by his endless enthusiasm, and for the innumerable discussions we had, which were both inspiring and extremely helpful.

I also thank Peter Schröder for co-advising this thesis, for his precious hints giving me a better understanding of the structure of Laplacian matrices, and for motivating me to further investigate on efficient methods for their solution.

I am grateful to Hans-Peter Seidel, since his lectures in Erlangen were the main motivation for me to start my research in the field of Computer Graphics. Many thanks go to my friends and former colleagues at the MPI in Saarbrücken, in particular Christian Rössl, Jens Vorsatz, and Uli Schwanecke.

The team of the Computer Graphics group at the RWTH Aachen supported me a lot by providing valuable feedback in many fruitful discussions. Special thanks go to Stephan Bischoff, Alexander Hornung, Martin Habbecke, and Darko Pavic for proofreading this thesis and improving it by helpful suggestions.

This thesis would not have been possible in this form without the support and collaboration of several people at the BMW group in Munich, in particular Wolf Bartelheimer, Oliver Theissen, and Peter Hoff.

Finally, I thank my parents for always motivating and supporting me and my beloved wife Ramona for her infinite amount of love, support, tolerance, and patience, which enabled me to fully concentrate (maybe too much) on my research work. Danke, Mini!

Contents

1 Introduction

Geometry processing is a steadily growing research field that became increasingly important during the last years. Digitizing physical prototypes using surface reconstruction techniques based on structured light or laser scanning has become affordable and produces accurately and densely sampled models, which often consist of up to several hundred millions of sample points. Due to their efficient processors and high memory capacities, current personal computers are able to efficiently process the acquired geometry data, and with the help of optimized graphics hardware even interactive processing of these datasets is possible nowadays.

A major problem is the large variety of possible data formats used in practice, that correspond to different surface representations for geometric models. Various representations are used by different (although collaborating) groups: while designers and CAD engineers typically work on higher order NURBS surfaces, numerical simulations or rapid prototyping techniques are based on piecewise linear polygonal meshes.

The necessary and frequent conversions between these surface representations are both time and resource consuming. Moreover, each conversion step corresponds to a resampling process and therefore inevitably introduces sampling artifacts and geometric aliasing. As a consequence, the required conversion steps should be reduced to a minimum, which corresponds to using *one* surface representation for as many as possible geometry processing tasks. Due to their high flexibility and efficiency, we propose to use triangle meshes as the main explicit surface representation.

Compared to NURBS surfaces, triangle meshes are far more flexible, since they can represent surfaces of arbitrary topology without the need to decompose them into several patches. Moreover, by applying discrete shape optimization based on physical models to the setting of irregular triangle meshes, a comparable high surface quality can be achieved. The conceptual simplicity of using triangles as surface primitives furthermore enables highly efficient implementations of all kinds of geometry processing algorithms.

As a consequence, triangle meshes started to assist or even replace classical NURBS surfaces in more and more engineering applications.

In this context, however, it is crucial to guarantee a sufficient approximation of the underlying surface geometry w.r.t. various application-dependent quality criteria and error tolerances. In the first part of this thesis we therefore focus on high quality surface approximations based on triangle meshes. The mesh generation and optimization algorithms introduced in Chap. 3 provide all the tools necessary for producing meshes of superior quality. In Chap. 4 these methods are further extended by feature-sensitive surface processing techniques, thereby enabling a high quality approximation also of technical datasets with complex geometric features.

The resulting triangle meshes faithfully approximate the geometry as well as the normal field of given surfaces up to a prescribed error tolerance. Their tessellation can additionally be optimized to consist of close-to-equilateral triangles only, thereby enabling numerically robust computations. As a consequence, these meshes are very well suited for use in numerical simulations and other engineering applications.

After discussing the generation and optimization of triangle meshes in the first part, the second part of the thesis deals with high quality shape deformations. In the concrete example of iterative optimization of aero dynamics in conceptual car design, for each optimization cycle the car's geometry is traditionally modified in a CAD system, and the resulting surface is converted to a triangle mesh for another CFD simulation. In order to avoid surface conversions as well as highly complicated CAD systems, these shape deformations should consequently be performed on the triangle mesh, using modeling metaphors which are sufficiently intuitive to be used by the CFD engineer himself.

Any practically useful shape deformation technique obviously has to preserve important geometric features of the surface, as those are the key component for high quality surface approximation. Physically plausible preservation of surface details under shape deformations is provided by multiresolution modeling techniques, which are introduced and discussed in Chap. 5.

The actual shape editing is computed by our boundary constraint modeling approach presented in Chap. 6. Since it is based on a constrained minimization of a physically motivated energy functional, it produces surface deformations of provably high smoothness, which are comparable to surfaces used in CAD systems. However, being completely

based on triangle meshes, our multiresolution shape editing framework is intuitive, highly flexible, and sufficiently fast to process even complex models in real-time.

Since the above variational energy minimization requires several solutions of large sparse linear systems, important issues concerning numerical robustness and computational efficiency are finally discussed in Chap. 7, where a detailed comparison of linear system solvers is given.

Contributions

The two main goals of this thesis are the generation of high quality surface approximations and their physically plausible and efficient deformation. Our main contributions to the two topics are:

- High quality mesh generation:
  - An efficient isotropic remeshing method with GPU-accelerated error control.
  - A feature-sensitive isosurface extraction technique
  - A resampling technique for blend regions in technical datasets
- Efficient multiresolution surface editing:
  - A multiresolution surface representation based on displacement volumes
  - An intuitive framework for high quality real-time shape deformations

Outline

The thesis is structured as follows:

Chapter 2 introduces different classes of shape representations, analyzes their strengths and drawbacks, and describes conversion algorithms between them.

Chapter 3 presents mesh optimization techniques for improving both the geometry and the tessellation of a given surface. It also proposes a global error framework in order to control the geometric deviation throughout all optimization processes.

Chapter 4 introduces feature-sensitive surface generation and remeshing techniques as the key component for a faithful approximation of the sharp surface features and highly curved blend regions typically contained in technical datasets.

Chapter 5 starts the second part of the thesis by introducing the concept of multiresolution modeling and presenting a novel multiresolution surface representation which provides physically plausible deformations without local self-intersections.

Chapter 6 concentrates on the freeform editing operator of a general multiresolution modeling framework. Our boundary constraint modeling approach allows for flexible and intuitively controlled deformations and is sufficiently fast for real-time editing even of complex models.

Chapter 7 finally discusses numerical robustness and performance issues related to our freeform shape editing. Different classes of linear system solvers are evaluated.

2 Surface Representations

The main topics of this thesis are the generation, optimization, and interactive deformation of surfaces. As a consequence, we are strongly interested in the efficient processing of all kinds of geometric objects, which requires — analogously to other fields of computer science — the design of suitable data structures. Since in our case the data to be processed are geometric shapes, each specific problem requires the right shape representation to be chosen in order to enable efficient access to the relevant information. In this context, there are two major classes of surface representations: *explicit* surfaces and *implicit* surfaces.

Explicit surfaces are defined by a vector-valued parameterization function $\mathbf{f} : \Omega \to \mathcal{S}$, that maps a two-dimensional parameter domain $\Omega \subset \mathbb{R}^2$ to the surface $\mathcal{S} \subset \mathbb{R}^3$. In contrast, an implicit (or volumetric) surface is implicitly defined to be the zero-set of a scalar-valued function $F : \mathbb{R}^3 \to \mathbb{R}$, i.e., $\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^3 \mid F(\mathbf{x}) = 0\}$. From an abstract point of view, parametric surfaces can be considered as the *range* of a function, while implicit ones are defined to be the *kernel* of a function.

Both representations have their own strengths and weaknesses, such that for each geometric problem the better suited one should be chosen. In order to analyze geometric operations and their requirements on the surface representation, we classify them into the following three categories, that have first been defined in this form in [Kob03]:

Evaluation: The sampling of the surface geometry or of other surface attributes, e.g., the surface normal field. A typical example is surface rendering.

Query: Spatial queries are used to determine whether or not a given point $\mathbf{p} \in \mathbb{R}^3$ is inside or outside of the solid bounded by a surface $\mathcal{S}$, which is a key component for solid modeling operations. Another typical query is the computation of a point's distance to the surface $\mathcal{S}$.

Modification: A surface can be modified either in terms of *geometry* (surface deformation), or in terms of *topology*, e.g., when different parts of the surface are to be merged.

We will see in Sect. 2.1 and Sect. 2.2 that explicit and implicit surface representations have complementary advantages w.r.t. these three kinds of geometric operations, i.e., the strengths of the one are the drawbacks of the other. Hence, for each specific geometric problem the more efficient representation should be chosen, which, in turn, requires efficient conversion routines between the two representations (Sect. 2.3). Consequently following this idea, highly efficient algorithms can be derived if the strengths of both representations are combined, e.g., by simultaneously using an explicit and an implicit representations of the same surface geometry [KB03b, BBK04].

2.1 Explicit Surface Representations

Explicit (or parametric) surface representations have the advantage that their parameterization $\mathbf{f} : \Omega \to \mathcal{S}$ enables the reduction of several three-dimensional problems on the surface $\mathcal{S}$ to two-dimensional problems in the parameter domain Ω. For instance, points on the surface can easily be generated by simple function evaluations of $\mathbf{f}$, which obviously allows for efficient evaluation operations. In a similar manner, geodesic neighborhoods, i.e., neighborhoods on the surface $\mathcal{S}$, can easily be found by considering neighboring points in the parameter domain Ω. A simple composition of $\mathbf{f}$ with a deformation function $\mathbf{d} : \mathbb{R}^3 \to \mathbb{R}^3$ results in an efficient modification of the surface geometry.

On the other hand, generating an explicit surface parameterization $\mathbf{f}$ can be very complex, since the parameter domain Ω has to match the topological and metric structure of the surface $\mathcal{S}$. When changing the shape $\mathcal{S}$, it might even be necessary to update the parameterization accordingly in order to reflect the respective changes of the underlying geometry: A low-distortion parameterization requires the metrics in $\mathcal{S}$ and Ω to be similar, and hence we have to avoid or adapt to excessive stretching.

However, since the surface $\mathcal{S}$ is the range of the parameterization $\mathbf{f}$, its topology can be controlled explicitly. In turn, changing the topology of an explicit surface $\mathcal{S}$ can be extremely complicated, since the parameterization as well as the domain Ω have to be adjusted accordingly. The typical inside/outside or distance queries are in general also

very expensive on explicit surfaces. Hence, topological modification and spatial queries can be noticed to be the weak points of explicit surfaces.

The main surface representation used in this thesis is the explicit *triangle mesh*, because it provides good approximation properties and allows for efficient processing. But before describing this *discrete* surface representation, we first outline the traditional *continuous* spline surfaces.

2.1.1 Spline Surfaces

Tensor-product spline surfaces are the standard surface representation of today's CAD systems. They are used for constructing high-quality (class A) surfaces from scratch as well as for later surface deformation tasks. Since tensor-product spline surfaces are defined in terms of spline curves, we will start with the description of the univariate curve case first. More details on both spline curves and spline surfaces can be found in [Far97, PT97, PBP02].

A spline curve $\mathbf{f}$ of degree n is a piecewise polynomial curve that is built by connecting several polynomial segments of maximal degree n in a smooth C^{n-1} manner. The parameter domain of a spline curve is the union of the segments' domain intervals $I_i := [u_i, u_{i+1}]$ and is defined by a knot vector $u_0 \leq u_1 \leq \ldots \leq u_L$:

$$\mathbf{f} : [u_0, u_L] \rightarrow \mathbb{R}^3, \quad \mathbf{f}|_{I_k} \in \Pi^n, \quad \mathbf{f} \in C^{n-1} \ .$$

The *B-spline* basis functions $N_i^n(u)$ are an efficient and intuitive basis for spline curves. The support of these functions is $\text{supp}(N_i^n) = [u_i, u_{i+n+1}]$, which is why $m + 1 = L - n$ basis functions $\{N_0^n \ldots, N_m^n\}$ are needed to cover the whole parameter domain. A three-dimensional B-spline curve is then represented by

$$\mathbf{f} : [u_n, u_{m+1}] \rightarrow \mathbb{R}^3, \quad u \mapsto \sum_{i=0}^{m} \mathbf{c}_i N_i^n(u) \ .$$

Here, the *control points* $\mathbf{c}_i \in \mathbb{R}^3$ define the so-called *control polygon* of the spline curve. Because $N_i^n(u) >= 0$ and $\sum_i N_i^n \equiv 1$, each curve point $\mathbf{f}(u)$ is a convex combination of the control points $\mathbf{c}_i$, i.e., the curve lies within the convex hull of the control polygon. Due to the small support of the basis functions, each control point has local influence on the curve only. These two properties cause spline curves to closely follow the control

polygon, thereby providing a geometrically intuitive metaphor for modeling curves by adjusting its control points. The intuitive control point metaphor and the built-in C^{n-1} smoothness are the reasons why spline curves are used in almost every CAD system.

The standard approach to generalize spline curves to spline surfaces are *tensor-product* surfaces, which are constructed by sliding curves on other curves, resulting in a surface that is swept in space. The swept curve is represented by a spline curve, whose control points slide on other spline curves:

$$\begin{aligned} \mathbf{f} : [u_n, u_{m+1}] \times [v_{n'}, v_{m'+1}] &\rightarrow \mathbb{R}^3 \\ (u, v) &\mapsto \sum_{i=0}^{m} \underbrace{\sum_{j=0}^{m'} \mathbf{c}_{ij} N_j^{n'}(v)}_{=:\mathbf{c}_i(v)} N_i^n(u) \quad . \end{aligned}$$

The two-dimensional parameter domain of a tensor-product surface is partitioned by two knot vectors u_i and v_j, and the geometric realization of the surface is defined by a regular *control lattice* $\left\{\mathbf{c}_{ij} \in \mathbb{R}^3 \,|\, 0 \leq i \leq m,\, 0 \leq j \leq m'\right\}$. Besides from that, tensor-product surfaces are basically "curves on curves", and therefore most of the curve properties generalize to surfaces.

A tensor-product surface — as the image of a rectangular domain under the parameterization $\mathbf{f}$ — always represents a rectangular surface patch embedded in $\mathbb{R}^3$. If shapes of more complicated topological structure are to be represented by spline surfaces, the model has to be composed of several (possibly trimmed) tensor-product patches.

Local level-of-detail control is another reason for decomposing a surface into several patches. In the case of spline curves, more degrees of freedom can locally be added by inserting additional knots u_i into the knot vector, resulting in more polynomial segments with associated control points $\mathbf{c}_i$. For tensor-product surfaces, however, inserting a knot into one of the two knot vectors causes a whole row or column of control points to be inserted into the regular control lattice. In order to replace this global surface refinement by a local one, the surface is subdivided into several patches, such that additional knots can be inserted into the necessary patches only. Recent work on T-splines [SZBN03, SCF+04], which allow for more general control lattices with T-junctions, could remedy these deficiencies to some degree, but topologically complicated surfaces still have to be composed from a large number of patches.

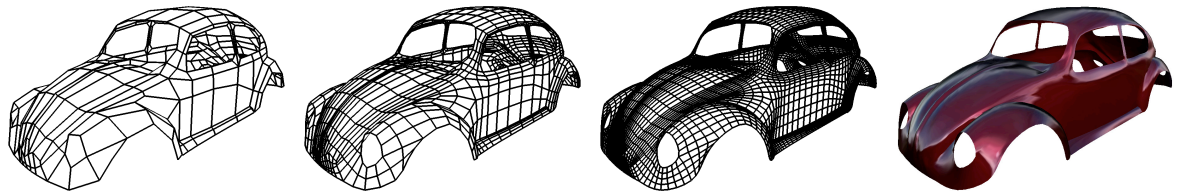


Figure 2.1: Subdivision surfaces are generated by an iterative refinement of a coarse control mesh.

As a consequence of these *topological constraints*, typical CAD models consist of a huge collection of surface patches. In order to represent a high quality globally smooth surface, these patches have to be connected in a smooth manner, leading to additional *geometric constraints*, that have to be taken care of throughout all surface processing phases. The large number of surface patches and the resulting topological and geometric constraints greatly complicate the surface construction and in particular the later surface modeling tasks.

2.1.2 Subdivision Surfaces

Subdivision surfaces [ZSD+00] can be considered as a direct generalization of spline surfaces, since they are also controlled by a coarse scale control mesh, but in contrast to spline surfaces allow to represent surfaces of arbitrary topology by a single control mesh. Subdivision surface are generated by a repeated refinement of control meshes: After each topological refinement step, the positions of the (old and new) vertices are adjusted based on a set of local averaging rules. A careful analysis of these rules reveals that in the limit this process results in a surface of provable smoothness (cf. Fig. 2.1).

As a consequence, subdivision surfaces are restricted neither by topological nor by geometric constraints as spline surfaces are, and their inherent hierarchical structure allows for highly efficient algorithms. However, subdivision techniques are restricted to surfaces with so-called *semi-regular subdivision connectivity*, i.e., surfaces that are (topologically) the result of repeated refinement of a coarse control mesh. As this constraint is not met by arbitrary surfaces, those would have to be *remeshed* to subdivision connectivity in a preprocessing step [EDD+95, LSS+98, KVLS99, GVSS00]. But as this remeshing corresponds to a resampling of the surface, it usually leads to sampling artifacts and loss of information. In order to avoid the restrictions caused by these *connectivity constraints*,

our goal is to work on arbitrary triangle meshes, as they provide higher flexibility and also allow for efficient surface processing.

2.1.3 Triangle Meshes

In contrast to spline surfaces, triangle meshes are neither specified in terms of a surface parameterization nor do they provide an inherent parameterization as subdivision surfaces do. However, triangle meshes are also defined in an explicit manner, and therefore are categorized to be an *explicit* surface representation, although not a *parametric* one.

A triangle mesh consists of a geometric and a topological component, where the latter can be represented by a set of vertices

$$\mathcal{V} = \{v_1, \ldots, v_V\}$$

and a set of triangular faces

$$\mathcal{F} = \{f_1, \ldots, f_F\}, \quad f_i \in \mathcal{V} \times \mathcal{V} \times \mathcal{V} \ ,$$

such that each triangle specifies its three vertices from $\mathcal{V}$. However, as we will see below, it is sometimes more efficient to represent the connectivity of a triangle mesh in terms of the edges of the respective graph:

$$\mathcal{E} = \{e_1, \ldots, e_E\}, \quad e_i \in \mathcal{V} \times \mathcal{V} \ ,$$

instead of in terms of faces. The geometric realization of a triangle mesh is specified by associating a 3D position $\mathbf{p}_i$ to each vertex $v_i \in \mathcal{V}$:

$$\mathcal{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_V\}, \quad \mathbf{p}_i := \mathbf{p}(v_i) = \begin{pmatrix} x(v_i) \\ y(v_i) \\ z(v_i) \end{pmatrix} \in \mathbb{R}^3 \ ,$$

such that each face $f \in \mathcal{F}$ actually represents a triangle in 3-space specified by its three vertex positions.

A connected triangle mesh therefore represents a continuous piecewise linear surface. If a sufficiently smooth surface is approximated by such a piecewise linear function, a local Tailor expansion reveals that the approximation error is of the order $O(h^2)$, with h denoting the maximum edge length. Due to this quadratic approximation power, the

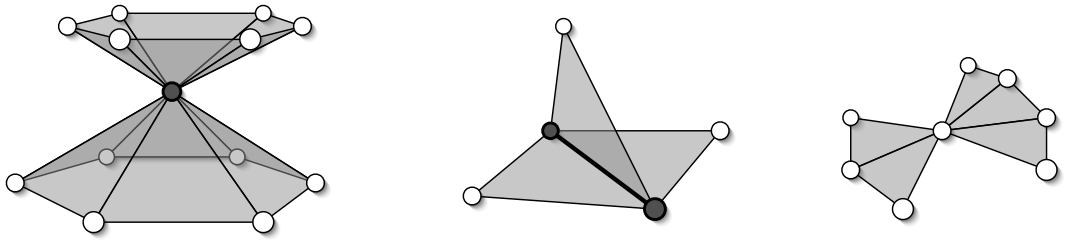


Figure 2.2: Two surface sheets meet at a non-manifold vertex (*left*). A non-manifold edge has more than two incident faces (*center*). The right configuration, although being non-manifold in the strict sense, can be handled without problems by halfedge data structures.

error is reduced by a factor of 1/4 by halving the edge lengths. As this also increases the number of triangles from F to $4F$, the approximation error of a triangle mesh is inversely proportional to the number of its faces. The approximation error depends on the higher order terms of the Taylor expansion, i.e., mainly on the second derivatives or the curvature of the underlying smooth surface. From this we can derive that a decent approximation is possible with just a moderate mesh complexity: The vertex density has to be locally adapted to the surface curvature, such that flat areas are sparsely sampled, while in detailed regions the sampling density is sufficiently higher.

In comparison to spline and subdivision surfaces, triangle meshes are not restricted by geometric, topological, or connectivity constraints, and hence can be considered to be the most flexible of these surface representations. Being 2-simplices, triangles are the conceptually simplest primitives for representing surfaces, and thus allow for the implementation of very efficient geometry processing algorithms. As a consequence, triangle meshes gained increasing attention in the field of engineering applications during the last years and have started to assist or even replace classical spline surfaces in this area.

An important topological characterization of a surface is whether or not it is *two-manifold*, which is the case if for each point the surface is locally homeomorphic to a disk (or a half-disk at boundaries). A triangle mesh is considered to be two-manifold, if it does neither contain non-manifold edges, non-manifold vertices, nor self-intersections, where a *non-manifold edge* has more than two incident triangles and a *non-manifold vertex* is generated by pinching two surface sheets together at that vertex, such that

the vertex is incident to two fans of triangles (cf. Fig. 2.2). Non-manifold meshes are in general problematic for surface processing algorithms, since around non-manifold configurations there is no clearly defined notion of local geodesic neighborhoods.

For the development of efficient algorithms for triangle meshes suitable data structures have to be found, which should provide fast access to the most frequently accessed kinds of neighborhood information. An analysis of the typical mesh processing algorithms and their neighborhood access patterns reveals that the time-critical neighborhood information is the so-called *one-ring neighborhood*, i.e., all vertices, edges, or faces being incident to a given vertex. Fast access to and iteration around a one-ring neighborhood requires to store and manage the connectivity information at the *edges* of the mesh instead of at its *faces*. Even higher efficiency can be achieved by using so-called *directed halfedges* to encode the connectivity [CKS98]. Our publicly available implementation of such a halfedge data structure, which closely follows the design principles of [Ket98], is `OpenMesh` [BSBK02, BSM05], which was used to implement all algorithms presented in this thesis.

Coming back to the previously mentioned three classes of geometric operations, triangle meshes share the typical strengths and drawbacks of explicit representations. Their elements can easily be enumerated and they are especially designed for efficient access to geodesic neighborhoods; by changing their vertex positions, geometric shape deformations can easily be performed. However, avoiding excessive stretching of triangles requires a local restructuring of the tessellation [KBS00], similar to an adjustment of a surface parameterization. Changing the topology, like cutting parts of the surface or merging them, is — due to the higher topological flexibility — less complicated than for spline surfaces, but can still get quite involved. Spatial queries are very expensive for triangle meshes unless spatial helper data structures are used, which, in turn, can also be regarded as an additional implicit surface representation.

In contrast to spline or subdivision surfaces, triangle meshes do not provide an explicit parameterization. Deriving a suitable global parameterization is a non-trivial problem, since this requires the mesh to be topologically equivalent to a disk. Otherwise, the mesh has to be opened by a series of carefully placed cuts. The construction of high-quality stretch-minimizing parameterizations is an active research area, that gained much attention in the last years [AMD02, DMA02, GGH02, LPRM02].

Analogously to other explicit surface representations, the problematic operations for triangle meshes are spatial queries and topological changes, which will turn out in the next section to be exactly the strengths of implicit surface representations.

2.2 Implicit Surface Representations

The basic concept behind *implicit* or *volumetric* representations of geometric models is to characterize the whole embedding space of an object by classifying each 3D point to lie either inside, outside, or exactly on the surface $\mathcal{S}$ bounding a solid object. By this they are independent from the actual surface topology, and because of this volumetric representations are preferred in applications where the topology of an object is complicated or even changes during an operation.

There are different conceptual frameworks for implicit surface representations, like for instance continuous algebraic surfaces and radial basis functions, or discrete voxelizations. In any case, the surface $\mathcal{S}$ is defined to be the zero-level iso-surface of a scalar-valued function $F : \mathbb{R}^3 \rightarrow \mathbb{R}$. By definition, negative function values of F designate points inside the object and positive values points outside the surface, i.e., the iso-surface $\mathcal{S}$ separates the inside from the outside of $\mathcal{S}$. As a consequence, geometric inside/outside queries simplify to function evaluations of F and checking the sign of the resulting value. The implicit function F for a given surface $\mathcal{S}$ is not uniquely determined, but the most common and most natural implicit representation is the so-called *signed distance function*, which maps each 3D point to its signed distance from the surface $\mathcal{S}$. In addition to inside/outside queries, this representation also simplifies distance computations to simple function evaluations.

On the other hand, enumerating points on the surface or finding geodesic neighborhoods is hardly possible with implicit representations. Moreover, implicit surfaces do not provide any means of parameterization, which is why it is almost impossible to consistently paste textures onto evolving implicit surfaces. However, they allow for the design of algorithms that are free of parameterization artifacts, since they are only based on intrinsic geometric properties of the surface [Set96, Set99, OF02].

One of the strengths of implicit surfaces is that they can easily change their topology, which is crucial, for instance, in the context of constructive solid geometry. However, this

can also be considered one of their main problems, since there is no general mechanism to prevent the topology from changing accidentally, i.e., to prevent the surface from merging or splitting. However, including additional explicit surface information allows to preserve the topology of an evolving surface, as was shown in [BK03b]. On the other hand, an implicit surface is a level-set of a potential function, therefore geometric self-intersections cannot occur.

The parameter domain of the implicit function is the whole 3-space, but in practice the function F is usually restricted to some bounding box around the surface. In order to efficiently process volumetric representations, a discrete approximation of the continuous scalar field F is generated by sampling F on a sufficiently dense grid

$$\left\{\mathbf{g}_{i,j,k} \in \mathbb{R}^3 \mid 1 \leq i \leq l,\, 1 \leq j \leq m,\, 1 \leq k \leq n\right\} \quad .$$

Hence, the most basic representation is a uniform scalar grid of sampled values $F_{i,j,k} := F(\mathbf{g}_{i,j,k})$, where function values in the interior of voxels are obtained by tri-linear interpolation, which results in quadratic approximation order. However, the memory consumption of this naïve data structure grows cubically if the precision is increased by reducing the edge length of grid voxels.

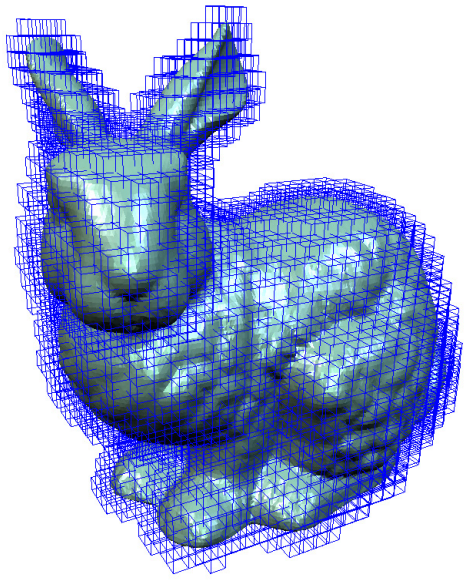


Figure 2.3: Adaptive octree refinement around the surface $\mathcal{S}$.

Therefore the sampling density is often adapted to the local geometric significance in the scalar field F: Since the signed distance values are most important in the vicinity of the surface, a higher sampling rate can be used in these regions only. Instead of a uniform 3D grid, a hierarchical octree is then used to hold the sampled values [Sam94]. The further refinement of an octree cell lying completely inside or outside the object does not improve the approximation of the surface $\mathcal{S}$. Adaptively refining only those cells that are intersected by the surface yields a uniformly refined crust of leaf cells around the surface and reduces the storage complexity from cubic to quadratic.

If the local refinement is additionally restricted to those cells where the tri-linear interpolant deviates more than a prescribed tolerance from the actual distance field, the resulting approximation adapts to the locality of the surface as well as to its shape complexity [FPRJ00]. Since extreme refinement is only necessary in regions of high surface curvature, this approach reduces the storage complexity even further and results in a memory consumption comparable to explicit representations. Similarly, an adaptive space-decomposition with linear (instead of tri-linear) interpolants at the leaves can be used [WK03]. Although the asymptotic complexity as well as the approximation power are the same, the latter method provides slightly better memory efficiency and allows for simpler algorithms.

If the implicit surface evolves over time, the direction of the surface movement can locally be restricted to the surface normal direction, leading to the so called *level set surfaces* [Set96, Set99, OF02]. Since the direction of motion is fixed, the surface evolution is fully determined by providing a scalar speed function $s(\mathbf{x}, t)$, defining the amount of movement for each point $\mathbf{x}$ at a certain time t. For instance, the distance field to a given surface can easily be computed by using the isotropic constant speed function $s(\mathbf{x}, t) \equiv 1$, a fact we will use in Sect. 3.3.

2.3 Conversion Methods

The last sections pointed out that a suitable shape representation has to be chosen in a problem-dependent manner, as this allows for the most efficient algorithms. In order to be able to freely choose between explicit and implicit representations of the same surface geometry, efficient conversion methods between the different representations are necessary.

Since both kinds of representations are usually finite samplings (triangle meshes in the explicit case, uniform/adaptive grids in the implicit case), each conversion corresponds to a re-sampling step. Hence, special care has to be taken in order to minimize the loss of information during these conversion routines.

2.3.1 Explicit to Implicit

The conversion of an explicit surface representation to an implicit one amounts to the computation of its signed distance field. This can be done very efficiently by voxelization or 3D scan-conversion techniques [Kau87], but the resulting approximation is piecewise constant only. More accurate approximations can be achieved by using higher order implicit surfaces [VG96, YT02].

As a surface's distance field is in general not smooth everywhere, a piecewise linear or piecewise tri-linear approximation seems to be the best compromise between approximation accuracy and computational efficiency. Since our standard explicit representation is a triangle mesh, the conversion to an implicit representation basically requires the computation of signed distances to the triangle mesh at the nodes of a (uniform or adaptive) 3D grid. This task can efficiently be performed using *Fast marching* methods [Set96], as also proposed in [MBWB02].

The marching process is initialized by computing the exact distance values for all grid nodes in the immediate vicinity of the triangle mesh. Since we want to approximate a *signed* distance field, we have to determine for each distance computation whether a grid node lies inside or outside the object. If $\mathbf{g}$ denotes the grid node and $\mathbf{c}$ its closest point on the surface, then the orientation can be derived from the angle between $(\mathbf{g}-\mathbf{c})$ and the normal $\mathbf{n}(\mathbf{c})$: $\mathbf{g}$ is defined to be inside if $(\mathbf{g}-\mathbf{c})^T\mathbf{n}(\mathbf{c}) < 0$. The robustness and reliability of this test strongly depends on the way the normal $\mathbf{n}(\mathbf{c})$ is computed. Using barycentric normal interpolation within triangles' interiors and computing per-vertex normals using angle-weighted averaging of face normals was shown to yield correct results [AB03].

After this initialization, a standard fast marching method with constant speed function $s(\mathbf{x}) \equiv 1$ is used to derive the distance values at the unknown grid nodes. Starting from the initialized grid nodes, all their immediate neighbors are inserted into a min-heap based on their distance from the advancing front. After conquering the nearest of these candidate nodes, all of its non-conquered neighbors are inserted into the heap, and this process is continued until all grid points have been conquered. Since we want to compute the distance field in the interior and exterior of the surface, a second marching step with inverted signs is used to compute the distances at the interior nodes.

2.3.2 Implicit to Explicit

An explicit surface can be constructed from an implicit representation by extracting the zero-level iso-surface of the scalar field F. For the conversion to a triangle mesh, two sub-problems have to be solved: In a first step a dense set of surface samples has to be found, that is to be connected in a topologically consistent manner in the second step, where we distinguish between grid-based and grid-less techniques. Higher order parametric representations, such as spline surfaces, are usually constructed in a second step by surface fitting techniques [HDD+94].

Grid-less techniques start with an initial triangle mesh approximating the surface $\mathcal{S}$, which is then iteratively improved by attracting its vertices to the isosurface $\mathcal{S}$ based on the scalar field F. Combining this *attracting force* with a *regularizing force* improves the aspect ratio of triangles and leads to high quality meshes if the underlying surface $\mathcal{S}$ is smooth [KWT98, LKE98, WDSB00].

In contrast, grid-based methods sample the implicit function on a regular grid and process each cell of the discrete distance field separately, thereby allowing for trivial parallelization. For each cell which is intersected by the iso-surface $\mathcal{S}$ a surface patch is generated based on local criteria. The collection of all these small pieces eventually yields a triangle mesh approximation of the complete iso-surface $\mathcal{S}$. Most grid-based techniques are conceptually derived from the *Marching Cubes* algorithm [LC87], which is the de-facto standard technique for iso-surface extraction.

For each edge intersecting the surface $\mathcal{S}$ the Marching Cubes algorithm computes a sample point which approximates this intersection. In terms of the scalar field F this means that the sign of F differs at the edge's endpoints $\mathbf{p}_1$ and $\mathbf{p}_2$. Since the tri-linear approximation F is actually linear along the grid edges, the intersection point $\mathbf{s}$ can be found by linear interpolation of the distance values $d_1 := F(\mathbf{p}_1)$ and $d_2 := F(\mathbf{p}_2)$ at the edge's endpoints:

$$\mathbf{s} \;=\; \frac{|d_2|}{|d_1| + |d_2|}\,\mathbf{p}_1 \;+\; \frac{|d_1|}{|d_1| + |d_2|}\,\mathbf{p}_2 \;\;.$$

The resulting sample points of each cell are then connected to a triangulated surface patch based on a triangulation look-up table holding all possible configurations of edge intersections (cf. Fig. 2.4). Since the possible combinatorial configurations are determined by the signs at a cell's corners, their number is $2^8 = 256$.

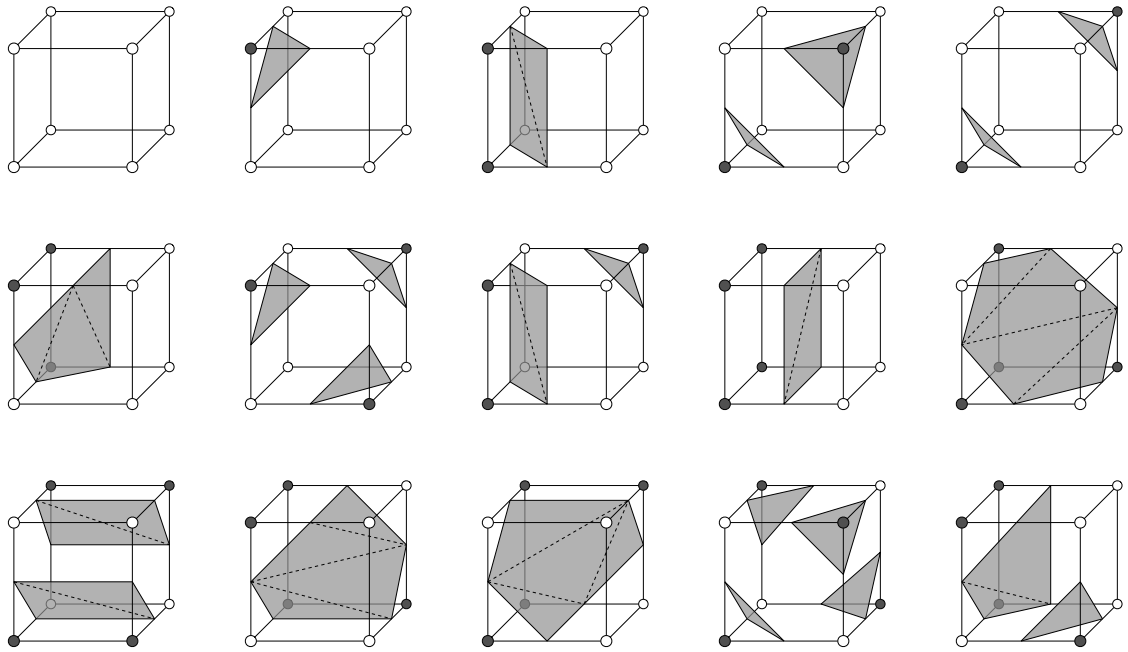


Figure 2.4: The 15 base configurations of the Marching Cubes triangulation table. The other cases can be found by rotation or symmetry.

The Marching Cubes algorithm is a conceptually very simple method and therefore allows for an efficient implementation. However, due to the regular 3D grid its complexity grows cubically with the grid resolution, i.e., with the approximation accuracy. Hence, in practice an adaptive octree should be used instead of the regular grid, such that only the octree cells which actually intersected the iso-surface are refined further, leading to a uniformly refined crust around the surface $\mathcal{S}$ (cf. Fig. 2.3). Since all leaves are on the same refinement level, the simple algorithmic structure is preserved.

Many variants of this basic algorithm have been published, which resolve ambiguities [MSS94b, NH91] or suggest alternative ways to approximate the surface samples [MSS94a]. One of the main advantages of this kind of iso-surface extraction is that the resulting mesh is guaranteed to be a closed 2-manifold surface. Therefore most methods for repairing inconsistent non-manifold meshes with possible gaps and holes first convert the input data to an implicit representation and extract a clean mesh in a second step [BPK04, Ju04, KBSS01, SOS04].

Since applying the Marching Cubes algorithm corresponds to a regular re-sampling of F, high-frequency geometric details, like sharp edges or corners, will not be captured. A refinement of the underlying 3D grid is not capable of removing the resulting alias-artifacts [KBSS01, BK01a]. We discuss these issues in Chap. 4, where we present a solution to the aliasing problem by a feature-sensitive surface extraction technique.

3 High-Quality Mesh Generation

In this chapter we focus on the generation of "good" meshes, that are of sufficient quality to be used in numerical simulations. The process starts with the generation of an initial mesh (Sect. 3.1), whose input data may come from a variety of sources, like 3D range-scanning, medical imaging, a solid modeling software package, or the tessellation of CAD surfaces initially represented by spline surfaces. However, the initial surfaces are in most cases not yet suited for further processing in engineering applications. They usually have to be optimized, e.g., by removing geometric noise, by reducing their complexity, and by optimizing their triangles' shapes to have bounded aspect ratio (Sect. 3.2). During the whole mesh generation and mesh optimization process, a user-specified upper bound on the approximation error to the original data has to be satisfied (Sect. 3.3), since otherwise the resulting meshes might deviate too much from the "real" surface geometry to yield meaningful results in numerical simulations.

A good, although slightly outdated overview of these and other mesh generation and mesh optimization techniques can be found in our tutorial notes of [KBB+00].

3.1 Mesh Generation

In the following discussion we distinguish the different methods for generating the initial mesh by the intermediate surface representation they are based on. First, there are methods which stitch together a collection of acquired surface patches in order to eventually integrate them into one surface. The intermediate representation of these patches is an explicit one (Sect. 3.1.1). On the other side are approaches that construct a signed distance function in a first step, and use an iso-surface extraction algorithm like Marching Cubes to compute the final surface. These methods are categorized to be implicit mesh generation techniques (Sect. 3.1.2).

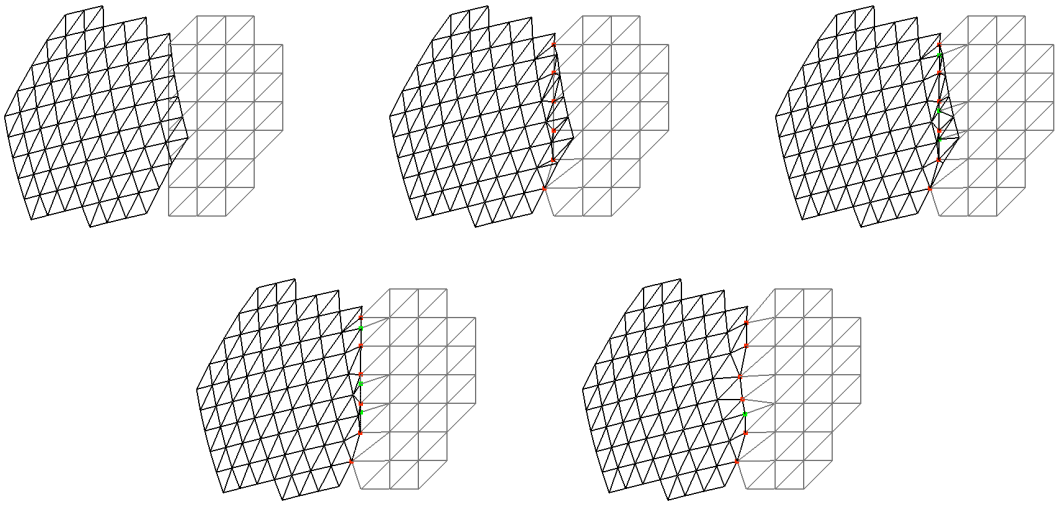


Figure 3.1: When the right mesh is to be stitched into the left one (1), first its boundary vertices are inserted into the left mesh (2). Splitting or flipping of edges leads to a common boundary polygon (3), such that redundant regions can easily be removed (4). Finally collapsing short edges in the resulting mesh removes badly shaped triangles (5).

3.1.1 Explicit Mesh Generation

The methods in this category work by integrating a set of triangulated surface patches into one consistent target model. Typical examples of how to generate the initial surface patches are the tessellation of the different patches of a spline model, or the patch-wise acquisition of a model's geometry by range scanning [CL94]. Conceptually similar to range scanning is our *virtual range scanning* method [KB00], which scans a set of unorganized input points by rendering them from several viewpoints and reading back the OpenGL z-buffer. After a filtering of the resulting depth values, the back-projection of all valid pixels $(x, y, z(x, y))^T$ of a rendered image yields a regular grid of 3D positions, similar to a structured range image.

These surface patches are then merged into one single model by a mesh zippering or mesh stitching technique as proposed in [TL94, KB00] and depicted in Fig. 3.1. After removing redundant triangles in overlapping regions, the boundaries of the two meshes are clipped against or intersected with each other in order to generate one common boundary polygon. This allows to connect the two meshes along this common boundary line in a consistent manner and thereby merges the two input meshes into one. However,

the required intersection of edges and faces inevitably leads to arbitrary small and skinny triangles, which makes a post-optimization of the edges and triangles in the seam regions mandatory. Iteratively stitching all source patches one by one finally results in a single mesh.

For this kind of explicit mesh generation, the original surface patches are parts of the final mesh, which can be advantageous if they are of high quality, e.g., when they are built by a feature-aligned tessellation of a CAD surface. The drawback of these methods is that the stitching of two meshes can produce poorly shaped triangles, requiring a post-optimization of the seam region. Hence, when there are not too many complicated seams, i.e., when the surface geometry allows to construct large patches with simple boundary curves, then the stitching approach is preferable.

3.1.2 Volumetric Mesh Generation

When reconstructing surfaces from scanned data, holes resulting from missing data due to inevitable occlusions or difficult surface reflectances are a major problem. On the other side, it is well known that iso-surface extraction methods yield clean two-manifold surfaces without holes. Therefore, Curless and Levoy [CL96] proposed to construct an (implicit) global signed distance function from the set of acquired range scans in a first step and to reconstruct the (explicit) surface using the Marching Cubes algorithm (Sect. 2.3.2).

Similarly, many approaches for triangulating unorganized point clouds also extract the kernel surface of a properly generated signed distance field. They differ in how they reconstruct and represent the implicit distance function, for instance by using a piecewise linear approximation [HDD+92], radial basis functions [CBC+01], or a spatial decomposition with separate quadratic approximants in each leaf node [OBA+03].

Besides from their generation from scanned data, implicit distance fields are also the natural surface representation in the field of constructive solid geometry (CSG) or general solid modeling [Hof89]. In this context a complex object is built by boolean combinations of several simpler primitives. If $F_1(\mathbf{x})$ and $F_2(\mathbf{x})$ denote the signed distance functions corresponding to two objects $\mathcal{S}_1$ and $\mathcal{S}_2$, then the following combinations of implicit functions yield the union, intersection, and difference of the two objects, respectively:

$$
\begin{aligned}
F_{\mathcal{S}_1 \cup \mathcal{S}_2}(\mathbf{x}) &= \min\{F_1(\mathbf{x}), F_2(\mathbf{x})\} \\
F_{\mathcal{S}_1 \cap \mathcal{S}_2}(\mathbf{x}) &= \max\{F_1(\mathbf{x}), F_2(\mathbf{x})\} \\
F_{\mathcal{S}_1 \setminus \mathcal{S}_2}(\mathbf{x}) &= \max\{F_1(\mathbf{x}), -F_2(\mathbf{x})\} \quad .
\end{aligned}
$$

No matter how and from which data the signed distance field was built from, the explicit surface is generated by extracting the zero-level iso-surface using a contouring algorithm like Marching Cubes.

3.2 Mesh Optimization

The methods outlined in the last section provide an initial triangle mesh representation of a (re-)constructed surface geometry. However, the resulting meshes are in general not of sufficient quality for further down-stream applications, but instead have to be optimized w.r.t. different quality criteria depending on the specific target application.

Meshes derived from a physical scanning process inevitably contain a certain amount of high-frequency measurement noise, which has to be smoothed out by low-pass filtering the geometry (Sect. 3.2.1). After this geometric optimization, further topological optimizations improve or adjust the tessellation of the surface depending on the application's needs, e.g., by reducing the mesh complexity (Sect. 3.2.2) or by improving the triangle shapes by remeshing techniques (Sect. 3.2.3).

3.2.1 Smoothing

In a typical mesh optimization pipeline mesh smoothing is mainly used to remove noise caused by physical measurements from a surface. But the notion of smooth or *fair* surfaces is also of major relevance for freeform modeling (Chap. 6) and multi-resolution modeling (Chap. 5), and hence we discuss mesh smoothing in more detail.

If we consider a triangle mesh as an irregularly sampled signal, then we are interested in smoothing out the high frequency details (*noise*) with the constraint to preserve the lower frequency components (*global shape*). For a mathematically well-founded treatment of this problem, standard signal processing techniques are extended to functions defined on

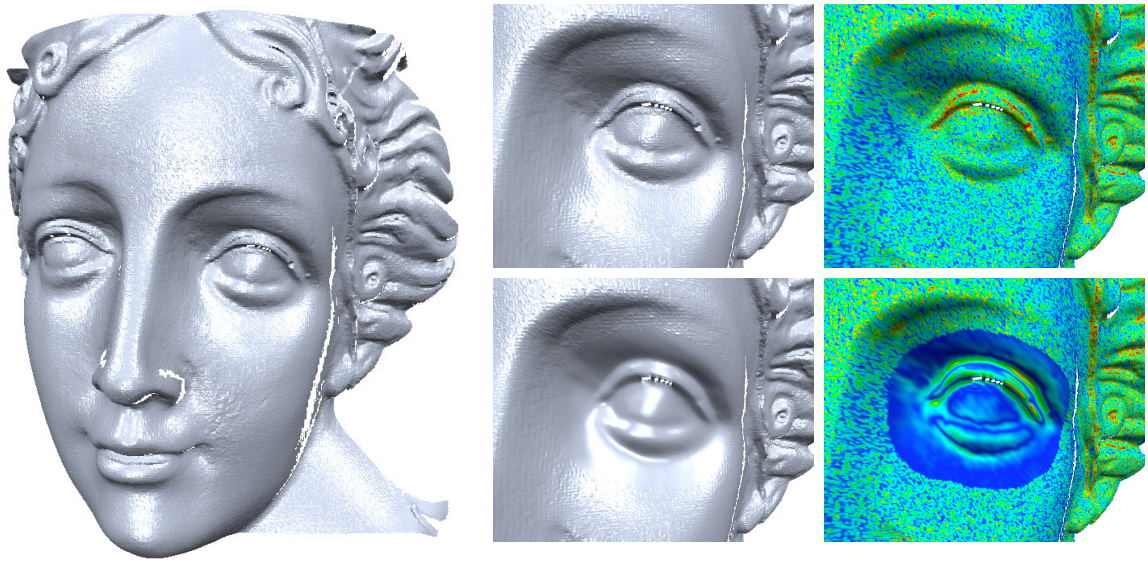


Figure 3.2: This scan of a statue's face contains typical measurement noise, which can be removed by low-pass filtering the surface geometry. The bottom row shows selective smoothing of the region around the eye, the right column depicts a color coding of the respective mean curvature.

triangle meshes, i.e., on irregularly sampled two-manifold domains of arbitrary topology. This will allow us to smooth arbitrary scalar functions $f : \mathcal{S} \rightarrow \mathbb{R}$ on meshes. A smoothing of the surface geometry can then be achieved by a component-wise smoothing of the piecewise linear geometric realization function $\mathbf{p}(v) = (x(v), y(v), z(v))^T$ (cf. Fig. 3.2).

Ideal low-pass filtering means to transform the signal f from spatial domain to frequency domain using the (discrete) Fourier transform, to discard the high frequency components, and to map the truncated signal back to spatial domain by the inverse Fourier transform. The orthonormal basis functions of the frequency domain are built from scaled and shifted sine waves. Having the property that their second derivatives are multiples of themselves, these functions can be found to be the eigenfunctions of the Laplace operator $\Delta f(u,v) = \operatorname{div}(\nabla f(u,v)) = f_{uu}(u,v) + f_{vv}(u,v)$.

Since we are considering the Laplace of functions defined on a two-manifold surface $\mathcal{S}$, the Laplace-Beltrami operator $\Delta_{\mathcal{S}}$ [dC76] actually has to be used. Assuming a proper

generalization and discretization of the Laplace-Beltrami on discrete triangle meshes, the ideal filtering framework could be transferred to meshes as well. However, it would be too expensive for complex meshes in terms of both computation time and memory consumption.

Therefore Taubin [Tau95] proposed to low-pass filter the surface signal f by a convolution with a Gaussian filter kernel instead. This low-pass filter is equivalent to iterative *Laplacian smoothing* [Tau95, KCVS98], that is defined by the following simple update rule for each vertex $v_i \in \mathcal{V}$:

$$f(v_i) \leftarrow f(v_i) + \lambda \Delta_{\mathcal{S}} f(v_i) \quad , \tag{3.1}$$

were $0 < \lambda < 1$ denotes a time-step or damping factor, that has to be less than 1 to guarantee convergence. Another approach to mesh smoothing was given by Desbrun et al. [DMSB99] by formulating surface smoothing as a diffusion process:

$$\frac{\partial f(v_i)}{\partial t} = \lambda \Delta_{\mathcal{S}} f(v_i) \quad . \tag{3.2}$$

When the signal to be smoothed is the surface geometry itself, this means that in each time-step each vertex is moved by a scalar multiple of its Laplace-Beltrami, leading to the so-called *mean curvature flow* [dC76]:

$$\frac{\partial \mathbf{p}(v_i)}{\partial t} = \lambda \Delta_{\mathcal{S}} \mathbf{p}(v_i) = -2\lambda H(v_i)\mathbf{n}(v_i) \quad .$$

Smoothing is then performed by integrating this PDE over time. In this context, the simple Laplacian smoothing update (3.1) corresponds to an explicit forward Euler integration of the PDE (3.2), requiring a time-step $\lambda < 1$ to ensure convergence. To allow for arbitrary large time-steps, Desbrun et al. proposed to use an implicit integration instead. The price to pay is that this approach requires to solve the following sparse linear system in each time-step, where $\mathbf{f}^n = (\ldots, f(v_i), \ldots)$ denotes the vector containing the surface signal sampled at the vertices $v_i \in \mathcal{V}$ at time-step n:

$$(I - \lambda \Delta_{\mathcal{S}})\, \mathbf{f}^{n+1} = \mathbf{f}^n \quad . \tag{3.3}$$

The last missing component is a proper generalization of the Laplace-Beltrami operator to discrete triangle meshes, i.e., for each vertex $v \in \mathcal{V}$. Taubin [Tau95] proposed the uniform discretization of the Laplace

$$\Delta_{uni} f(v) := \frac{1}{|N_1(v)|} \sum_{v_i \in N_1(v)} (f(v_i) - f(v)) \quad , \tag{3.4}$$

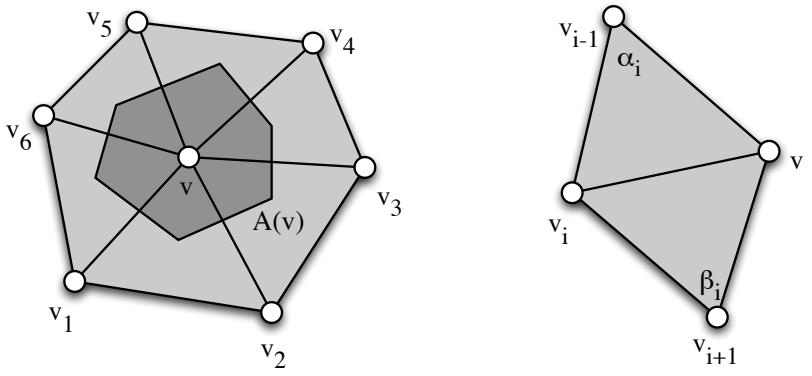


Figure 3.3: The Laplace-Beltrami $\Delta_{\mathcal{S}} f(v)$ of a vertex $v \in \mathcal{V}$ is computed by a linear combination of its function value $f(v)$ and those of its one-ring neighbors $f(v_i)$. The corresponding weights are given by the cotangent values of α_i and β_i and the Voronoi area $A(v)$.

where the sum is taken over all one-ring neighbors $v_i \in N_1(v)$ (cf. Fig. 3.3). However, this discretization does not take any local geometry of the domain mesh (edge lengths or angles) into account and hence cannot give a sufficient approximation for irregular tessellations. In case of smoothing a planar (and hence perfectly smooth) triangulation, e.g., this operator may still shift vertices within the surface by moving each vertex to the barycenter of its neighbors. Although this leads to an improvement of the triangle shapes, it is a bad approximation to the Laplacian of the geometry (which should be parallel to the surface normal: $\Delta_{\mathcal{S}} \mathbf{p}(v) = -2\, H(v)\, \mathbf{n}(v)$). A better (and the current standard) discretization was proposed in [PP93, DMSB99, MDSB03]:

$$\Delta_{\mathcal{S}} f(v) \;:=\; \frac{2}{A(v)} \sum_{v_i \in N_1(v)} \left(\cot\alpha_i + \cot\beta_i\right) \left(f(v_i) - f(v)\right) \quad ,$$

where $\alpha_i = \angle\left(\mathbf{p}(v), \mathbf{p}(v_{i-1}), \mathbf{p}(v_i)\right)$, $\beta_i = \angle\left(\mathbf{p}(v), \mathbf{p}(v_{i+1}), \mathbf{p}(v_i)\right)$, and $A(v)$ denotes the Voronoi area around the vertex v (cf. Fig. 3.3). We discuss the linear system (3.3) required for the mean curvature flow integration and present efficient techniques for its solution in in Chap. 7.

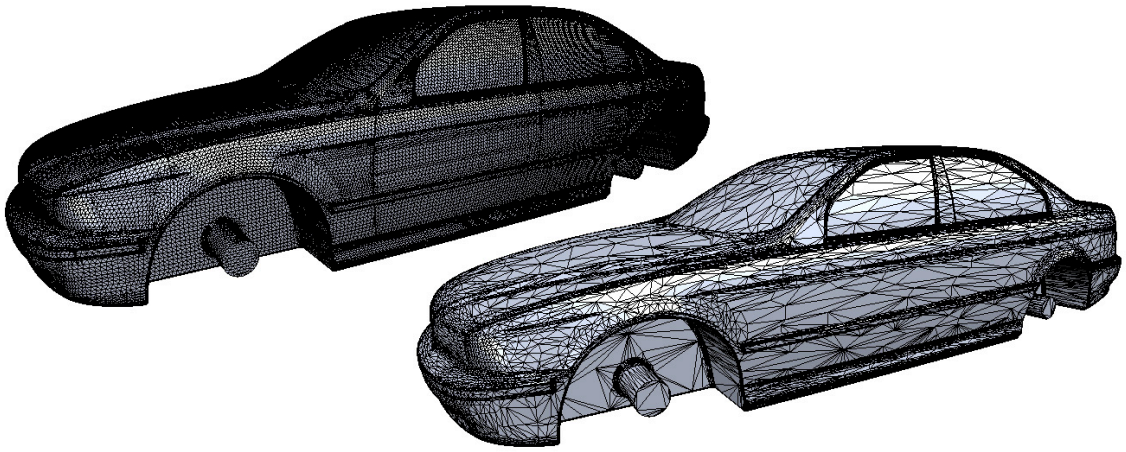


Figure 3.4: Mesh decimation removes geometric redundancy by reducing the sampling density in flat surface regions. In this example a tessellated CAD surface was reduced from 260k triangles to 35k triangles while satisfying a user-defined error bound of 3mm for the positions and 3 degrees for the normal deviation.

3.2.2 Decimation

One problem of most surface reconstruction schemes is that they produce meshes of enormous complexity. Most mesh generation algorithms allow the user to control the output complexity by globally adjusting the resolution, e.g., for the scanning device or for an intermediate volumetric grid representation. However, this resolution can only be changed globally and hence one either loses relevant geometric detail (if the resolution is set too low) or flat surface regions are extremely oversampled (if the resolution is too high). Locally adapting the resolution is difficult since this requires to detect the presence of fine detail, i.e., to estimate the surface curvature, *before* the surface is actually generated.

To avoid these difficulties, the mesh resolution is chosen as high as possible in order to capture as much geometric detail as possible [LPC+00], thereby leading to very complex meshes, whose sampling densities do not reflect the geometric complexity of the underlying surface. As a consequence, these meshes contain a large amount of redundancy, such that their complexity can be reduced significantly by mesh decimation techniques without losing relevant geometric details (cf. Fig. 3.4).

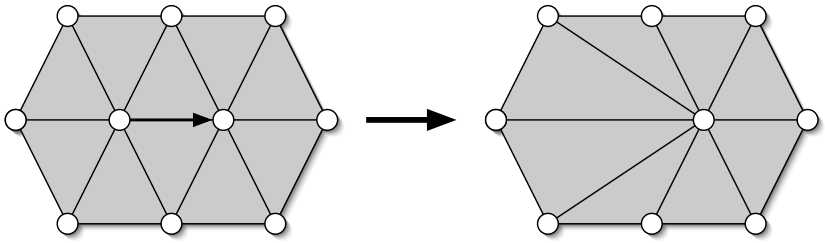


Figure 3.5: The halfedge collapse removes one vertex, two triangles, and three edges.

From the large set of different mesh decimation approaches [Gar99], we will focus on *incremental decimation* based on halfedge collapses [KCS98], since this allows for the most fine-grained control over the simplification process. These methods iteratively remove one vertex (and hence two faces, three edges) by a halfedge collapse at a time (cf. Fig. 3.5). This atomic vertex removal operation is repeated until either the target complexity is reached or a prescribed error tolerance would be violated by any additional collapse.

The two decisions that mainly influence the decimation results are what halfedges to collapse, and in which order to collapse them. Between all possible collapses, a set of *binary* criteria is used to filter out the unwanted ones: each candidate halfedge collapse is simulated and the resulting local configuration is tested not to violate the legality criterion, e.g., a geometric error bound. All valid collapses (that pass the binary tests) are sorted w.r.t. a set of *continuous* criteria. The highest-rated valid halfedge collapse is then performed and the costs of affected candidate collapses in its local neighborhood are re-evaluated and re-sorted. Hence, the decimation is guided and controlled by the different criteria. We list the most important ones below, that can all be used both for a binary legality check and as a continuous sorting priority.

Error Control If the decimated model is to be used in engineering applications, then the approximation error has to be bounded in most cases. The intuitive — but usually too complex — error measure is the two-sided Hausdorff distance [KLS96]. However, in the case of scanned models, Kobbelt et al. [KCS98] argue that the one-sided Hausdorff distance from the original sample points is a sufficient error criterion, similar to scattered data approximation. A very efficient but only approximate error measure is given by the so-called *quadric error metric* [GH97], that approximates the L^∞ distance from a

set of triangles by the L^2 distance from a set of planes. We will discuss more general error control mechanisms, that can be used to bound the global error of an arbitrary mesh processing algorithm, in Sect. 3.3.

Normal deviation Bounding the approximation error results in an adaptation of the vertex density to the surface curvature. This alone, however, cannot guarantee a sufficient preservation of sharp or highly curved surface features. The visual appearance of these features depends on the lighting, i.e., on the surface normals. But even when the approximation error is tightly bound, the normal field of the resulting surface may still deviate significantly [KBSS01]. As a consequence, the error of the normal field approximation has to be bound as well, if geometric features are to be faithfully preserved [BK01a]. Recently, Cohen-Steiner et al. [CSAD04] even use the L^2 normal error alone as a measure of geometric deviation. Transferring this idea to the mesh decimation setting, we use the accumulated deviation of triangle normals as a continuous decimation priority and the geometric approximation error as binary legality criterion, leading to a high quality approximation within a certain error bound, that minimizes the deviation of the surface normal field.

Triangle roundness The robustness of numerical computations on triangle meshes strongly depends on the triangles' shapes: For degenerate triangles neither area nor normal vectors or other derivative information can be computed, while equilateral triangles are optimal for numerical stability. Since it is in general very complex to remove these degeneracies from a given triangle mesh [BK01b], one should consequently avoid creating them. Hence, if a decimated mesh is to be used for any kind of numerical simulation, then the shape of the triangles has to be controlled in order to avoid skinny triangles. There are several ways to measure (and hence to control) the roundness of a triangle, e.g., the ratio of the radius of the triangle's circum-circle to its shortest edge length, or the ratio of its longest edge to its shortest height (so called *aspect ratio*).

Edge Length If the shape quality of the resulting mesh is more important than an efficient approximation in terms of approximation error for a given vertex budget, then high variations in edge length should be avoided. Sorting the candidate halfedge collapses by increasing edge length and controlling the triangle roundness like in the previous

paragraph leads to high quality tessellations that are especially suited for numerical computations.

Building a mesh decimation framework based on the described components allows to generate coarser approximations with exact control over the approximation error, the deviation of the surface normal field, and the roundness of triangles. Using a decimation based on halfedge collapses, the vertices of the resulting coarser meshes are a subset of the original vertices. This can be seen as an advantage, since as much as possible of the original surface information is preserved, but it also limits the degrees of freedom for generating superior tessellations. If triangle shape is the overall priority, then more general remeshing schemes may be better suited, as shown in the next section.

3.2.3 Isotropic Remeshing

In contrast to mesh decimation, which removes vertices from an existing triangulation, remeshing methods are more general, because they can also insert new vertices into the mesh in order to have more degrees of freedom for optimizing the tessellation. Therefore remeshing techniques can be thought of to spread new vertices over a given surface and to connect them in a topologically consistent manner, e.g., by building a kind of Voronoi diagram and Delaunay triangulation on the surface [EDD+95]. While there are many possible remeshing objectives, we will concentrate on *isotropic remeshing*, i.e., on generating a uniform sampling with all triangles being close to equilateral.

If a global parameterization of the surface is available, this problem can be reduced to a re-sampling and re-triangulation of the two-dimensional parameter domain [ACSD+03, ACdVDI03, AMD02]. However, as mentioned in Sect. 2.1.3, the construction of a suitable global parameterization is an equally hard problem. In order to avoid the expensive global parameterization step, several authors base the remeshing on local mesh operations (cf. Fig. 3.6) and local patch parameterizations instead [WW94, KBS00, SG03, SAG03, VRS03].

Surazhsky et al. [SG03] first adjust the number of vertices to a given target complexity by edge collapses or edge splits. Then the vertices are moved on the surface such that all triangle areas are equalized, possibly leading to very skinny triangles. Therefore the triangulation is improved by a sequence of edge flips which increase the minimal inner angle. Finally, an angle-based vertex relocation tries to move vertices on the surface in

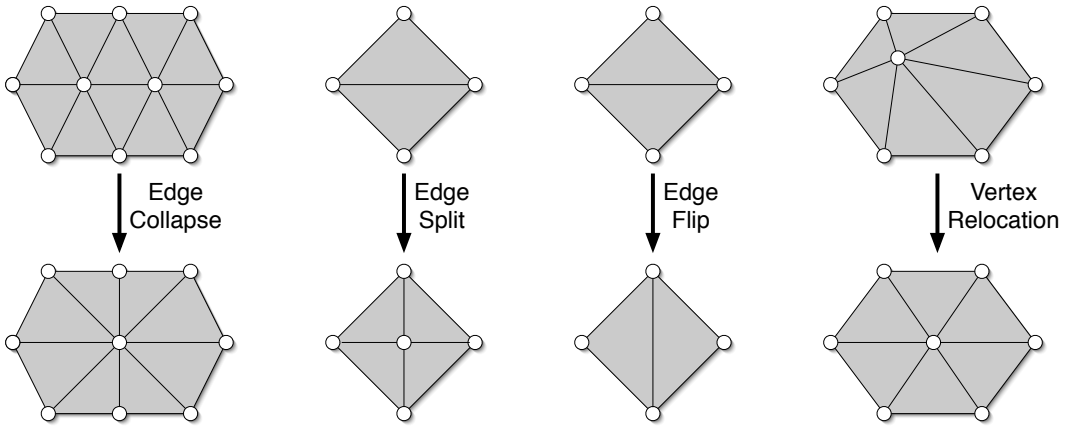


Figure 3.6: The different local mesh operations used for remeshing.

order to equalize all inner triangle angles. In [SAG03] this method has been combined with [ACdVDI03] to generate a centroidal Voronoi diagram on the surface, the dual of which yields an extremely regular tessellation. These techniques lead to regular high quality triangulations, but are computationally very expensive.

In our remeshing approach [BK04b], we follow the more intuitive and conceptually simpler remeshing framework of [KBS00, VRS03] instead. Given a target edge length l, we iteratively perform the following steps to generate a regularly remeshed surface:

1. Edge length equalization by edge splits and edge collapses.
2. Valence regularization by edge flips.
3. Improving the vertex distribution by tangential smoothing.

Usually, 5–10 iterations of these three steps are sufficient to yield a high quality remeshed surface.

The use of local remeshing operators (instead of a global parameterization) allows for a highly efficient implementation, whose computational effort depends on the complexity and degeneracy of the input mesh and on the target edge length l. Hence, it is hard to give precise timings, but as a general rule of thumb, typical meshes of about $100k$ triangles can be processed within 5–10 seconds.

In the following we describe the three remeshing phases in more detail.

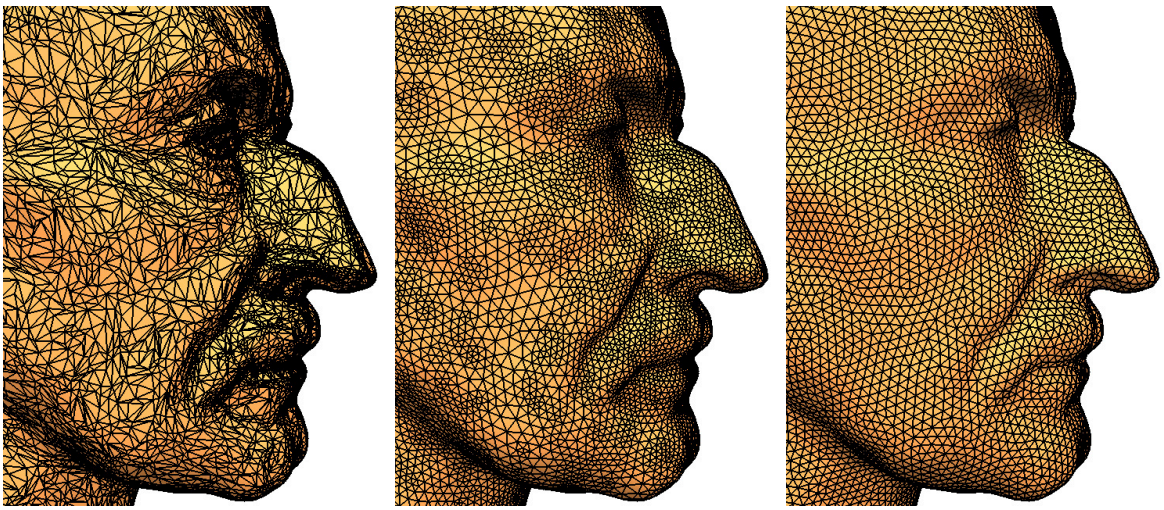


Figure 3.7: The irregular input mesh of 100k triangles (*left*) has been remeshed in about 5s using sequences of local operations. While the edge length thresholds $\left(\frac{1}{2}l, 2l\right)$ lead to local vertex clusters (*center*), the optimal thresholds $\left(\frac{4}{5}l, \frac{4}{3}l\right)$ result in a very uniform sampling (*right*).

Edge length equalization

In order to bring the edge lengths closer to the target edge length l, we intuitively split edges being too long and collapse edges being to short. In a first pass we split all edges at their midpoints that are longer than $l_{\max}$. After that we collapse all edges shorter than $l_{\min}$ into their midpoint (cf. Fig. 3.6). As a result the edge lengths get closer to l. The important question is how to choose the two thresholds $l_{\min} < l < l_{\max}$.

The intuitive thresholds would be $l/2$ and $2l$, but this still leaves a noticeable variation of edge lengths, resulting in a clustering of vertices in certain mesh regions (cf. Fig. 3.7, center). However, optimal thresholds can easily be derived by considering the average local edge length deviation before and after an edge split or edge collapse, respectively (cf. Fig. 3.8). Splitting an edge of length $l_{\max}$ improves the local situation if $|l_{\max} - l| > \left|\frac{1}{2}l_{\max} - l\right|$, leading to an upper threshold of $l_{\max} = \frac{4}{3}l$. The lower bound $l_{\min} = \frac{4}{5}l$ is derived similarly from $|l_{\min} - l| > \left|\frac{3}{2}l_{\min} - l\right|$. The meshes obtained from these optimal thresholds provide a superior uniformity of edge lengths (cf. Fig. 3.7, right).

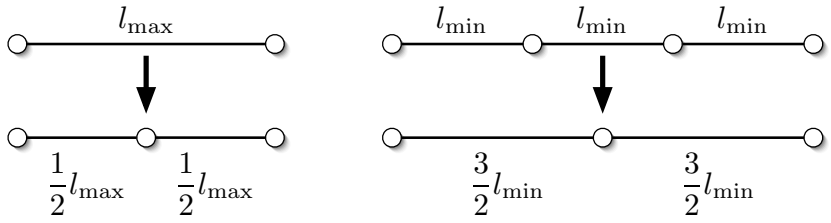


Figure 3.8: These local configurations can be used to derive the (heuristically) optimal values for the edge length thresholds $l_{\min}$ and $l_{\max}$.

Connectivity regularization

The Euler characteristic for polygon meshes [Cox89] states that in a perfectly regular triangulation the valence of inner vertices is 6 and that of boundary vertices is 4. If we flip an edge between two triangles, four vertices are affected (cf. Fig. 3.6): The valences of two vertices are increased by one, the valences of the other two vertices are decreased by one. The regularity of vertex valences can therefore be improved by flipping all the edges for which this operation reduces the valence excess

$$\sum_{v_i \in \mathcal{V}} \left(\text{valence}\left(v_i\right) - \text{optimal\_valence}\left(v_i\right)\right)^2$$

from the optimal valences 6 and 4, respectively.

Tangential smoothing

In this step vertices are moved *on* the surface in order to further improve their distribution and achieve a more uniform surface sampling. As mentioned in Sect. 3.2.1, mesh smoothing using the *uniform* discretization of the Laplacian (Eq. (3.4)) does not only smooth the geometry, but additionally improves the tessellation by moving each vertex to the barycenter of its one-ring neighbors:

$$\mathbf{p}\left(v_i\right) \leftarrow \mathbf{p}\left(v_i\right) + \lambda \Delta_{uni}\, \mathbf{p}\left(v_i\right) \quad .$$

Restricting this operation to vertex movements within the surface preserves the geometry and only improves the vertex distribution. This can be achieved by either using local

parameterizations [SG03, SAG03, VRS03] or by projecting the Laplacian update vector back into the tangent plane (given by the vertex normal $\mathbf{n}(v_i)$) [WDSB00, BK04b]:

$$\mathbf{p}(v_i) \leftarrow \mathbf{p}(v_i) + \lambda \left(I - \mathbf{n}(v_i)\, \mathbf{n}(v_i)^T \right) \Delta_{uni}\, \mathbf{p}(v_i) \quad .$$

Due to the local linear approximation this simple method does not manage to keep the vertices exactly on the original surface, therefore a post-processing step is added at the end of the remeshing procedure that orthogonally projects each vertex back onto the original reference surface.

Area-weighted tangential smoothing

The method presented so far yields a regular remeshing result with all edge lengths being close to the target value l and all inner triangle angles being close to 60°. However, as we showed in [BK04b], the uniformity of the vertex distribution can still be improved, since it does not only depend on the edge lengths, but also on the Voronoi area around each vertex (cf. Fig. 3.3).

These areas may still differ noticeably, since a vertex of valence k is surrounded by k almost equilateral triangles with edge lengths close to l. As a consequence, a vertex of valence 7 will have a larger Voronoi area than a vertex of valence 6 or 5. By consequence, all irregular vertices (with valences different from 6) result in a locally imbalanced sampling density. This means that we have to trade-off equilateral triangles against equal Voronoi areas, i.e., equalized vertex distributions.

To account for this we propose a fine-tuning by an area-based tangential smoothing. Each vertex v_i is assigned a gravity that equals its Voronoi area $A(v_i)$. Our adjusted tangential smoothing moves each vertex to the *gravity-weighted* centroid of its one-ring neighbors

$$\Delta_{grav}\, \mathbf{p}(v) \;:=\; \frac{1}{\sum_{v_i \in N_1(v)} A(v_i)} \sum_{v_i \in N_1(v)} A(v_i) \left(\mathbf{p}(v_i) - \mathbf{p}(v) \right) \quad .$$

To ensure a tangential smoothing *on* the surface, the update vector is again projected into the tangent plane. Vertices with large Voronoi area have a higher gravity and therefore attract their surrounding vertices, thereby reducing their own area. Usually very few (about 5) iterations of this area-weighted tangential smoothing are sufficient to reduce the total variation of vertex areas by a factor of about 5 (cf. Fig. 3.9).

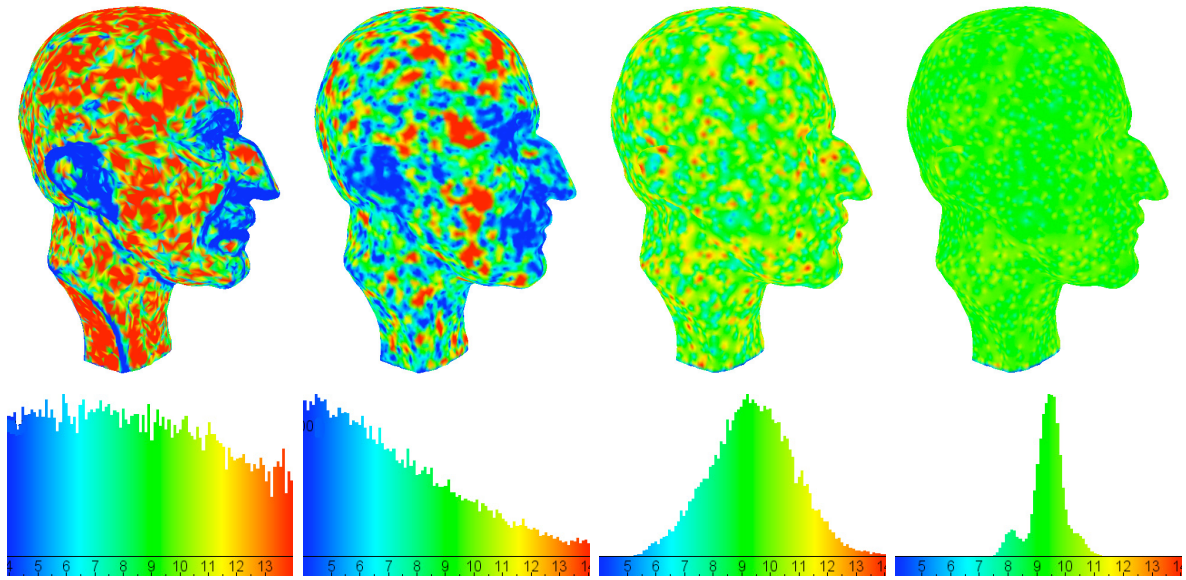


Figure 3.9: Area-weighted tangential smoothing equalizes the Voronoi areas of vertices and thereby improves their global distribution. From left to right: original mesh, naïve thresholds $\left(\frac{1}{2}, 2\right)$, optimal thresholds $\left(\frac{4}{5}, \frac{4}{3}\right)$, and optimal thresholds plus area-weighted smoothing. The images show a color coding of the Voronoi areas (*top row*) and their respective histograms (*bottom row*). The respective relative mean deviations from the target edge length are 148%, 55%, 27%, and 21%. The mean deviations of inner angles from 60° are 26.6°, 6.7°, 4.0°, and 5.6°. The relative mean deviations from the mean Voronoi area are 65%, 34%, 13%, and 4%.

Feature-sensitive remeshing

The main objective for the presented isotropic remeshing is the generation of a high quality tessellation with close-to equilateral triangles. However, if the original surface is a technical dataset containing sharp geometric features, these have to be preserved faithfully. Since the remeshing is performed by a set of local geometric or topological operations, slightly adjusting these operators easily allows for a feature-sensitive variant of the remeshing algorithm.

Following the general ideas of Vorsatz et al. [VRS03], a set of feature edges and feature vertices is marked on the original surface based on the dihedral angle between adjacent triangles. These feature lines can be preserved by the following simple rules:

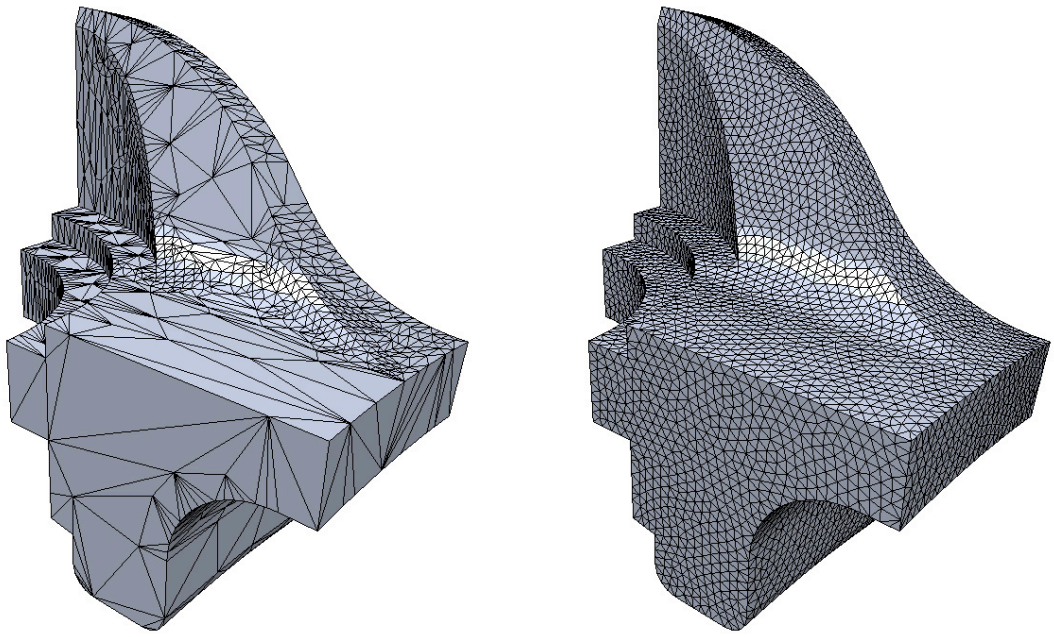


Figure 3.10: The iterative remeshing can easily be modified to preserve sharp features by using special rules for the processing of feature edges or feature vertices for each local remeshing operator.

- Corner vertices, i.e., vertices with more than two incident feature edges, have to be preserved and are excluded from all topological and geometric operations.
- Vertices on feature edges are only collapsed along their two incident feature edges. Non-feature vertices, however, are allowed to be collapsed into a feature vertex.
- Splitting a feature edge creates two new features edges and one new feature vertex.
- A feature edge is never flipped. In order to avoid caps opposite to feature edges a special edge split has to be used instead.
- The tangential smoothing of vertices on feature edges is restricted to a univariate smoothing along the corresponding feature lines.

These simple rules cause almost no performance penalty and allow for a high quality isotropic remeshing of technical datasets. As an example a feature-sensitive remeshing of a decimated version of the well-known fandisk dataset is shown in Fig. 3.10.

3.3 Global Error Control

The last sections showed that most meshes resulting from a surface reconstruction process, no matter whether an explicit or implicit approach was chosen, have to be optimized in several ways before they are ready to be used in engineering applications. Especially for this kind of applications, a prescribed approximation tolerance to the original reference geometry must not be violated, otherwise the simulation's results might become meaningless. As a consequence, it is crucial to provide an *exact* or at least conservative *global error* bound for all mesh processing algorithms that are applied to the mesh.

As the results of the different mesh optimizations are often hard to predict, they are usually applied repeatedly to (regions of) the input mesh in any order. Even if each of these mesh optimization algorithms provides its own (local) error bound, these individual errors can accumulate during multiple optimization loops, especially if several optimization algorithms are applied alternatingly. Additionally, the individual error measures usually exploit the fact that only a small region of the mesh is modified at one time. For instance, the one-sided Hausdorff distance for mesh decimation [KCS98] can only be computed at reasonable cost, because each halfedge collapse removes just one vertex and all remaining vertices are a subset of the original ones. Hence, only the distances from the already removed vertices to the current mesh have to be considered. However, as soon as different methods are interleaved, these locality assumptions break down. For example, one smoothing iteration changes the positions of all vertices, and one re-meshing step can affect the complete mesh connectivity.

In order to prevent such kind of error accumulation, a global approximation error to the initial reference geometry has to be taken into account. To allow for greatest flexibility, this global error measure should be independent of the individual algorithms to be applied to the mesh. This is provided by the concept of *tolerance volumes* or *simplification envelopes*: An envelope of user-specified thickness $\varepsilon_{\max}$ (the error tolerance) encloses the reference geometry, and for each mesh modification the vertices and triangles are tested to stay within this tol-

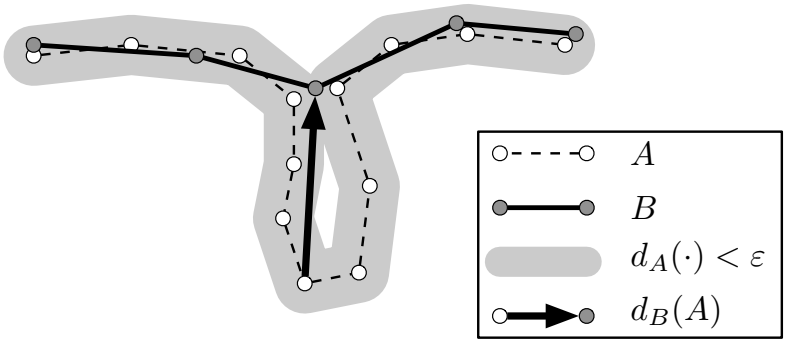


Figure 3.11: The gray tolerance volume around the original surface A guarantees the decimated surface B to stay within a distance $d_A(B) < \varepsilon$ to A. However, this does in general not bound the Hausdorff distance $d_B(A)$ from A to B, as shown by the thick arrow.

erance volume, thus guaranteeing an upper bound on the one-sided Hausdorff distance from the current mesh to the original data. Notice that this error measure is not the same as the one-sided Hausdorff error from the original data to the current mesh as used for mesh decimation (Sect. 3.2.2), but it turned out to be sufficient for all our test cases (cf. Fig. 3.11).

This error measure was used for the *simplification envelopes* of Cohen et al. [CVM+96]. In their work polygonal meshes are used for constructing the simplification envelope and for performing the inside tests for given candidate triangles. Both problems are hard to solve robustly using triangle meshes, as the tolerance volumes corresponds to Minkowski sums of the reference surface and a sphere of radius $\varepsilon_{\max}$, and they are bounded by two offset surfaces in positive and negative normal direction. As a consequence, their method is both algorithmically and computationally very complex.

When comparing the strengths and drawbacks of explicit and implicit surface representations (see Chap. 2), the latter ones are clearly preferable for constructing the tolerance volumes as well as for the required distance queries for candidate triangles. Zelinka and Garland [ZG02] therefore proposed to discretize the characteristic function of the tolerance volume into a uniform binary *permission grid*. In order to check a candidate triangle it is rasterized into the grid and tested to pass only through "valid" grid cells that lie completely inside the tolerance volume. A drawback of this otherwise

very efficient method is that the resulting piecewise constant approximation suffers from aliasing artifacts. To reduce these artifacts, a rather fine grid resolution is needed, which, in turn, is limited by its cubic memory growth.

A more memory efficient representation was proposed by Frisken et al. [FPRJ00], using an adaptively sampled piecewise tri-linear approximation of the signed distance field (ADF). An also piecewise linear, but C^{-1} approximation of the distance field was shown to lead to an even further reduction of memory consumption [WK03]. Although these two approaches consume significantly fewer memory, testing whether a given triangle lies within an approximation tolerance gets more complicated. For instance, restricting an ADF to a candidate triangle results in a piecewise cubic distance function, whose maximum has to be found.

In [BBVK04] we propose an approach that can be categorized to lie between permission grids and the latter two methods. Similar to permission grids, we sample the signed distance function on a regular grid. However, by using a piecewise tri-linear approximation instead of a piecewise constant one, the higher approximation order allows us to work with coarser grid resolutions. Although the distance test for a given triangle is more complicated than in [ZG02], it is simpler compared to [FPRJ00, WK03], since we sample on a regular grid. To check a given candidate triangle, the distance function has to be evaluated and examined on that triangle, which amounts to a tri-linear interpolation of distance values within the cells of the regular 3D grid.

However, this tri-linear interpolation task is exactly what texture units of modern graphics hardware have been optimized for. We therefore propose to represent the piecewise tri-linear distance volume by a three-dimensional texture, as this allows us to exploit the hardware acceleration of modern graphics processors (GPUs). Testing whether a given triangle lies within the tolerance volume then basically amounts to rendering it using the pre-computed 3D distance texture. The required voxelization and tri-linear interpolation will automatically and efficiently be performed by the GPU. The rapidly increasing computational power of GPUs and their flexible programmability in terms of vertex and pixel shaders makes GPUs very suitable for complex streamable computations and even triggered the new GPGPU research field [BGH+04].

An additional obvious application of our approach is the accurate and efficient visualization of the approximation error between two meshes by color-coding the respective per-pixel distance values using high quality post-classification transfer functions known

from direct volume rendering [RS01]. In comparison to a 2D texture based error visualization like in Metro [CRS98], we do not have to pre-compute a per-triangle error texture, but exploit the 3D texturing hardware instead. As a result, we can even visualize the distance of a dynamically changing mesh to a reference surface at a rate of 15M triangles/sec.

In Sect. 3.3.1 we first present the initial generation of the 3D distance texture. Sect. 3.3.2 then describes the implementation of a generic distance check for a given triangle on the GPU. A slightly more detailed explanation with additional implementation notes can be found in [BBVK04]. Using these ingredients our method can be encapsulated into an easy-to-use module for distance checks, which can be incorporated into any mesh processing algorithm, of which we show a few examples in Sect. 3.3.3.

3.3.1 Distance Texture Generation

Given an initial reference surface represented by a triangle mesh, a piecewise linear approximation of its signed distance field is computed on a regular 3D grid by a fast marching method as described in Sect. 2.3.1. In the case of models with boundaries, we can simply fall back to a *unsigned* distance fields instead of a signed ones, which leads to a small over-estimation of the error by $h/2$ in grid cells that are intersected by the reference surface (h denoting the edge length of grid cells). For reasonable grid resolutions, this over-estimation does not lead to problems.

The accuracy of the distance field approximation is determined by the grid resolution R, or by the edge length h, respectively. The tri-linear approximation within a grid cell may under-estimate the exact error by at most one half of the cell diagonal in the worst case. Hence, the user-specified error tolerance is adjusted to

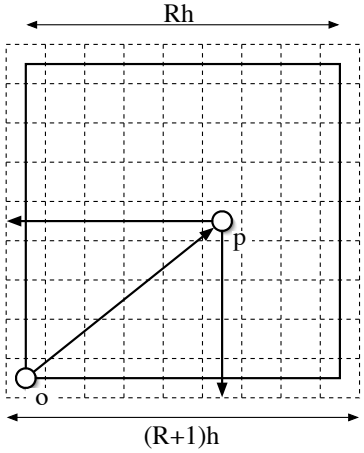
$$\varepsilon_{\max} \;\leftarrow\; \varepsilon_{\max} - \frac{\sqrt{3}}{2}h$$

in order to take this into account. Since the distance field is smooth in most regions, and since it is approximated by a piecewise linear function, the approximation error decreases like $O(h^2)$ when increasing the grid resolution [Dav75]. In contrast, the piecewise constant approximation of [ZG02] improves just linearly. This allows us to use coarser grid resolutions compared to them, as we will see in Sect. 3.3.3.

The resulting regular grid of distance values now has to be used as an OpenGL 3D texture. Until recently, the texture size had to be a power of two in each dimension, but this restriction has been removed by the OpenGL extension `ARB_texture_non_power_of_two`. For better memory efficiency unsigned byte values should be stored in the distance texture by mapping the range $[-\varepsilon_{\max}, \varepsilon_{\max}]$ to the integers $\{0, \ldots, 255\}$, leading to an 8 bit quantization of the acceptable error values. Although this turned out to be sufficient in all experiments, higher precision integer values or even floating point textures can also be used.

3.3.2 Triangle Distance Check

Given a candidate triangle, an error check as well as an error visualization can now be performed by simply rendering this triangle using the pre-computed 3D distance texture. The triangle will automatically be rasterized and the distance values will be fetched from the 3D distance texture using tri-linear interpolation.



In order to properly access the texture, 3D texture coordinates have to be computed from the relative position of a vertex w.r.t. the grid. Since OpenGL assigns texture coordinates to the centers of grid cells (*texels*), instead of to the grid nodes (see [SA03], p. 134), the texture coordinate $\mathbf{t}(\mathbf{p})$ associated to a 3D point $\mathbf{p}$ is its relative position w.r.t. a grid extended by $h/2$ in each dimension:

$$\mathbf{t}(\mathbf{p}) \;=\; \frac{\mathbf{p} - \mathbf{o} + \left(\frac{h}{2}, \frac{h}{2}, \frac{h}{2}\right)^T}{(R+1)h} \;\;,$$

where h again denotes the edge length of a cell, R the grid resolution, and $\mathbf{o}$ the lower left front corner of the grid. The on-the-fly computation of texture coordinates can easily be mapped to the GPU using a small vertex shader [LKM01].

The setup described so far can already be used for pixel-accurate distance visualization. A transfer function which maps the interpolated distance values to a given color range can be represented by a second RGB texture, such that a dependent texture lookup results in the desired color coding of per-pixel distance values (cf. Fig. 3.13).

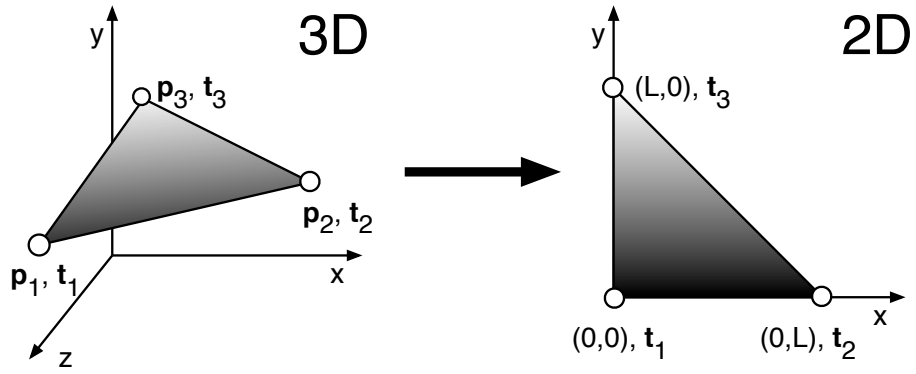


Figure 3.12: Since the pixel colors, i.e., the distance values, depend on the interpolated texture coordinates $\mathbf{t}_i$ only, the vertex positions $\mathbf{p}_i$ can be changed to the simple 2D setup on the right hand side.

This corresponds to high quality post-classification methods frequently used in hardware-accelerated direct volume rendering [RS01].

Basically the same idea is used to test whether or not a given triangle lies completely within a tolerance volume around the reference surface: a special color is assigned to distance values greater than the prescribed tolerance, the candidate triangle is rendered and the framebuffer is examined for this color. As described above, rendering a triangle using the distance texture will automatically interpolate the distance function during rasterization. Notice that the resulting per-pixel distance values depend on the 3D texture coordinates only, such that the vertex positions can be adjusted as long as the texture coordinates stay the same.

However, when checking a given triangle by rendering it, one has to make sure that it is visible and that sufficiently many pixels are generated by its rasterization. Instead of adjusting the camera position to view perpendicular on each candidate triangle, we simply render the 2D triangle $((0,0),(L,0),(0,L))$, but still use the correct texture coordinates $\mathbf{t}(\mathbf{p}_i)$ computed from the corresponding 3D positions (cf. Fig. 3.12). In order to have a sufficient resolution in the rasterization of the candidate triangle, and hence a sufficient sampling of the distance field, the edge length L is determined such

that the pixel resolution meets the texture resolution. If $\mathbf{p}_0$, $\mathbf{p}_1$ and $\mathbf{p}_2$ denote the positions of the triangle's vertices, this edge length is

$$L = \left\lceil \frac{1}{h} \cdot \max\left\{ \|\mathbf{p}_0 - \mathbf{p}_1\|, \|\mathbf{p}_1 - \mathbf{p}_2\|, \|\mathbf{p}_2 - \mathbf{p}_0\| \right\} \right\rceil .$$

In order to detect pixels violating the error bound $\varepsilon_{\max}$ we use a transfer function assigning a completely transparent color ($\alpha = 0$) to distance values less than or equal to $\varepsilon_{\max}$ and an opaque color ($\alpha = 1$) otherwise. Hence, as soon as one pixel is rendered, the candidate triangle violates the error bound. This, however, can easily be checked using occlusion queries (`ARB_occlusion_query`), which return the number of pixels being rendered during a query period.

3.3.3 Applications

The distance texture generation and the generic triangle test can be encapsulated within an easy-to-use global error module. After initializing the distance texture by specifying a reference triangle mesh, the error tolerance $\varepsilon_{\max}$, and the grid resolution, an arbitrary list of triangles can be tested. In addition, the distance texture can also be used to visualize the error by color-coding per-pixel distances.

We integrated this global error plug-in into mesh decimation and mesh smoothing applications (cf. Fig. 3.13). For mesh decimation, a candidate halfedge collapse is tested against the error bound by simulating the collapse and rendering the one-ring triangles of the remaining vertex (cf. Fig. 3.5). The mesh smoothing algorithm, as described in Sect. 3.2.1, computes an update vector for each vertex based on its Laplacian. For each vertex this update step is simulated and its one-ring triangles are tested to stay within the error bound by rendering them.

For rating the efficiency of GPU-based tolerance volumes, we compared the mesh decimation to a one-sided Hausdorff error decimation [KCS98] and to the permission grids of Zelinka and Garland [ZG02]. Our new method is faster by a factor of about 2 compared to the Hausdorff decimation, and faster by a factor of 1.6 than permission grids. Although this improvement is not too impressive, our method can be used for any mesh processing algorithm, in contrast to the one-sided Hausdorff error, that is specialized for mesh decimation. In comparison to permission grids, we achieve comparable decimation results with a grid resolution R being $\frac{1}{3.5}$ of theirs, due to our better approximation of

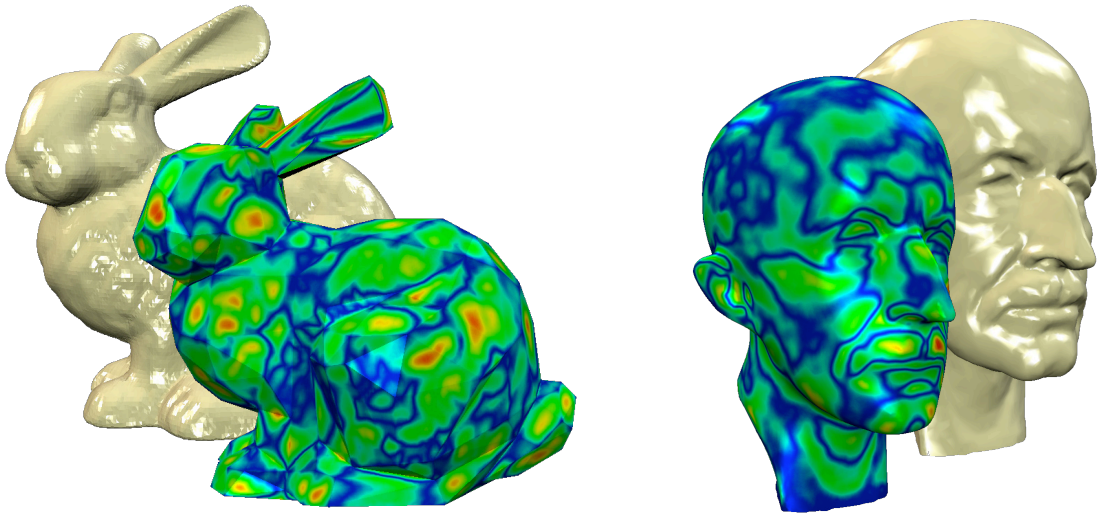


Figure 3.13: The GPU-based tolerance volumes can easily be encapsulated into a framework for error control and error visualization, that can be used by any mesh processing algorithm. The images show examples for mesh decimation (*left*) and mesh smoothing (*right*).

the distance field. Although we use one byte for each grid node instead of one bit only, our memory consumption is still smaller by a factor of about 6.

We also integrated the tolerance volumes into an interactive free-form modeling tool (cf. Fig. 3.14). One drawback of its otherwise intuitive user interface is that it is hard to predict by which amount the surface is changed when some of its points are dragged around. After integrating the global error module, the distance visualization gives real-time feedback to the designer at a rate of about 15M triangles/sec, such that precise deformations can be performed at no additional cost. In addition, a global error check can be applied to all triangles being affected by a deformation using one global query only. Blocking a deformation that would otherwise violate the error tolerance ensures that the deformed model does not deviate too much from its initial state.

These examples point out that the GPU-based tolerance volumes are an efficient and versatile tool for controlling and visualizing the deviation of one mesh to a reference surface. Besides its high efficiency, one of the main advantages of this approach is that it is both easy to use and easy to implement, since all complicated algorithmic tasks are performed by the graphics card. Although texture memory is practically more limited

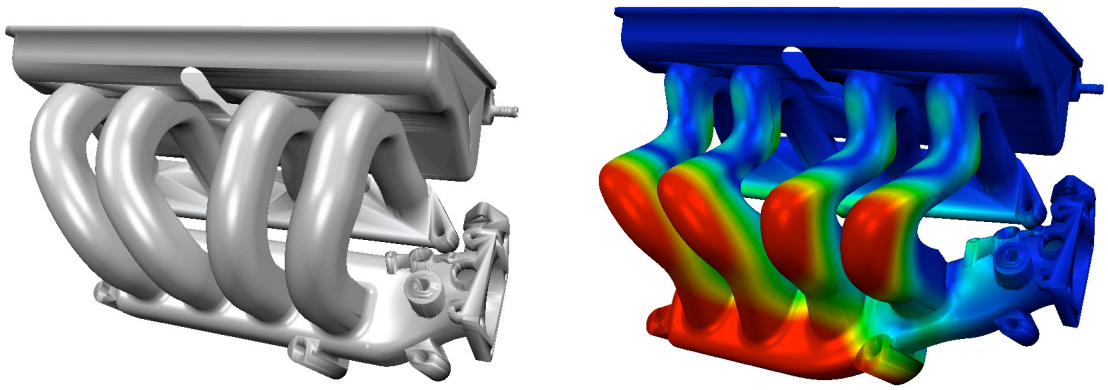


Figure 3.14: The GPU-based tolerance volumes provide real-time visual feedback to the designer during a surface deformation process, no matter which kind of surface deformation technique is used.

than main memory, the possible grid resolutions have proven to be sufficient due to the piecewise linear distance field approximation.

Additionally, the GPU tolerance volumes can be used in combination with any geometry representation that can be rasterized and rendered by OpenGL, possibly at the cost of computing the texture coordinates on a per-pixel level. One particularly interesting alternative to triangle meshes are point-sampled geometry representations, that have become more and more popular within the last years due to the steadily increasing complexities of range-scanned models [KB04]. Exploiting the programmability of modern GPUs, these datasets can nowadays be visualized with the same quality and similar performance as provided by rendering triangle meshes [BK03c, BSK04, BHZK05]. As a consequence, our real-time distance checking and distance visualization can directly be transferred to point-sampled geometries.

4 Feature-Sensitive Mesh Generation

In Chap. 2 we pointed out that triangle meshes are an extremely flexible and powerful surface representation. The mesh generation and mesh optimization algorithms presented in Chap. 3 additionally showed that triangle meshes allow for efficient surface processing and provide high quality approximations of geometric shapes. Because of these strengths, polygonal meshes are increasingly often used in engineering applications and numerical simulations. In this context one has to guarantee that meshes which are to be used for simulations are sufficiently good approximations of their continuous physical counterparts. In the presence of sharp or highly curved features, a high-quality approximation of these geometric features becomes even more important, since they are usually very relevant for the simulation and will therefore have a strong influence on the results.

Unfortunately, faithfully representing sharp or strongly bent surface features, like edges or corners in technical datasets, is also the most complicated sub-task in surface approximation [KB03a]. Since the key to successfully using polygonal meshes in engineering applications is a high quality representation of these geometric features, we first review the approximation properties of polygonal meshes in the presence of features in Sect. 4.1.

We described in Sect. 3.2 that mesh optimization methods like decimation or general remeshing can quite easily be extended to properly preserve *existing sharp* surface features by tagging feature edges based on their dihedral angle and defining special rules for their treatment. This, however, requires the sharp features to exist in the initial mesh, a condition that is not satisfied in most cases: The majority of mesh generation methods is based on an intermediate implicit surface representation and uses a contouring method like Marching Cubes to extract the surface mesh (see Sect. 3.1.2). As briefly mentioned in Sect. 2.3.2, this may lead to severe alias-artifacts in the vicinity of sharp features, since the standard Marching Cubes is not a feature-sensitive algorithm. In Sect. 4.2 we

will therefore describe this problem in more detail and present a solution by enhancing the standard Marching Cubes algorithm to detect and reconstruct sharp features.

Surface features that are not sharp, but highly curved, are similarly complicated configurations for surface approximation techniques. These features cannot simply be detected by a curvature discontinuity, like a large dihedral angle, and therefore are usually not preserved by mesh processing methods. In Sect. 4.3 we derive an optimal sampling pattern for blend regions in technical datasets, which are a typical example for such features, and use this pattern to re-sample existing meshes, such that alias artifacts are effectively reduced.

4.1 Approximation Properties and Normal Noise

From approximation theory it is known that approximating a smooth (sufficiently differentiable) surface $\mathcal{S}$ by a piecewise linear and continuous interpolant $\mathcal{M}$ (a polygonal mesh) converges with quadratic approximation order. However, in the vicinity of sharp features the surface $\mathcal{S}$ is no longer differentiable, since normal vectors are not continuous across that feature. This loss of differentiability causes the approximation power to drop down to linear order, leading to much slower convergence in these areas.

Since sharply curved features correspond to high frequencies of the surface $\mathcal{S}$ and since the polygonal mesh $\mathcal{M}$ is a finite discrete sampling of this surface, signal processing theory tells us that the sampling density has to sufficiently adapt to the signal's frequency spectrum; otherwise it will not be possible to capture all geometric features and the surface will be affected by aliasing artifacts. Following these principles, the sampling density of $\mathcal{M}$ has to be adjusted to the curvature of the underlying surface $\mathcal{S}$, i.e., coarser sampling can be used in smooth and flat areas, but the vertex density has to be increased in highly curved regions of the surface.

Unfortunately, the degenerate case of sharp features of $\mathcal{S}$ corresponds to an infinitely high curvature or frequency, requiring (in theory) an infinitely dense sampling in their vicinity. As Fig. 4.1 clearly depicts, increasing the sampling density results in a sequence of meshes that converge point-wise to $\mathcal{S}$, but whose normal vectors will never converge to the correct normals of $\mathcal{S}$. Hence, refining the mesh will in fact *not* remove the alias-artifacts, it will just shift them to a higher frequency band. Algorithms requiring

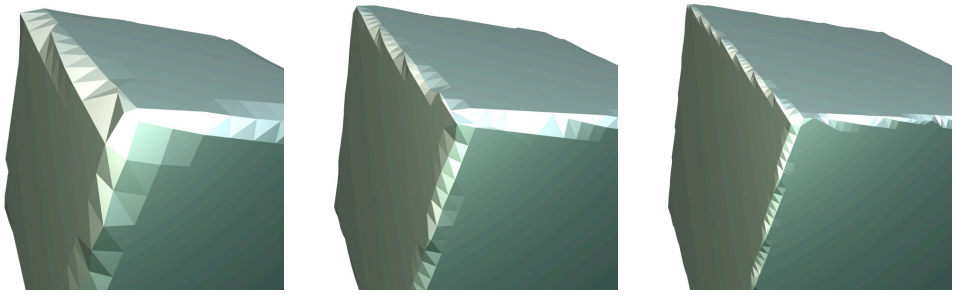


Figure 4.1: Alias error at high-frequency geometric details. By refining the mesh, the effect becomes less and less visible due to the convergence of the mesh $\mathcal{M}$ to the continuous surface $\mathcal{S}$, but the problem is not really solved, since the normal vectors of $\mathcal{M}$ do not converge to the normals of $\mathcal{S}$. The resulting alias-errors can only be removed by placing samples exactly on the sharp features.

derivative information like normal vectors will then give anything but reliable results in these cases. Such methods may be as simple as surface shading, showing specular artifacts, or more sophisticated simulations like computational fluid dynamics (CFD), where these randomly tilted normals may cause erroneous turbulences.

Since already small perturbations of nearby vertex positions can cause large deviations of normal vectors, the approximation error alone is not a sufficient measure for reconstruction quality. Considering the piecewise linear nature of triangle meshes, the global sampling pattern also has to be taken into account — in addition to individual vertex positions only. In [BK01a], we therefore define that a mesh $\mathcal{M}$ is a high-quality approximation of a surface $\mathcal{S}$, if the mesh normals of $\mathcal{M}$ are a subset of the real surface normals of $\mathcal{S}$. If instead the normal vectors are randomly tilted away from the correct direction, we refer to this effect as *normal noise*, similar to surface noise being a high-frequency perturbation of vertex positions (cf. Figs. 4.1 and 4.2). The process of reducing or even removing these alias-artifacts is then called *surface anti-aliasing*.

The amount of normal noise is also a measure for surface quality: high quality surfaces in geometric modeling and CAD are usually characterized by a low variation of curvature, also called *fairness* [MS92, WW92]. In the discrete setting of polygonal meshes one way to derive a discrete analogon to the concept of surface curvature is to consider the *normal jump* across an edge, i.e., the dihedral angle between the normals of two incident faces. If the triangle mesh is an orientable manifold, one can additionally distinguish

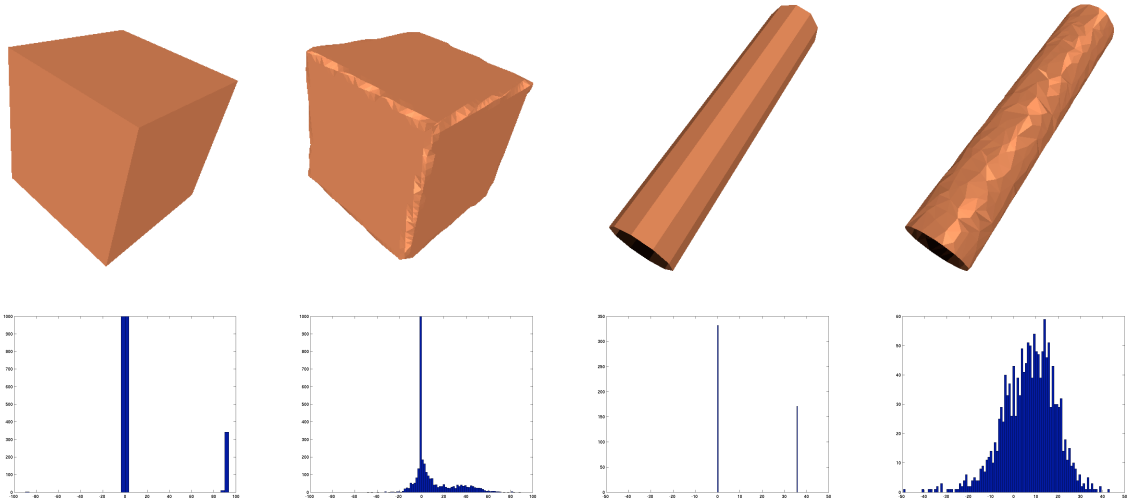


Figure 4.2: Different approximations of a cube and a cylinder and their corresponding histograms of *normal jumps*. Meshes obtained from a feature-sensitive sampling process provide a high approximation quality and are characterized by few peeks in the histogram, corresponding to the different principal curvatures of the underlying surfaces. In contrast, the randomly sampled meshes — although having a higher sampling density — are of lower quality, because they are affected by *normal noise*.

between *convex* normal jumps (positive sign) and *concave* normal jumps (negative sign). A triangle mesh is then said to be of high quality if the variation of its normal jumps is low. For low quality meshes with a strong variation of normal jumps we are back to the notion of normal noise again (cf. Fig. 4.2).

This shows that geometric features are on the one hand the most significant mesh regions for many applications and that they are on the other hand the most difficult regions from an approximation point of view. If we are given a fixed vertex budget and since all samples have to be placed *on* the surface, the only remaining degrees of freedom are to move the vertices *within* the surface, i.e., to choose the sampling pattern.

The only way to generate meshes of superior quality and free of normal noise is to have this sampling pattern aligned to the surface features. Sect. 4.3 shows that for strongly curved features the mesh has to be aligned to the principal curvature directions of the underlying geometry. In the extreme case of infinitely sharp features, surface samples consequently have to be placed exactly on the respective feature edges or corners to

get rid of the alias-artifacts. Building on this fact, we first derive a feature-sensitive iso-surface extraction as an enhancement of the standard Marching Cubes algorithm in the following section.

4.2 Feature-Sensitive Iso-Surface Extraction

As mentioned in Sect. 2.3.2, the Marching Cubes algorithm (MC) is the widely-used standard technique for extracting iso-surfaces from volumetric representations. It is used in many different areas, like medical imaging, CSG solid modeling, and implicit mesh generation. While the organic structures in medical datasets are mostly smooth, the surfaces generated in the latter two cases will contain sharp features in general.

The problem with all Marching-Cubes-like grid-based algorithms is that they are not capable of reconstructing sharp surface features, but instead lead to severe alias artifacts in the vicinity of those features in the reconstructed surface. This is due to the fact that these algorithms process *discrete* volume data and that the sampling of the implicit surface $F(\mathbf{x}) = 0$ is performed on the basis of a *regular* spatial grid.

Fig. 4.3 shows this effect on the well-known fandisk model, which has been converted to an implicit representation by sampling its signed distance field on a uniform $65 \times 65 \times 65$ grid. Using the standard MC algorithm to convert the implicit representation back to an explicit triangle mesh leads to severe alias artifacts and normal noise near the sharp features of the original geometry. As shown in the last section, refining the underlying 3D grid will only reduce the size of these artifacts, but will not solve the basic alias problem (cf. Fig. 4.1).

In [KBSS01] we therefore proposed two enhancements of the standard MC algorithm that allow us to faithfully reconstruct sharp features. The use of *directed distance* fields enables the computation of *exact* surface samples on the grid edges, which is not possible using the standard *scalar* valued distances (Sect. 4.2.1). Second, the *extended Marching Cubes* algorithm (EMC) robustly detects and reconstructs features within grid cells based on the distance field's gradient information (Sect. 4.2.2). Both components can be used independently to improve the surface extraction, but the best results are obtained by combining both components (cf. Fig. 4.3, right). Since the simple algorithmic structure of

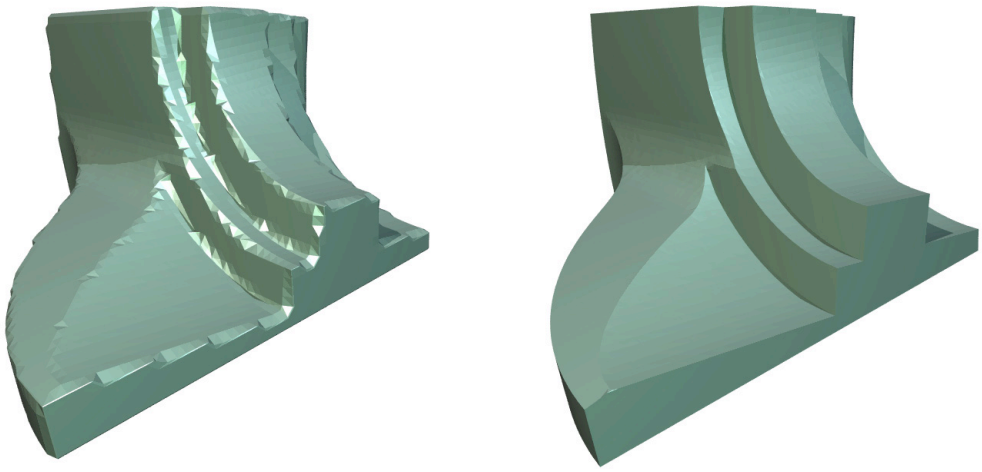


Figure 4.3: Two reconstructions of the "fandisk" dataset from a $65 \times 65 \times 65$ sampling of its signed distance field. The standard Marching Cubes algorithm leads to severe alias artifacts near sharp features (*left*), whereas our feature-sensitive iso-surface extraction faithfully reconstructs them (*right*).

the MC is preserved, our feature-sensitive isosurface extraction can replace the standard MC in many applications, which we demonstrate in Sect. 4.2.3.

4.2.1 Directed Distance Fields

When discussing implicit surfaces in Sect. 2.2, it turned out that from all possible implicit functions representing the outer surface $\mathcal{S}$ of a solid object, the signed distance field is the most natural one. In order to apply the standard MC algorithm, the signed distance field F is sampled on a (uniform or adaptive) spatial grid with nodes $\mathbf{g}_{i,j,k} = (ih, jh, kh)^T$, and the resulting distance values $d_{i,j,k} := F(\mathbf{g}_{i,j,k})$ are tri-linearly interpolated within the cells' interiors.

For each edge intersected by the zero-level isosurface $\mathcal{S}$, a sample point approximating this intersection is computed by a linear combination of its endpoints' distance values (see Sect. 2.3.2). However, the resulting samples are not necessarily close to $\mathcal{S}$ in the vicinity of sharp features, as shown in a two-dimensional example in Fig. 4.4. Since both grid points find their closest surface point (i.e., their minimum distance) in differ-

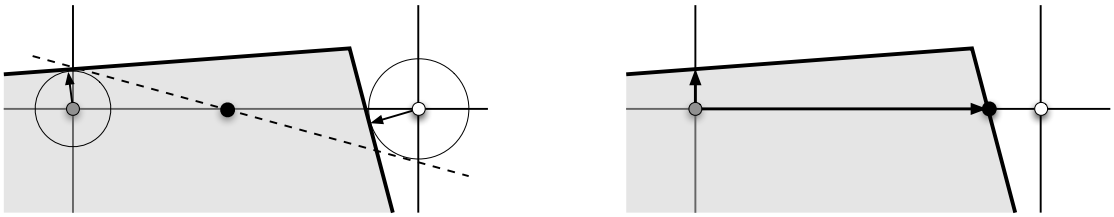


Figure 4.4: Consider two neighboring grid points in the vicinity of a sharp corner of the contour $\mathcal{S}$. Sampling the *scalar-valued* distance function F at both grid points (*circles*) and estimating the sample point by linear interpolation leads to a bad estimation (*black*) of the true intersection point between the contour $\mathcal{S}$ and the cell edge (*left*). Using *directed* distance values at each grid point allows to compute exact intersection points of the contour $\mathcal{S}$ with the grid's edges (*right*).

ent directions, which cannot be captured by scalar valued distances, the simple linear interpolation fails. Fig. 4.5 shows the same effect on a three dimensional example.

The distance field approximation can be improved by using a different discretization of F, the so-called *directed distance field*. Since the MC algorithm computes sample points on the grid edges only, it is sufficient to also restrict the approximation of F to these edges. The directed distance field stores at each grid point $\mathbf{g}_{i,j,k}$ three directed distances in positive x, y, and z direction instead of the scalar valued distances $d_{i,j,k}$, i.e.,

$$\mathbf{d}_{i,j,k} = \begin{pmatrix} \text{dist}_x(\mathbf{g}_{i,j,k}, \mathcal{S}) \\ \text{dist}_y(\mathbf{g}_{i,j,k}, \mathcal{S}) \\ \text{dist}_z(\mathbf{g}_{i,j,k}, \mathcal{S}) \end{pmatrix} .$$

The sign of the distance values again indicates whether a grid point lies inside or outside of the object, and hence the processing of directed distance fields is basically identical to that of scalar distance fields. For instance, the min/max computations for the boolean operations of CSG modeling (see Sect. 2.2) just have to be applied component-wise to the directed distances.

The MC algorithm can be applied to the directed distance field data structure without significant modifications. The local configuration can still be derived from the sign pattern at the cell's corners, since the three directed distances at one grid point always have the same sign (inside/outside status). In order to compute an intersection point on an edge, its position can now be determined *exactly* by the directed distances along that

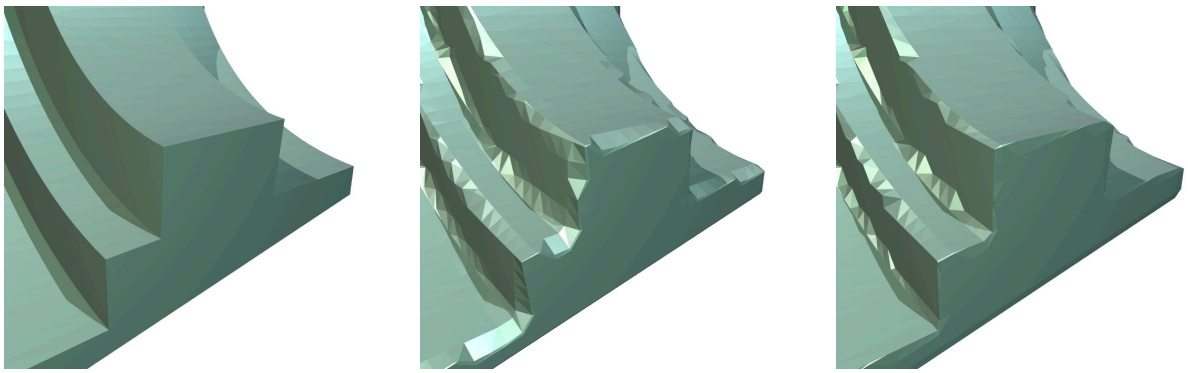


Figure 4.5: The center and right surfaces are generated by the MC algorithm applied to the uniformly sampled distance field of the object on the left. In the center, *scalar* distance values are stored for each grid point, while on the right three *directed* distances are stored to enable exact surface sampling. This reduces the alias errors to a small region around the feature.

edge. For instance, the intersection point $\mathbf{s}$ for the cell edge between $\mathbf{g}_{i,j,k}$ and $\mathbf{g}_{i+1,j,k}$ is computed by

$$\mathbf{s} \;=\; \left(1 - \frac{|\mathbf{d}_{i,j,k}[x]|}{h}\right) \mathbf{g}_{i,j,k} \;+\; \frac{|\mathbf{d}_{i,j,k}[x]|}{h}\, \mathbf{g}_{i+1,j,k} \;\;.$$

Although storing the directed distances $\mathbf{d}_{i,j,k}$ increases the memory consumption by a factor of three, it provides sample points lying *exactly* on the surface $\mathcal{S}$ to the MC algorithm (cf. Fig. 4.4, right). As demonstrated in Fig. 4.5, this significantly improves the quality of the extracted surface in the immediate vicinity of sharp features.

As another advantage, the computational effort for generating directed distance fields is actually lower than for scalar distance fields in the case of typical shape representations. Instead of searching for the minimum distance in *all* directions, the directed distances can be computed by ray casting along the grid axes, therefore all the acceleration techniques for fast ray-tracing can be exploited [AK89]. There are two different options for generating the directed distance field: If each cell edge is processed separately, the locality of the interrogation due to the small edge length can be exploited. On the other side, axis-aligned rays can be shot through whole rows of grid points, e.g., $\mathbf{g}_{0,j,k}, \ldots, \mathbf{g}_{n,j,k}$. All intersections along this ray are then used to store the directed distances at the corresponding grid points.

For *implicit surfaces* the ray intersection requires a univariate root finding scheme [KB89], which becomes particularly simple, as we only search along the x, y, or z axis. Due to the small edge length sufficiently good starting values guarantee a fast convergence of Newton-type iterations. Hence, the ray intersection is much simpler than a closest point search [BBB+97]. In the case of *polygonal meshes*, straightforward ray casting methods can be applied and accelerated by spatial data structures [Sam94]. If the input data is just a *point cloud*, like for volumetric surface generation (see Sect. 3.1.2), then each point has to be equipped with a properly oriented normal vector in order to define a signed distance field [ABK98, HDD+92]. Ray intersections can then be computed like presented in [SJ00, AA03].

4.2.2 Extended Marching Cubes

Even with the exact surface samples provided by the directed distance field, the alias artifacts at sharp features of the underlying surface $\mathcal{S}$ remain, since the standard MC algorithm computes these samples on a globally uniform grid. Locally adapting the sampling grid to the features of an object is critical, since this would sacrifice the simplicity and hence the efficiency of the basic algorithm. We therefore propose to detect sharp features within grid cells and to reconstruct them by placing *additional* sample points on the sharp feature in the cell's interior.

The feature detection and sampling is based on using the local gradient ∇F to extrapolate the behavior of the surface near features. Fig. 4.6 depicts the technique in two dimensions: Instead of directly connecting the sample points on the cell edges, the contour's normals are used to compute a linear local approximation (*tangent element*) for each intersection point. The intersection of the two tangents yields an additional sample point close to the sharp feature, such that including this additional sample into the contour approximation results in a much better reconstruction.

This simple technique works, because a reasonable geometric model — although not being differentiable near a sharp feature — can still be assumed to be *piecewise* differentiable. Hence, using point and normal information to generate tangent elements yields good approximations on both sides of the feature, such that the intersection of these approximations gives a good estimate of the actual feature position.

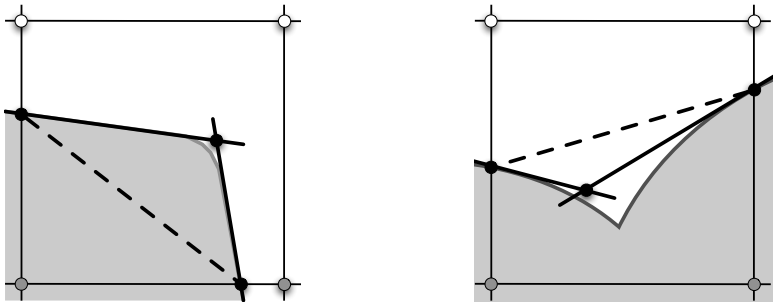


Figure 4.6: By using point and normal information on both sides of the sharp feature one can find a good estimate for the feature point at the intersection of the tangent elements. The dashed line is the result the standard Marching Cubes algorithm would produce.

It was pointed out in Sect. 4.1 that the approximation order of a piecewise linear mesh is $O(h^2)$, but that near sharp features the surface is not differentiable and hence the approximation order drops down to $O(h)$. Using the tangent element approximation, however, increases the local convergence rate in feature cells, since the quadratic order approximation is done on both sides of the feature separately.

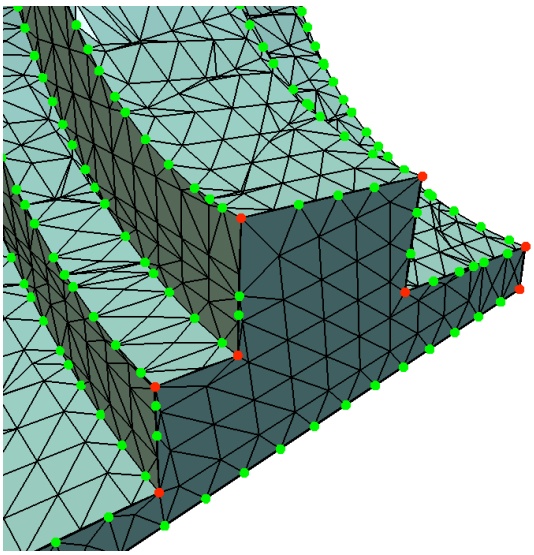


Figure 4.7: Feature samples are classified as either *edges* (green) or *corners* (red).

In our extended Marching Cubes algorithm we generalize this univariate feature point extrapolation technique to surfaces. In this case the situation is more complicated, since different types of features have to be handled in a different manner. These types are *feature edges*, where two smooth surface regions meet along a sharp feature line, and *feature corners*, where more than two smooth components meet or, equivalently, where more than two feature edges intersect (cf. Fig. 4.7). Just like the standard MC, the extended algorithm processes each cell separately. Each cell is checked whether a feature is present and if yes, which type of feature it is.

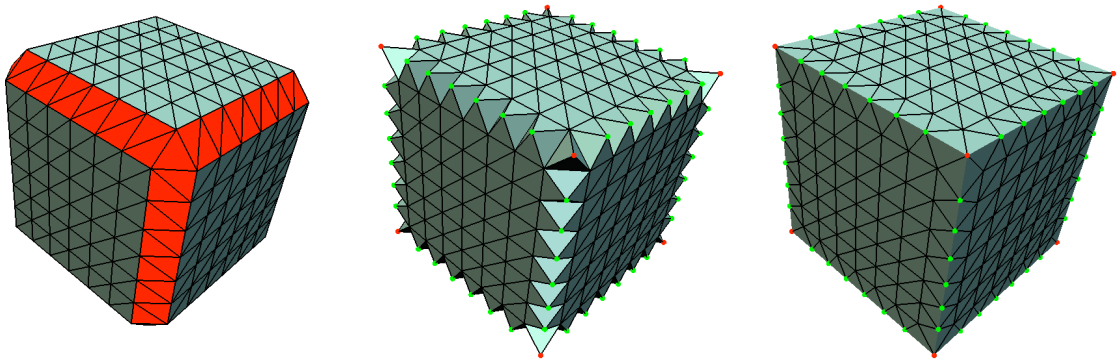


Figure 4.8: The feature sensitive sampling in the extended Marching Cubes algorithm works in three steps. First, the cells/patches that contain a feature are identified (*left*). Then one new sample is included per cell (*center*) and finally one round of edge flipping reconstructs the feature edges (*right*).

If the cell does not contain a sharp feature, a local patch is generated using the standard MC triangulation look-up table. However, if a feature is detected, the gradient information at the edge intersection points is used to define local tangent elements, the (pseudo-)intersection of which yields *one* new sample point lying close to the feature. This additional sample is included into the reconstructed surface using a triangle fan (instead of the standard triangulation) (cf. Fig. 4.8). Hence, the global uniform sampling grid can still be used to compute points on the surface $\mathcal{S}$, but additional sample points are included in those cells where sharp features are detected. By this, the extended MC combines the advantages of regular data structures with the flexibility of adaptive sampling.

Since the feature samples are inserted into the mesh as the center of a triangle fan without considering neighboring cells, the tessellation of the resulting mesh does not reflect the presence of feature line. However, this can easily be adjusted in a post-processing step by flipping all mesh edges where this modification connects two feature samples after the flip. The edge flipping does not produce any undesired side effects since the restriction to one feature sample per cell guarantees their sufficient separation. After the flipping, the edges connecting feature samples provide an explicit representation of the feature lines within the triangle mesh (cf. Fig. 4.8). Fig. 4.9 shows the results of the extended MC algorithm for the same dataset that has been used in Fig. 4.5.

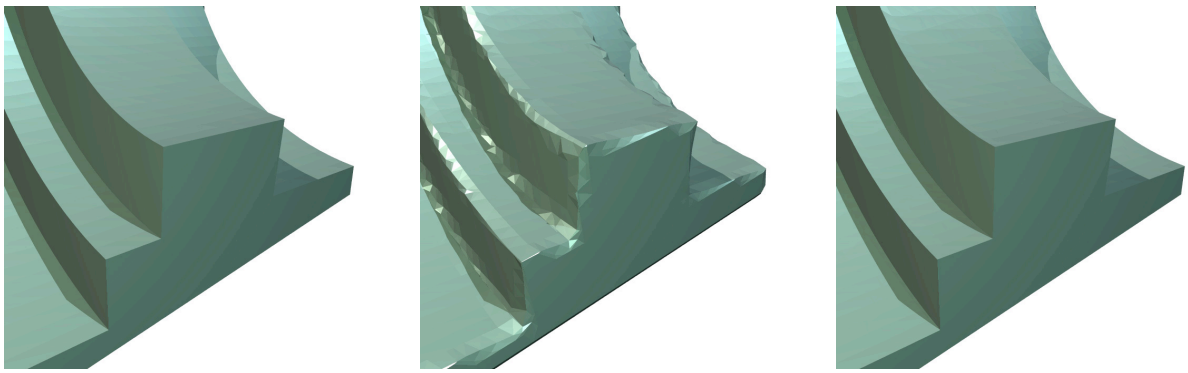


Figure 4.9: The original object on the left is converted into a volume representation with the same resolution as in Fig. 4.5. In the center and on the right we applied the extended Marching Cubes algorithm with feature sensitive sampling. The necessary gradient information is estimated from the discrete *scalar* distance field in the center and evaluated exactly from the *directed* distance field on the right. The result of the combination of the directed distance field with the extended Marching Cubes algorithm is indistinguishable from the original.

After this general description of the algorithm, the remaining technical questions are how to do the feature classification and how to compute the feature sample point. There are different ways to implement this functionality, where we designed our solution to not contain any unintuitive parameter and to find the optimal position for the feature sample.

Surface normals

The feature detection and feature sampling both need additional information about the surface $\mathcal{S}$. In addition to the position of the sample points, their normal vectors are required to construct the local tangent elements. The directed distances have been shown to provide exact sample points at the edge intersections. For the surface normal information the gradient of the distance field now also has to be evaluated exactly.

If an analytic *implicit function* F is known for the surface $\mathcal{S}$, then its gradient can be evaluated exactly at any location, using either symbolic or numerical partial derivatives. For the evaluation of the directed distance to a *polygon mesh*, intersection points with axis-aligned rays are computed. The gradient of the distance field at these points is simply the normal vector of the intersected triangle. When computing ray intersections

with a *point cloud*, the scattered points are replaced by tangent elements, hence the situation is the same as for polygonal meshes: the gradient is the normal vector of the point whose tangent disk was intersected. If only *discretized scalar distances* are available out of some pre-process, the original implicit function F cannot be accessed. In this case, the gradient has to be estimated from the tri-linear interpolation of the scalar distance values.

Feature detection

Let $\mathbf{s}_0, \ldots, \mathbf{s}_n$ be surface samples obtained by intersecting the edges of a grid cell with the surface $\mathcal{S}$ defined by $F(x, y, z) = 0$. If the constellation of the edge/surface intersection indicates (according to the standard MC table) the occurrence of more than one connected component, then we assume that the $\mathbf{s}_i$ are a subset of the edge intersections corresponding to the same component. In this case, the grouping of the $\mathbf{s}_i$ is done based on the MC table. In cells with several unconnected components the edge detection and feature sampling is applied for each component separately.

Let $\mathbf{n}_i$ be the unit surfaces normals of $\mathcal{S}$ at $\mathbf{s}_i$, i.e., the normalized gradients of F. The goal is to detect if the surface patch of $\mathcal{S}$ corresponding to the samples $\mathbf{s}_i$ contains a sharp feature. One simple but quite effective and intuitive heuristic to do this is to compute the opening angle of the normal cone spanned by the $\mathbf{n}_i$:

$$\theta \;:=\; \max_{i,j} \left\{ \angle \left(\mathbf{n}_i, \mathbf{n}_j \right) \right\} \;=\; \operatorname{acos} \left(\min_{i,j} \left\{ \mathbf{n}_i^T \, \mathbf{n}_j \right\} \right) \quad .$$

If θ is larger than some threshold angle θ_{sharp}, then the surface is expected to have a sharp feature. In this case, let $\mathbf{n}_0$ and $\mathbf{n}_1$ be the two normals which enclose the largest angle and let

$$\mathbf{n}^* \;:=\; \frac{\mathbf{n}_0 \times \mathbf{n}_1}{\| \mathbf{n}_0 \times \mathbf{n}_1 \|}$$

be the normal vector of the plane spanned by $\mathbf{n}_0$ and $\mathbf{n}_1$. In order to determine whether the detected feature is an *edge* or a *corner* point (cf. Fig. 4.7), the maximum deviation of the normals $\mathbf{n}_i$ from the plane spanned by $\mathbf{n}_0$ and $\mathbf{n}_1$ is computed by

$$\varphi \;:=\; \max_{i} \left\{ \frac{\pi}{2} - \operatorname{acos} \left(\left| \mathbf{n}_i^T \, \mathbf{n}^* \right| \right) \right\}$$

and tested against some threshold φ_{corner}.

These simple criteria proved to be quite effective in all applications reported in Sect. 4.2.3. The two parameters θ_{sharp} and φ_{corner} are very intuitive, since they can be considered as threshold angles that measure the sharpness of a feature. The threshold θ_{sharp} can be chosen quite small, say $\theta_{sharp} = 25°$, if the gradient data is not too noisy. For stability reasons in the subsequent calculations, however, it is advisable to choose the corner threshold φ_{corner} large enough, say $\varphi_{corner} = 45°$, in order to reduce the number of erroneous classifications. This is necessary to distinguish between sharp corners and curved feature lines. In all our experiments, the feature detection worked robustly without being too sensitive to the particular choice of the threshold parameters.

Feature sampling

If the current cell is classified either as a feature line ($\theta > \theta_{sharp},\ \varphi \leq \varphi_{corner}$) or as a feature corner ($\theta > \theta_{sharp},\ \varphi > \varphi_{corner}$), a sample point has to be found as close as possible to the feature. As explained above, a tangent element is generated from each sample $\mathbf{s}_i$ and its normal $\mathbf{n}_i$, and the feature sample is placed at the intersection of all tangent elements. This means that the new sample $\mathbf{p}$ tries to solve the linear system

$$\underbrace{\begin{pmatrix} \mathbf{n}_0^T \\ \vdots \\ \mathbf{n}_n^T \end{pmatrix}}_{=:N} \mathbf{p} = \begin{pmatrix} \mathbf{n}_0^T \mathbf{s}_0 \\ \vdots \\ \mathbf{n}_n^T \mathbf{s}_n \end{pmatrix} . \tag{4.1}$$

In general, this system is overdetermined, since there are usually more than three edge intersections in each cell. However, at feature edges it can also happen that this system is underdetermined, since at a perfect feature edge the tangent elements $(\mathbf{s}_i, \mathbf{n}_i)$ are all sampled from two different planes, and hence the matrix of normal vectors has (numerically) only rank two.

To avoid the handling of special cases, we solve the system (4.1) using the pseudo-inverse based on the singular value decomposition (SVD) of the matrix N [GL89b]. If the feature is classified as corner, then this is a very stable way to compute the optimal feature sample point in the least squares sense, i.e., to find the point $\mathbf{p}$ where the sum of squared deviations from all tangent elements takes on its minimum, similar to the quadric error metric used for mesh decimation [GH97].

If the feature is classified as an edge, one of the singular values is expected to vanish, since the feature line lies in both tangent planes. However, on real data this will hardly happen, since the gradients are affected by arithmetic noise, or the surface might be slightly curved. Since the angle-based feature classification decided for an edge, the smallest singular value of N is explicitly clamped to zero, thus enforcing the proper structure of the (now) rank deficient system (4.1) and thus stabilizing its solution.

The pseudo-inverse of the modified matrix $\tilde{N}$ yields the least norm solution of the underdetermined system, i.e., the point $\mathbf{p}$ on the feature line which is closest to the origin. For this point to lie in a reasonable configuration to the samples $\mathbf{s}_i$, a coordinate transform is applied to the samples *before* setting up the system (4.1), such that their barycenter is the origin.

The two steps required for the feature sampling are the angle-based feature classification and the pseudo-intersection of tangent planes by a SVD of the matrix N. It is tempting to try to read off the feature classification directly from N's singular values, as it has been done in [JLSW02]. However, it turned out that this is a very unreliable criterion, since the singular values not only depend on the angles between the normals, but also on their distribution. A feature edge can cause up to seven edge intersections belonging to the same surface component. A priori it is not known how many of those samples lie on either side of the feature. This makes the SVD classification quite unreliable, since the matrix $[\mathbf{n}_0, \mathbf{n}_0, \mathbf{n}_0, \mathbf{n}_0, \mathbf{n}_0, \mathbf{n}_1]$ has a very different singular value distribution than the matrix $[\mathbf{n}_0, \mathbf{n}_0, \mathbf{n}_0, \mathbf{n}_1, \mathbf{n}_1, \mathbf{n}_1]$.

There are also cases where our heuristic of extrapolating and intersecting tangent elements may fail (cf. Fig. 4.10). These situations occur mainly due to insufficient grid resolution and hence can be resolved by extracting the iso-surface on a higher refinement level of the grid. In order to avoid extremely stretched triangles in cases where the reconstructed sample point does not lie within its grid cell, the new feature point can also be projected back into its grid cell or one simply does not add a feature sample at all for such cell configurations.

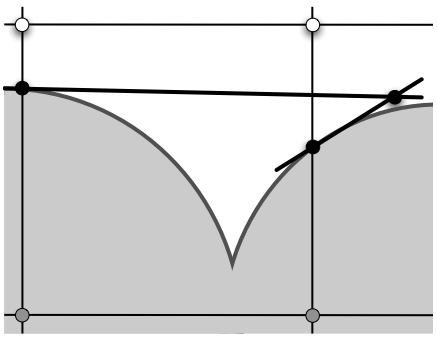


Figure 4.10: The tangent extrapolation may also fail in certain situations.

4.2.3 Results

The application examples presented in this section demonstrate the flexibility and effectiveness of our improved surface extraction scheme. In principle, the extended MC can always replace the original MC algorithm, since it has the same algorithmic structure and processes the same type of input data.

Obviously, the standard MC scheme will always outperform the extended version, since the extended MC has to do more involved computations for each cell. However, the feature sampling has to be done only in those cells where a feature configuration has been detected, and their number will increase only linearly with the grid refinement. For the technical examples shown in this section, the computational overhead is about 20%–40%. These examples contain about 10% more triangles compared to the standard MC's results, since the same refinement level was used for both algorithms. However, the necessary refinement level for a given accuracy is usually lower for the extended MC, since the feature sampling reduces the approximation error significantly.

The classical application area for volume representations is the design of solid objects by boolean operations (see Sect. 3.1.2). Feature sensitive sampling is very important in this context, since the sharp edges and corners indicate intersections of basic objects and carry significant design information (cf. Fig. 4.11).

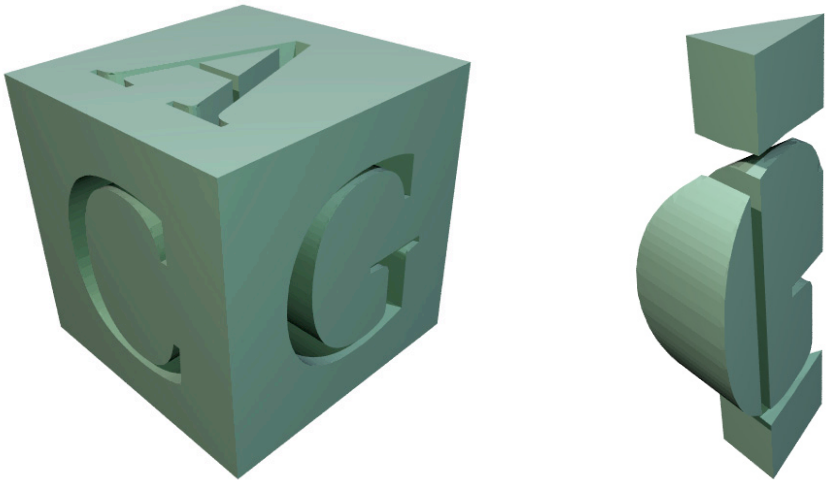


Figure 4.11: This figure shows a CSG example where the hollow letters are subtracted from a cube-shaped base object. The right image shows the pieces that are generated in the interior of the cube by the three cuts.

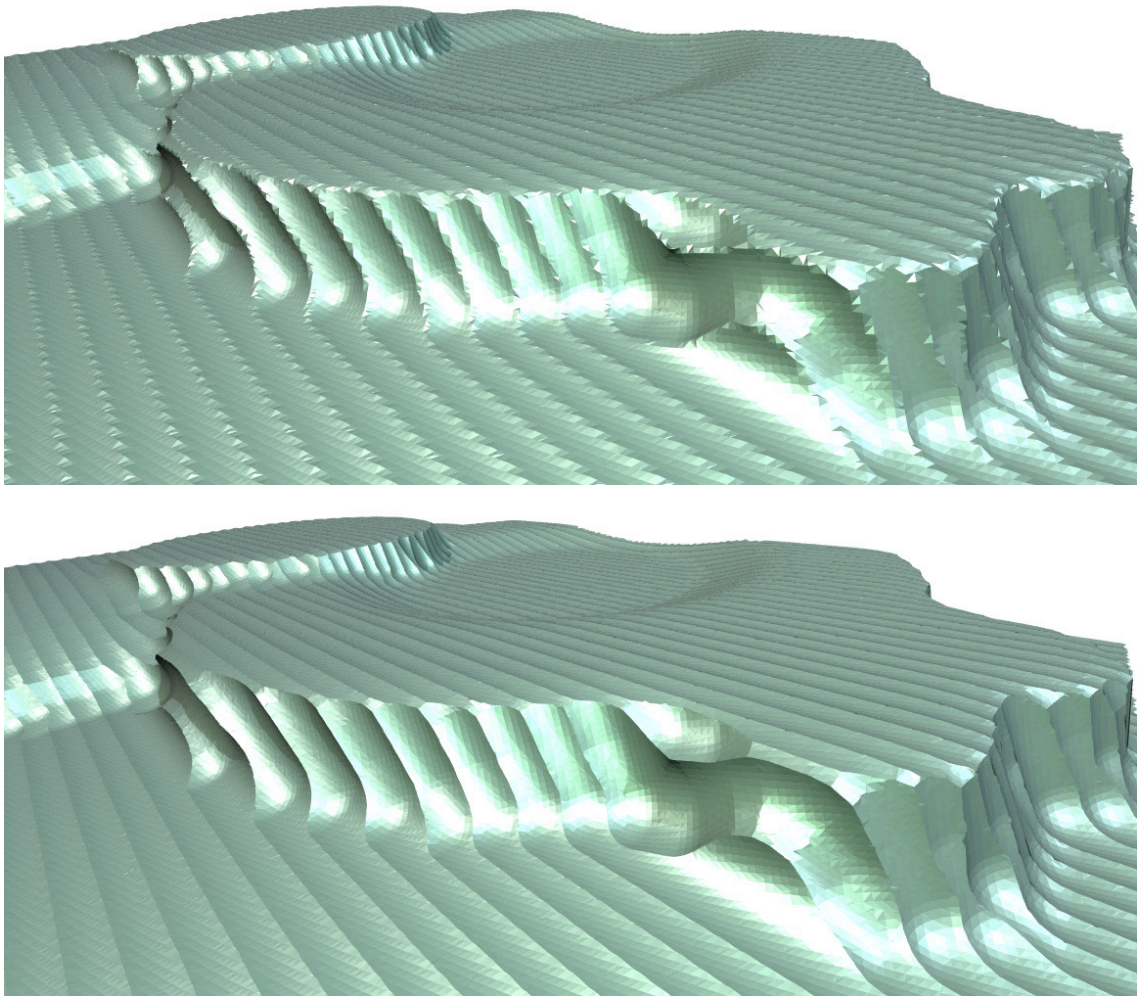


Figure 4.12: The result of a milling simulation computed by CSG techniques. The upper image shows the surface extracted by the standard MC algorithm, the lower image shows the extended MC surface. The sharp ridges are better visible due to the clearly reduced alias.

A very important practical application of this technique is the simulation of milling processes, where a milling tool is traced along a path and its envelope is to be generated. This application is very demanding for the solid modeling method, since the envelope surface usually intersects itself many times. The sharp ridges, that are characteristic for surfaces generated by a milling machine, carry crucial information, since they are used to rate the quality of the NC program (cf. Fig. 4.12).

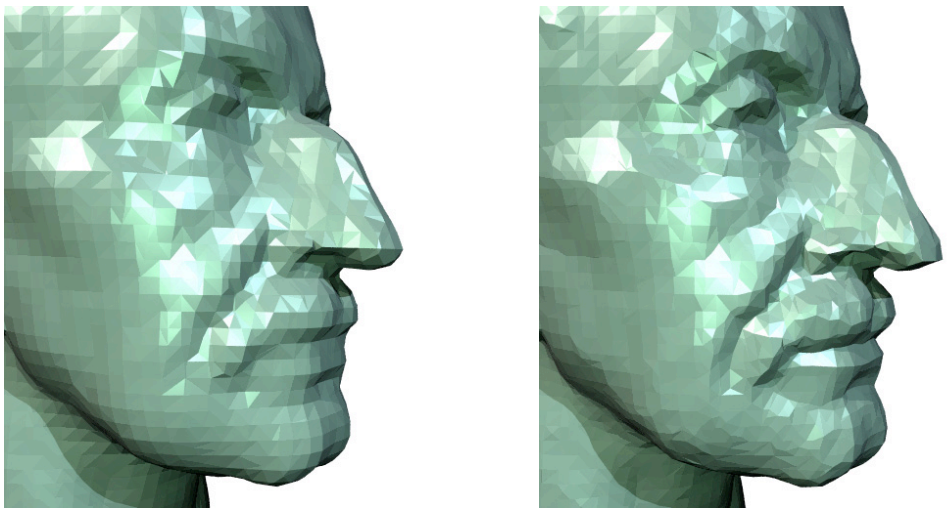


Figure 4.13: Triangle mesh reconstruction from a 3D scan of a bust of Max Planck consisting of 200k scattered points. In comparison to the result of the standard MC (*left*), the extended MC surface is less blurred and shows much more details around the mouth (*right*).

As described in Sect. 3.1.2, one well-established approach to surface reconstruction from unstructured point clouds is to estimate a signed distance function and then to apply the MC algorithm [HDD+92]. Since it is possible to compute *directed* distances and gradient information from point clouds with associated normal vectors, the extended MC can also be applied in this setting. Fig. 4.13 shows a surface reconstructed from a dataset of 200k scanned points.

Since scattered point datasets often come from a 3D scanning device, they are usually disturbed by noise, which affects the quality of the resulting 3D models and has to be removed by mesh smoothing techniques (see Sect. 3.2.1). For meshes generated by the extended MC algorithm, some mesh vertices and mesh edges are additionally tagged as feature points and feature edges, respectively. This information can be exploited in order to further improve the surface quality in a smoothing step by applying a univariate smoothing scheme to the feature lines and a bivariate smoothing scheme to the non-feature areas (cf. Fig. 4.14).

Polygonal meshes which are generated at some intermediate stage of an industrial CAD process often have a bad quality. To make this data accessible to applications other than mere display, their tessellations have to be optimized by remeshing algo-

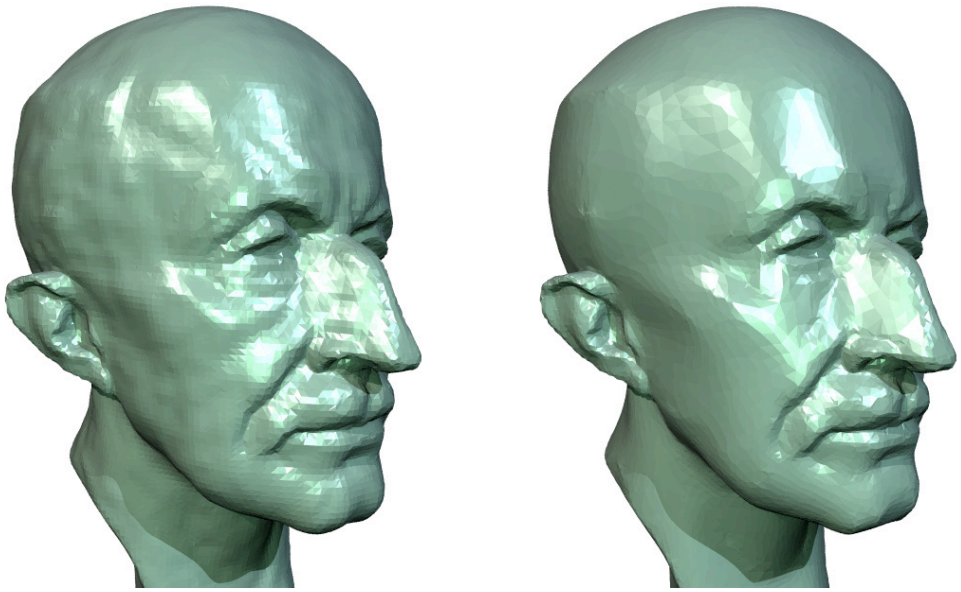


Figure 4.14: The left image shows a high resolution result of the extended MC applied to the point cloud of Fig. 4.13. Low-pass filtering the mesh while taking the feature information into account leads to the result on the right. All sharp features are well preserved, while in the non-feature areas noise is effectively removed.

rithms (see Sect. 3.2.3). As an alternative, one can also convert a CAD model into a volumetric representation, such that applying the extended MC algorithm to this volume gives a re-meshed version of the original with a more uniform vertex distribution. Similar to the feature sensitive smoothing, the additional feature information in the output can be exploited to control the behavior of a mesh decimation post-process (see Sect. 3.2.2). This results in effectively decimated meshes that preserve the relevant feature information (cf. Fig. 4.15).

The extended MC together with the directed distance field presents an important enhancement of standard MC like techniques. By exploiting gradient information of the distance field this method is able to reliably detect and classify sharp feature regions on the surface and to accurately sample these features in order to reduce alias artifacts.

However, a major problem of MC techniques is the high complexity of the resulting meshes due to the uniform over-sampling. Instead of decimating these meshes in a post-processing step, for instance by piping them through an out-of-core mesh decimation process [WK04], it would be better to directly extract an adaptively sampled mesh.

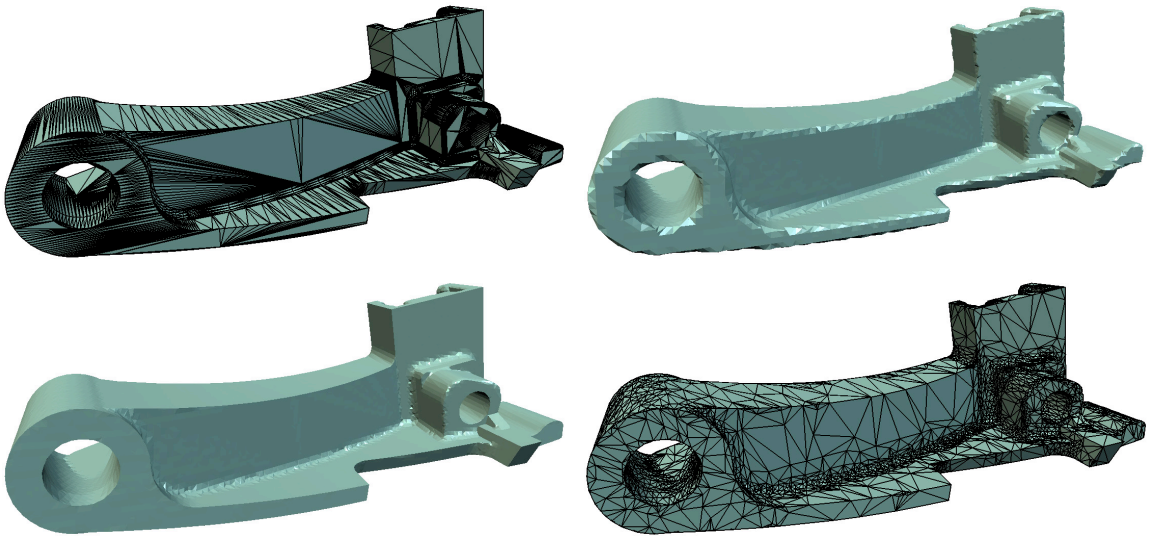


Figure 4.15: Remeshing of a polygonal mesh. The top left mesh has been generated from a CAD model and shows a very bad distribution of triangles. The distance field of the model was sampled on a 129^3 grid and the surface was reconstructed using the standard MC algorithm, leading to severe alias artifacts (*top right*). The lower left image shows that in the result of the extended MC all sharp features are reconstructed correctly. The lower right image finally shows the result of a feature-preserving mesh decimation algorithm (error tolerance 1%).

Combining the general idea of the extended MC with the *surface nets* of Frisken et al. [FPRJ00], Ju et al. [JLSW02] proposed the *dual contouring* approach, which works on an adaptive octree and allows for the extraction of adaptive meshes. However, the approach as described in their paper leads to non-manifold situations for those cell configurations where MC would build more than one patch per cell. Hence, these cases have to be specially handled to guarantee a clean two-manifold output.

An example implementation of the extended MC based on our OpenMesh data structure [BSM05] can be downloaded from [Bot05]. Since we did not optimize the extended MC code for computation speed, there should be room for improvements in the different phases of our current implementation. As the algorithm processes each cell individually (like the standard MC), a parallelization should be straightforward.

4.3 Feature-Sensitive Resampling of Blend Regions

The last section pointed out that in order to avoid alias artifacts in the representation of sharp features, samples have to be placed exactly on these features and the triangulation has to be adjusted, such that feature samples are connected along their corresponding feature lines. In this section we generalize this principle to *rounded* features, that are not sharp, but highly curved in an anisotropic manner, like the typical cylindrical blend regions in technical datasets, which we describe in Sect. 4.3.1.

The input models for numerical simulations, like computational fluid dynamics (CFD), are typically derived by a reverse engineering process. These meshes may initially contain several millions of sample points in order to not loose relevant geometric detail at the early stages of the reconstruction process. As a consequence, they have to be decimated down to a complexity the target application is able to handle. Most geometry-based decimation schemes are *greedy* algorithms that only consider the *local* shape to decide about which vertex to remove in the next step (see Sect. 3.2.2). As a consequence, there is no direct control of the distribution and alignment of the mesh vertices on the surface. Although the sampling density may locally adapt to the surface curvature, there is no possibility to achieve global effects, like an alignment of the triangulation to highly curved features in the geometry.

The meshes resulting from such a decimation process often turn out to be inappropriate for sophisticated downstream applications like numerical simulations, because the (weighted) random distribution of vertices results in severe alias errors, which can lead to erroneous simulation results and become visible as shading artifacts (cf. Fig. 4.16). Those alias errors are again caused by the fact that, although the decimated triangle mesh stays point-wise within some tolerance to the original data, the normal vectors can deviate significantly. The resulting normal noise becomes particularly evident in the vicinity of feature lines, where the two principal curvatures differ strongly.

The only way to solve this geometric alias problem in surface reconstruction is to choose the "right" sampling pattern, i.e., to globally adjust the distribution and alignment of mesh vertices, such that the normal vectors of the triangles are a sufficient approximation of the normal vectors of the original surface.

In an early approach we enhanced our virtual range scanning technique [KB00] (as described in Sect. 3.1.1) by a feature-aligned sampling pattern [BRK00]. Instead of re-

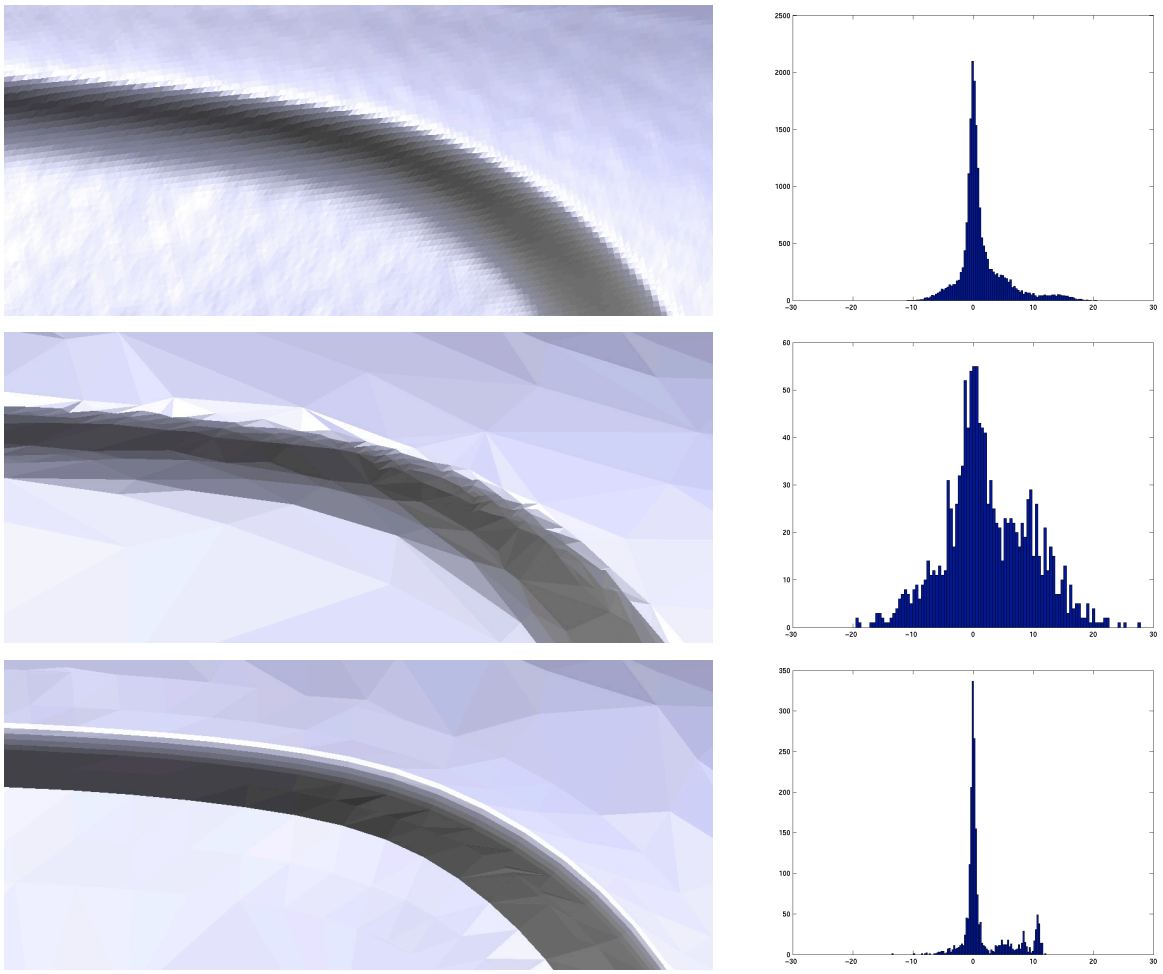


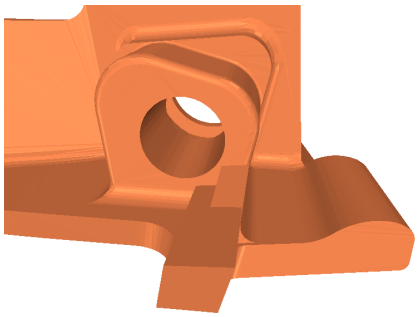
Figure 4.16: Geometric aliasing such as normal noise becomes clearly visible under specular shading. The top image shows an original 3D-scan of a feature region. Although the point positions have been sampled at high precision, the normals of the resulting mesh deviate strongly from the normals of the original surface. Applying mesh decimation (*center*) improves the situation slightly, since the triangles are stretched along the feature, but the normal noise is still disturbing. In the *bottom* image we apply our alias-reducing feature resampling. Although the mesh resolution has not changed, the quality has improved due to effective normal noise elimination. The right column shows the histograms of the respective normal jumps.

projecting each pixel after rendering the surface into the z-buffer (leading to a regular grid of 3D points), only a subset of the pixels is used, that is specified by a 2D sampling pattern in the image space of the z-buffer. The special feature-aligned pattern is constructed from the projection of lines of minimum and maximum curvature directions into the image plane. Although following the right principles, this approach is obviously limited by working on an intermediate two-dimensional image representation.

In [BK01a] we then proposed an object-space solution to this sampling problem that is described in this section. After deriving a sampling pattern for blend regions that minimizes normal noise (Sect. 4.3.2), we present an interactive technique to re-sample feature regions of *given* triangle meshes, such that the alias artifacts are strongly reduced (Sect. 4.3.3, cf. Fig. 4.16). We advocate for an integration of this technique into a semi-automatic setup, since we consider the problem of *detecting* feature regions to be independent from the actual (re-)sampling problem. For industrial surface design applications, manual feature detection is acceptable and even preferred by most designers.

4.3.1 Feature Regions

In the boundary representation of geometric (solid) models three types of surface regions can be distinguished: *geometric primitives* (e.g., parts of spheres, cylinders, or tori), *freeform surfaces* (smooth surface patches of general shape), and *blends*, that are used to join the other surface parts in order to obtain a consistent representation of a closed solid, like shown in the figure to the side.



A blend surface can be thought of as being generated either by rolling a ball of varying or constant radius over the gap between two surface segments or by sweeping a profile curve along the two opposite boundaries (cf. Fig. 4.17). In the extreme case, a blend can degenerate to a sharp feature curve, at which two surface segments meet with discontinuous tangent planes. They correspond to rolling ball blends with balls of vanishing radius.

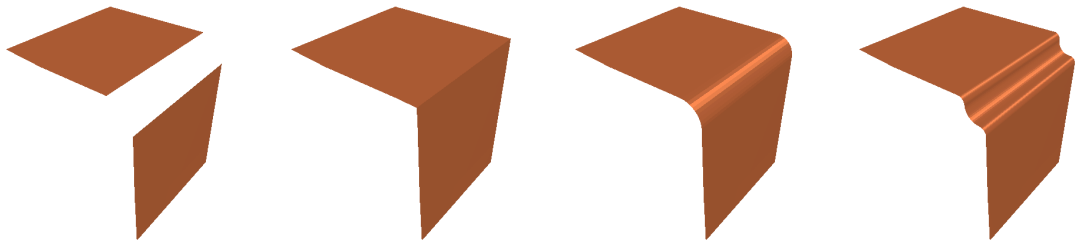


Figure 4.17: Feature regions on a complex surface usually emerge from blending two separate patches along their corresponding boundaries. The two patches on the *left* can be joined by computing their intersection, leading to a sharp feature line (*center left*). Alternatively we can roll a ball of prescribed radius (*center right*) or sweep a more complicated profile along a center curve (*right*).

When sampling a surface, the sampling density has to adapt to the local curvature distribution in order to capture all (and only) relevant geometric details. Obviously, in highly curved regions, where the principal curvatures κ_1 and κ_2 are both large, the sampling has to be denser than in regions where both principal curvatures are small. If the magnitude of the curvatures does not differ too much, then an isotropic sampling pattern is sufficient. However, since *two* principal curvatures characterize the local curvature, an optimal sampling pattern has different densities in the corresponding principal directions. Hence, in *feature regions* — characterized by $\kappa_1 \ll \kappa_2$ — an anisotropic sampling pattern that is aligned to the principal directions has to be used.

In terms of the above classification into *primitives*, *freeform patches*, and *blends*, the feature regions are usually the blend areas where, e.g., a sphere of radius $1/\kappa_2$ rolls along a curve with curvature $\kappa \approx \kappa_1 \ll \kappa_2$.

4.3.2 Sampling pattern for blend regions

Fig. 4.16 clearly depicts that the weighted random distribution of surface samples, as it typically emerges from mesh decimation, does not yield satisfactory results in feature regions due to *normal noise*. In this section, we present a simple sampling pattern for feature regions that reduces normal noise, i.e., the variation of normal jumps, to a minimum.

It is easy to see that random sampling generally leads to significant normal noise. Consider, e.g., the simple example of a cylinder as shown in Fig. 4.2. Placing the samples randomly on the surface causes an uncontrollable tilt of the triangle normals away from the original surface normals, which are all orthogonal to the cylinder axis. The generic configuration of a triangle's normal vector being orthogonal to the cylinder axis only occurs if the triangle's embedding plane intersects the cylinder in two parallel lines. Since the triangle is spanned by three surface samples, these samples also have to lie on those two lines. This implies that a triangle is free of normal noise if and only if one of its edges is parallel to the cylinder axis.

Now consider a sampling pattern where all samples lie on a set of lines which are parallel to the cylinder axis and distributed equally around the cylinder. Each strip between two of those lines can be tessellated by a planar triangulation. As a consequence, the normal jumps between triangles are either zero (within the same strip) or a constant angle that only depends on the number of strips. Hence, the normal noise is minimal. The two different values of normal jumps correspond to the two principal curvatures of the cylinder.

This idea is now generalized in order to derive a sampling pattern for surfaces that are part of an envelope generated by moving a sphere of constant radius along a space curve. In Sect. 4.3.4 the same sampling pattern will be applied to even more general profile sweep surfaces to empirically demonstrate that this still results in superior quality meshes compared to random sampling, although zero normal noise can no longer be guaranteed in this generalized setting.

The envelope of a moving sphere can be defined alternatively by a *center curve* $\mathbf{c}(t)$ along which a planar circle profile is moved. The normal of the circle's embedding plane at a time step t_0 is defined by the tangent $\mathbf{c}'(t_0)$ of the center curve. The sweep surface itself is the collection of all profiles at different time steps $t \in [a, b]$. We assume that the minimum curvature radius of the center curve $\mathbf{c}$ is larger than the radius of the circle profile to avoid the discussion of surface degeneracies.

According to the above definition we can distinguish two natural directions on such a sweep surface: one *along* the center line and one *around* the center line. We can use these directions for a natural parameterization $\mathbf{S}(t, u)$ with t varying along and u around the center curve. In this parameterization, the iso-curves with constant parameter t_0 are circles around the center $\mathbf{c}(t_0)$. Iso-curves with constant parameter u_0 are the *trajectories*

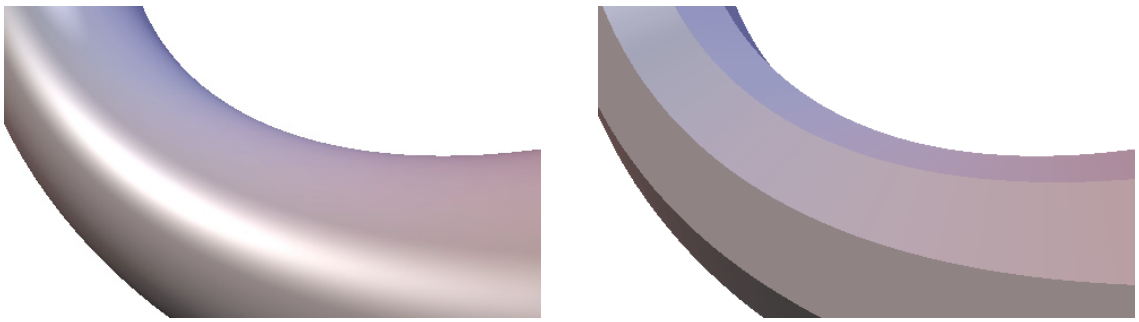


Figure 4.18: By discretizing the sweep profile the the original envelope surface is approximated by a collection of n ruled surface patches $\mathbf{R}_i$. Since the circular profile is replaced by a regular n-gon, which moves orthogonally to its embedding plane, all normal jumps between neighboring strips are constantly equal to $2\pi/n$.

along which a specific point on the circle profile moves. Obviously, the trajectories are offset-curves to the center curve and consequently the iso-curves with respect to the parameter t and u intersect perpendicularly. In fact, it can be shown that the iso-curves are the principal curvature lines of the sweep surface [dC76]. Another important property of the trajectories, which will be used later on, is that they have constant Euclidean distance as well as constant geodesic distance to each other.

The sampling pattern for the sweep surface has to discretize the parameter domain in t and u direction. We start by discretizing the moving profile itself, i.e., we approximate the circle $\mathbf{S}(0, u)$ by a closed polygon $(\mathbf{p}_0, \ldots, \mathbf{p}_{n-1})$ with $\mathbf{p}_i = \mathbf{S}(0, i/n)$. Sweeping this closed polygon instead of the circle results in a surface that consists of n ruled surface patches

$$\mathbf{R}_i(t, u) \;=\; (1-u)\,\mathbf{S}\left(t, \frac{i}{n}\right) \;+\; u\,\mathbf{S}\left(t, \frac{i+1}{n}\right) \quad .$$

The trajectories along which the points $\mathbf{p}_i$ move are perpendicular to the profiles. Hence, if the swept polygon is discretized by a regular n-gon, then the normal jump between neighboring ruled patches $\mathbf{R}_i$ is exactly $2\,\pi/n$ everywhere (cf. Fig. 4.18). As a consequence, we have a constant normal jump, i.e., zero variation.

Next, the ruled surfaces $\mathbf{R}_i$ have to be discretized in t direction along the feature, which corresponds to a triangulation of these patches, i.e., of the strips enclosed between the trajectories. Since the trajectories are lines of minimal curvature, triangles within the same strip have a small normal jump only. However, the constant normal jump property of the n-gon sweep should be preserved as good as possible. If a center curve segment $\mathbf{c}([t_0, t_1])$ is approximated by a straight line, the resulting patch $\mathbf{R}_i([t_0, t_1], [0, 1])$ is approximated by a bilinear surface patch. Since the approximation error of the line segment to the center curve decreases like $O(|t_1 - t_0|^2)$, even for general center curves the surface patch $\mathbf{R}_i([t_0, t_1], [0, 1])$ can locally be approximated by a bilinear patch, such that the quadrilateral spanned by the four points $\mathbf{R}_i(t_0, 0)$, $\mathbf{R}_i(t_0, 1)$, $\mathbf{R}_i(t_1, 0)$, and $\mathbf{R}_i(t_1, 1)$ is almost planar.

Consequently, no matter how the quadrilateral is split into two triangles, no significant normal jump will be generated between these triangles. In addition, the normal jumps between neighboring quadrilaterals generated from $\mathbf{R}_i([t_0, t_1], [0, 1])$ and $\mathbf{R}_{i+1}([t_0, t_1], [0, 1])$ is approximately $2\pi/n$, since the two quads are generated by two incident edges $(\mathbf{p}_i, \mathbf{p}_{i+1}, \mathbf{p}_{i+2})$ of the swept n-gon at time steps t_0 and t_1, respectively. Hence, it turns out that the regular triangulation for each strip, which uses the sample pairs $\mathbf{R}_i(t_j, 0)$ and $\mathbf{R}_i(t_j, 1)$ for any sequence of parameter values t_j, does not introduce significant normal noise. Moreover, it can be shown that any modification of this triangulation only increases the normal noise.

Consider, e.g., the four samples $\mathbf{a} = \mathbf{S}(t_0, i/n)$, $\mathbf{b} = \mathbf{S}(t_1, i/n)$, $\mathbf{c} = \mathbf{S}(t_2, (i-1)/n)$, and $\mathbf{d} = \mathbf{S}(t_3, (i+1)/n)$, which define two triangles $T_1 = (\mathbf{a}, \mathbf{b}, \mathbf{c})$ and $T_2 = (\mathbf{d}, \mathbf{b}, \mathbf{a})$ in neighboring strips. If the trajectories $\mathbf{S}(t, (i-1)/n)$, $\mathbf{S}(t, i/n)$, and $\mathbf{S}(t, (i+1)/n)$ are no straight lines, then the normal angle between T_1 and T_2 can differ significantly from the optimal value $2\pi/n$ when we choose the parameter values t_2 and t_3 from the interior of the interval $[t_0, t_1]$ (cf. Fig. 4.19). This local deviation of the normal jump from the average $2\pi/n$ propagates around the swept surface, because the sum of the normal jumps along a planar contour around the center line has to be equal to 2π.

In conclusion of this section we find that the key to an anti-aliased sampling pattern on spherical sweeps is to arrange the surface samples $\mathbf{p}_{i,j} = \mathbf{S}(t_i, u_j)$ such that the the points $(\mathbf{p}_{i,j})_j$ build a regular n-gon around the center curve and the samples $(\mathbf{p}_{i,j})_i$ lie on trajectories.

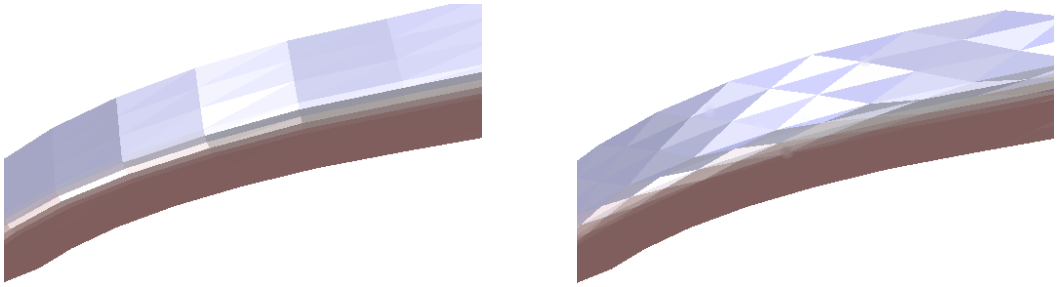


Figure 4.19: In both images, the feature region is reconstructed by placing the samples along the trajectories. One the left, the samples are "synchronized" in the orthogonal direction (i.e., along the contours) as well, leading a noise free appearance. One the right, the phase on every other trajectory is shifted, thus provoking normal noise.

4.3.3 Interactive Feature Resampling

In the last section we showed that a rolling-ball blend should be triangulated based on a sampling pattern that is aligned to the principal curvature directions (trajectories and circle profiles) in order to minimize normal noise. However, sampling an existing feature with unknown center curve $\mathbf{c}$ from a *given* triangle mesh is a different situation: unless we are dealing with a sharp feature (i.e., its radius equals zero) the center curve does not lie on the given surface, neither do we know the radius of the profile that was swept along it. Instead, the surface data only provides the resulting blend. In order to reverse-engineer the feature and to resample it in an anti-aliased manner, the sampling pattern has to be generated without explicitly knowing the parameterization $\mathbf{S}(t, u)$.

The sampling grid is generated using a *fishbone*-like structure: first a *spine curve* $\mathbf{T}_0$ is constructed to be approximately aligned *along* the feature, and then *rib curves* $\mathbf{U}_i$ are traced to branch off perpendicularly from it (and hence are aligned *around* the feature). In terms of the last section, the spine $\mathbf{T}_0$ corresponds to a trajectory on the sweeping profile, while the ribs $\mathbf{U}_i$ represent the contour at different time steps. Each rib $\mathbf{U}_i$ is then sampled uniformly with respect to an arc-length parameterization, resulting in a set of sample points $\mathbf{p}_{i,j}$. Connecting the jth sample from each rib therefore results in a curve $\mathbf{T}_j$ with constant geodesic distance from the spine. This implies that the curve $\mathbf{T}_j$ is another trajectory, and it follows that a regular triangulation of the samples $\mathbf{p}_{i,j}$ has the properties derived in the previous section.

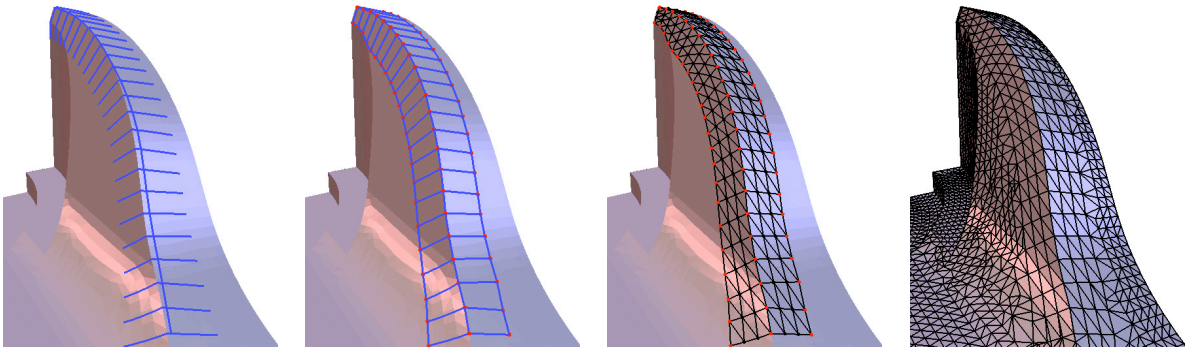


Figure 4.20: This sequence of images gives a rough overview of the resampling procedure. The spine is generated by interpolating user selected surface points and a set of ribs is created by intersecting the surface with a set of planes being orthogonal to the spine (*left*). Univariate feature snapping is used to re-position the spine exactly on a sharp feature line and two additional trajectories are created (*center left*). A uniform sampling of the rib curves allows for a regular triangulation of the feature region. The alignment of the sampling grid to the ribs in one direction and to the trajectories in the orthogonal direction guarantees an anti-aliased reconstruction (*center right*). Finally, the resampled mesh is stitched into the target mesh (*right*).

Consequently, the resampled patch is an anti-aliased approximation of the feature region, that can be inserted into the target mesh by some mesh-stitching method [KB00]. Notice that the target mesh does not need to be the same mesh as the one that was sampled from. Instead, the target mesh can be enhanced by stitching in alias-reduced patches that were sampled from the best available geometry, i.e., from the original non-decimated range scans. In the following we explain the basic steps of the resampling procedure in more detail (cf. Fig. 4.20).

The initial spine is constructed interactively: The designer sketches the feature by picking a few positions on an estimated trajectory and the spine curve is then automatically generated by smoothly interpolating or approximating these points. Since the spine curve is not required to lie exactly on the surface, this procedure allows to generate smooth spine curves even if the underlying surface data is noisy. The only soft requirement for the resulting spine curve is that it should be an approximate offset of a trajectory.

To generate the rib curves, the spine $\mathbf{T}_0$ is sampled either uniformly or with a curvature-dependent step width. For each of the sample points $\mathbf{t}_i$ a rib curve $\mathbf{U}_i$ is created by intersecting the given surface with the plane positioned at $\mathbf{t}_i$ and orthogonal to the spine's tangent (cf. Fig. 4.20, left). This special rib generation is the reason why the spine does not have to lie exactly on the surface. As a consequence, each rib is a *planar* polygon that *exactly* lies on the surface. If the given surface is a polygonal mesh, the plane intersection can be implemented by a simple local marching scheme, such that the computation costs do not depend on the overall complexity of the mesh.

To create a new trajectory the user selects one or more interpolation points on different ribs. Starting from such a point, the new trajectory is constructed by marching from rib to rib. The corresponding points on the neighboring ribs can be identified according to several different criteria:

- The point having the same geodesic distance to an already existing trajectory can be chosen, thereby mimicking the offset curve property of trajectories.
- One can proceed in orthogonal direction to the current rib, to mimic the principal direction property of trajectories.
- The local curvature maximum can be chosen in order to trace a sharp feature line. This is a convenient method to snap the spine to a feature line if the initial spine did not fit (cf. Fig. 4.20, center left). Notice that the snapping only requires *univariate* feature detection on each rib curve.
- A given set of points can simply be interpolated by a smooth curve, which provides full manual control to the designer.

In case of the spine snapping the ribs might have to be recomputed by intersecting the surface with a new set of planes being orthogonal to the new spine (cf. Fig. 4.20, center left). This technique is particularly useful in practice, since it allows the designer to precisely select sharp feature lines on a CAD model without having to pick points exactly on the feature by himself.

If the original surface geometry is noisy, both the rib curves and the new trajectories will be noisy as well. However, due to the enhanced structural information, *univariate* smoothing can now be applied to these curves, which is far more efficient than bivariate surface smoothing. By using additional trajectories as C^0 or C^1 boundary constraints for

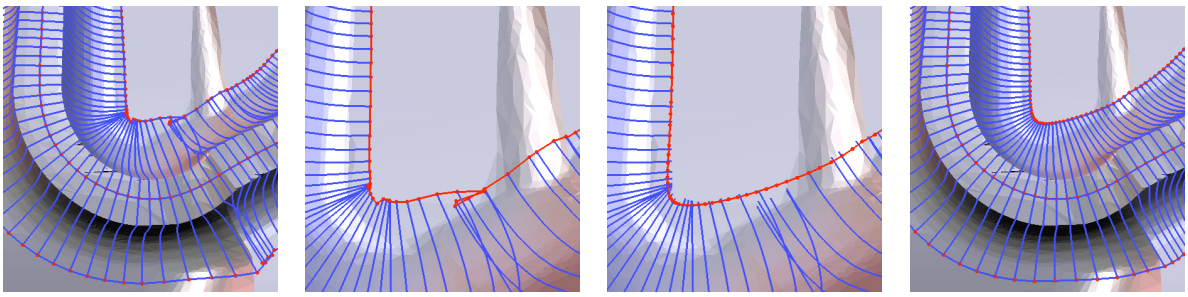


Figure 4.21: At strongly curved features the ribs may intersect each other. Giving up the requirement that the ribs have to be orthogonal to the trajectories, one can still find a decent triangulation with reduced normal noise.

the smoothing of rib curves, sharp features can also be preserved during the smoothing process.

In the beginning we assumed that the curvature κ_1 of the center curve is small. If this is not the case, then strongly curved features may lead to overlapping rib curves, as shown in Fig. 4.21. To still be able to generate an anti-aliased triangulation in this case, the requirement of ribs being orthogonal to trajectories has to be weakened. For a given fishbone (= spine samples $\mathbf{t}_i$ plus ribs $\mathbf{U}_i$) two additional trajectories with constant geodesic distance from the spine are generated by connecting the left and right endpoints of all ribs, $\{\mathbf{l}_i\}$ and $\{\mathbf{r}_i\}$, respectively. In the presence of overlapping ribs, these trajectories have kinks and loops (cf. Fig. 4.21, center left). Applying a simple low-pass filtering operator to the outer two trajectories straightens out these degeneracies (cf. Fig. 4.21, center right). After this filtering, each three points $(\mathbf{l}_i, \mathbf{t}_i, \mathbf{r}_i)$, which are associated with the ith rib, define a tilted plane. Intersecting the given surface with those new planes results in new ribs which no longer overlap (cf. Fig. 4.21, right).

The last step of the resampling procedure is to compute equidistant samples on each rib with respect to the arc-length parameterization. Those samples have the property that they are (trivially) aligned to the ribs (= contours), but are also aligned to the trajectories, since the jth sample on each ribs has the same geodesic distance to any other trajectory (cf. Fig. 4.20, center right). Hence the sampling grid matches the requirements from the previous section and therefore results in an anti-aliased surface reconstruction. The resampling procedure is concluded by stitching the new patch into the original mesh (cf. Fig. 4.20, right).

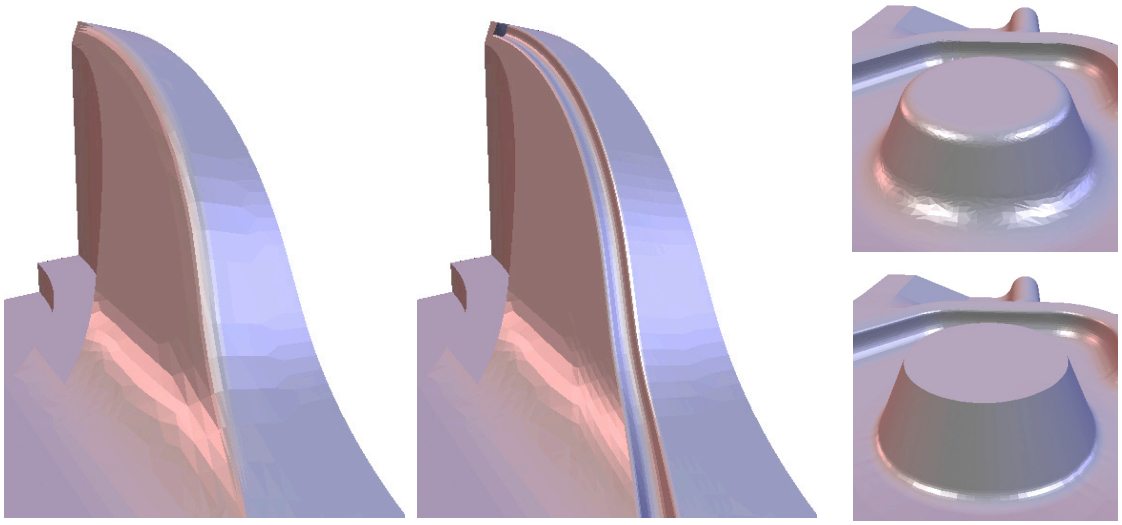


Figure 4.22: Feature modeling using the fishbone-metaphor. The sharp feature of Fig. 4.20 has been rounded to a prescribed radius by univariate modeling operations on each rib (*left*). More complex modeling operations are possible if we exchange the circular profile for a more complicated one (*center*). The rounded feature in the *right* image is sharpened by setting the blend radius for the rib profiles to zero for the upper ring and to a small but non-zero value for the lower ring.

4.3.4 Feature Modeling

The fishbone metaphor does not only provide a resampling of the geometry that strongly reduces normal noise, the additional structural information can also be used for higher level feature modeling. In general, any two neighboring trajectories can be used to "cut out" the parts of the ribs enclosed by them, such that these parts can be replaced by a general 2D Hermite interpolant for each rib.

Changing the characteristics of a feature is a very frequent operation in product design. For example, in CFD simulations it is often necessary to vary the sharpness of a feature (i.e., the radius of a rolling ball blend) to verify the impact on the overall aero dynamics. *Rounding* and *sharpening* are the operations which increase or decrease the blend radius along a feature. On a fishbone-wise resampled feature region such modeling operations are very easy to implement, due to the perfect alignment of the sampling grid to trajectories and ribs. Since the ribs are known to be planar, this reduces the

feature modeling to 2D operations on each rib curve separately. Even more complicated profile sweeps can be constructed by simply replacing the moving profile by a generic 2D Hermite interpolant (cf. Fig. 4.22).

We applied the surface anti-aliasing technique in the context of CFD simulation for conceptual car design to a detailed mesh model of the BMW Z8 car. The normal noise contained in the models after the triangulation and decimation phase could effectively be removed. Some results are shown in Fig. 4.23.

4.3.5 Discussion

The last sections pointed out that in order to reduce normal noise, i.e., the variation of normal jumps, the sampling pattern has to be aligned to the principal curvature directions of the underlying geometry. Our interactive fishbone re-sampling metaphor does exactly this by placing sample points along different trajectories in a phase-synchronized manner. The amount of normal noise in a triangle mesh can be regarded as a discrete fairness measure, similar to the surfaces of minimal curvature variation used in CAGD to generate high quality surfaces.

We restricted our resampling to feature and blend regions of technical datasets. However, such an anisotropic remeshing can also be applied to whole models in order to generate a tessellation that reflects the structure of the underlying geometry better than random sampling obtained by range scanning [ACSD+03, MK04]. In this context, lines of minimum and maximum curvature are traced over the surface in a first step. The intersection points of these lines yield the new surface samples, that are then connected and decomposed into quads and triangles. A piecewise linear approximation with elements that are aligned to the principal curvature directions and scaled according to the principal curvatures provides an (asymptotically) optimal L^2 approximation of height fields, and also of surface normal fields. This fact was very recently exploited for the high quality variational shape approximation of Cohen-Steiner et al. [CSAD04].

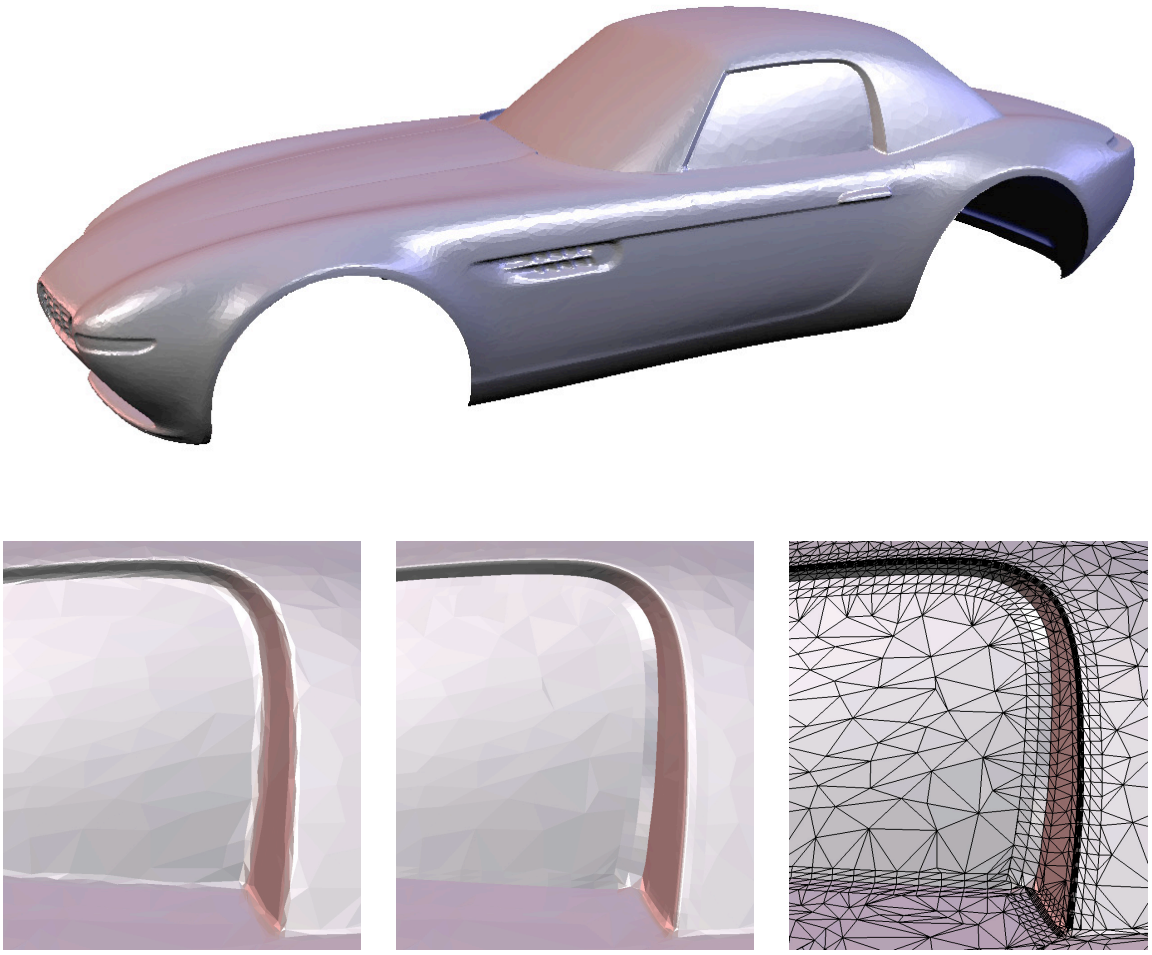


Figure 4.23: We applied the resampling and surface anti-aliasing technique to a detailed BMW Z8 model, which is supposed to be used for CFD simulation. The normal noise in the vicinity of the feature regions of the decimated model (*bottom left*) can cause numerical instabilities and erroneous turbulences. In the remeshed feature regions around the driver's window the normal noise is almost completely removed. The other parts of the model have not been resampled. A closeup of this model was shown in Fig. 4.16.

5 Multiresolution Techniques

In this chapter we introduce the multiresolution techniques needed for a general multiresolution modeling framework. In this context, the term multiresolution refers to modeling or editing a given surface at different scales or different levels of detail. This means, that on the one hand, the designer has to be able to edit the surface at the finest possible resolution, which is defined by the distribution and density of the mesh vertices. On the other hand, a multiresolution modeling framework also has to provide global deformations, like bending or stretching of the model.

The latter case is the more challenging one, since a global shape deformation of a given model additionally has to preserve all the fine surface details in an intuitive and physically plausible manner (cf. Fig. 5.1). Hence, after generating faithful approximations of technical datasets using the feature-sensitive surface processing methods described in the first part of the thesis, we now have to find surface deformation techniques that are also feature-aware. Because of this feature preservation the multiresolution modeling

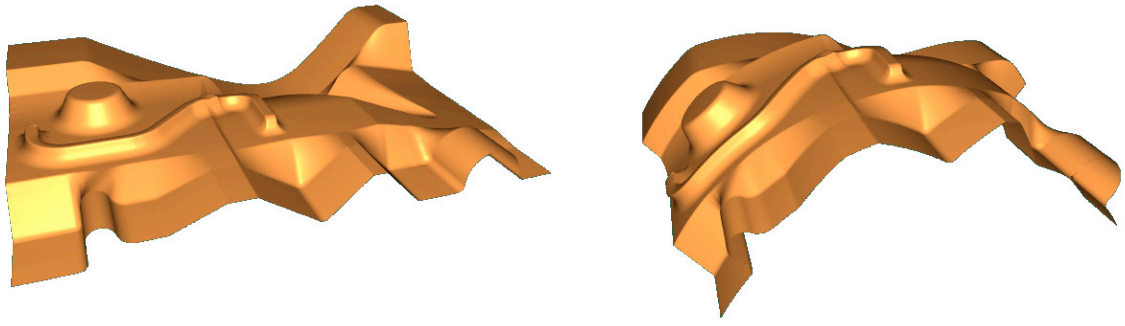


Figure 5.1: Multiresolution modeling has to provide global surface deformations with a physically plausible and intuitive preservation of all the fine details and sharp features of the surface.

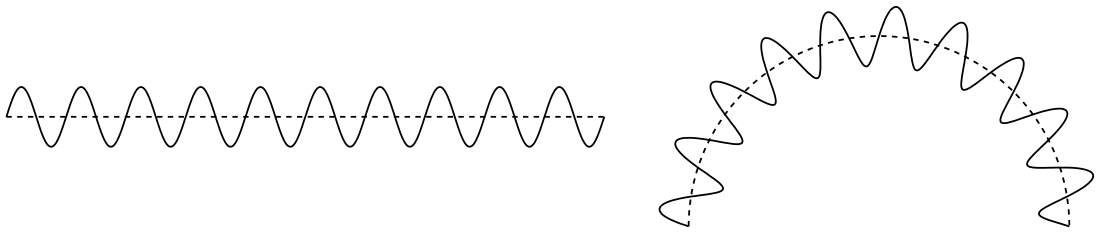


Figure 5.2: A multiresolution deformation of a sine wave. A frequency decomposition yields the dashed line as its low frequency component (*left*). Bending this line and adding the higher frequencies back onto it results in the desired global shape deformation (*right*).

paradigm has proven to be the most effective shape deformation concept when it comes to the modeling of highly complex surface, like those derived from a 3D scanning process.

5.1 Multiresolution Modeling Framework

In order to enable a deformation of the global shape of an object while at the same time preserving its fine details, a frequency decomposition of the object is performed. In the section on surface smoothing (Sect. 3.2.1) we have seen that signal processing techniques, like low-pass filtering and the notion of frequencies, can be generalized to (signals on) surfaces. In this setting the fine surface details correspond to the high frequencies of the surface signal and the global shape is represented by its low frequency components. But in contrast to surface smoothing we now want to explicitly modify the low frequencies and preserve the high frequency details, resulting in the desired multiresolution deformation. Fig. 5.2 shows a simple 2D example of this concept.

The complete multiresolution editing process is depicted in Fig. 5.3. In a first step a low-frequency representation of the given surface $\mathcal{S}$ is computed by removing the high frequencies, yielding a smooth base surface $\mathcal{B}$. The geometric details $\mathcal{D} = \mathcal{S} \ominus \mathcal{B}$, i.e., the fine surface features that have been removed, represent the high frequencies of $\mathcal{S}$ and are stored as detail information. By this we are able to reconstruct the original surface $\mathcal{S}$ by adding the geometric details back onto the base surface: $\mathcal{S} = \mathcal{B} \oplus \mathcal{D}$. The special operators $\ominus$ and $\oplus$ are called the *decomposition* and the *reconstruction* operator of the multiresolution framework, respectively.

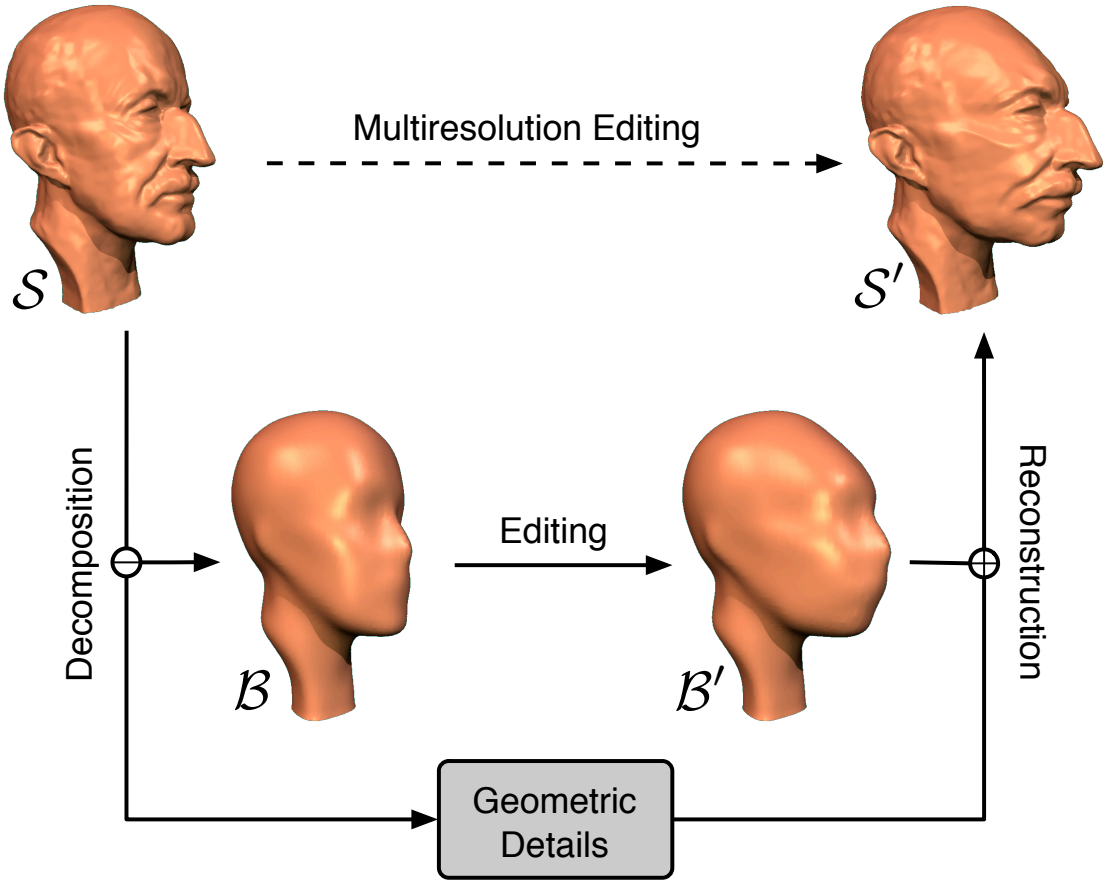


Figure 5.3: A general multiresolution editing framework consists of thee main operators: the *decomposition* operator, that separates the low and high frequencies, the *editing* operator, that deforms the low frequency component, and the *reconstruction* operator, that adds the details back onto the modified base surface. Since the lower part of this scheme is hidden in the multiresolution kernel, only the multiresolution edit in the top row is visible to the designer.

This multiresolution surface representation is now enhanced by an *editing* operator, that is used to deform the smooth base surface $\mathcal{B}$ into a modified version $\mathcal{B}'$. Adding the geometric details onto the deformed base surface then results in a multiresolution editing $\mathcal{S} \mapsto \mathcal{S}' = \mathcal{B}' \oplus \mathcal{D}$.

Notice that in general more than one decomposition step is used to generate a hierarchy of meshes $\mathcal{S} = \mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_k = \mathcal{B}$ with decreasing geometric complexity. In this case the frequencies that are lost from one level $\mathcal{S}_i$ to the next smoother one $\mathcal{S}_{i+1}$ are stored as geometric details $\mathcal{D}_{i+1} = \mathcal{S}_i \ominus \mathcal{S}_{i+1}$, such that after deforming the base surface to $\mathcal{B}'$, the modified original surface can be reconstructed by $\mathcal{S}' = \mathcal{B}' \bigoplus_{i=0}^{k-1} D_{k-i}$. Since the generalization to several hierarchy levels is straightforward, we restrict our explanations to the simpler case of a two-band decomposition, as shown in Fig. 5.3.

A complete multiresolution editing framework therefore has to provide the three basic operators shown in Fig. 5.3: The decomposition operator (*detail analysis*), the freeform editing operator (*shape deformation*), and the reconstruction operator (*detail synthesis*). In the remainder of this chapter we focus on multiresolution surfaces, i.e., the detail representation and the corresponding decomposition and reconstruction operators. After introducing the standard detail representation *displacement vectors*, we present our new representation for multiresolution models called *displacement volumes*, that is based on volume elements enclosed between the original surface $\mathcal{S}$ and the base surface $\mathcal{B}$. This representation provides a more natural behavior of the surface details and effectively avoids local self-intersections of the modified surface $\mathcal{S}'$. The freeform editing operator is described afterwards in Chap. 6.

5.2 Base Surface Generation

The first step for the generation of a multiresolution representation is the computation of the smooth base surface $\mathcal{B}$ from the original surface $\mathcal{S}$, or, in general, the creation of a hierarchy of meshes $\mathcal{S} = \mathcal{S}_0, \ldots, \mathcal{S}_k = \mathcal{B}$ with decreasing geometric level of detail.

Subdivison surfaces are generated by a repeated refinement of a coarse control mesh, which in the limit converges to a (piecewise) smooth surface (see Sect. 2.1.2). They are an inherent multiresolution surface representation, because each refinement level adds more geometric detail to the subdivision surface. The major drawback is that subdivision

techniques are restricted to surfaces of semi-regular connectivity, which either limits the range of input models, or requires a complete remeshing of the surface. Moreover, the layout of the initial control mesh is extremely important for later modeling operations, as we will see in Chap. 6. As a consequence, a sufficient initial patch layout usually has to be manually specified by an experienced designer.

Because of these restrictions, our goal is to work on arbitrary triangle meshes, as they allow for higher flexibility in all surface processing phases. We have seen in Sect. 3.2.1 that the amount of geometric detail in a mesh corresponds to its frequency spectrum, such that meshes corresponding to hierarchy levels of less details can be derived by successive low-pass filtering [KCVS98, KVS99, GSS99].

In [KVS99] Kobbelt et al. proposed to decimate the mesh $\mathcal{S}$ by collapsing all edges being shorter than a prescribed threshold ε_i, and then to re-insert the removed vertices again, but this time at positions determined by a discrete fairing scheme [Kob97], leading to a mesh $\mathcal{S}_i$ of the same connectivity as $\mathcal{S}$, but with less geometric detail.

If standard low-pass filtering is to be used for the hierarchy creation, then the speed of typical iterative smoothing schemes is locally a function of the vertex density. In the case of an extremely irregular input mesh $\mathcal{S}$, this might lead to an unwanted slower filtering of densely sampled regions. One possible solution is to isotropically resample the mesh before the low-pass filtering, leading to a more regular and more robust smoothing process, as we will show in Chap. 7.

Since usually only a part of the surface is to be deformed, we can also restrict to remove the high frequencies in this active region only. As we will shown in Chap. 6, the low frequency base surface can then be computed by a constrained minimization of a curvature energy functional, resulting in a surface that interpolates the fixed boundaries of the active region in a C^2 manner and is maximally smooth in its interior.

No matter how the smooth base surface $\mathcal{B}$ is computed, the difference in geometric detail $\mathcal{S} \ominus \mathcal{B}$ has to be encoded as local detail information $\mathcal{D}$. In the following we discuss two different kinds of representations for the detail information, that are displacement vectors and displacement volumes.

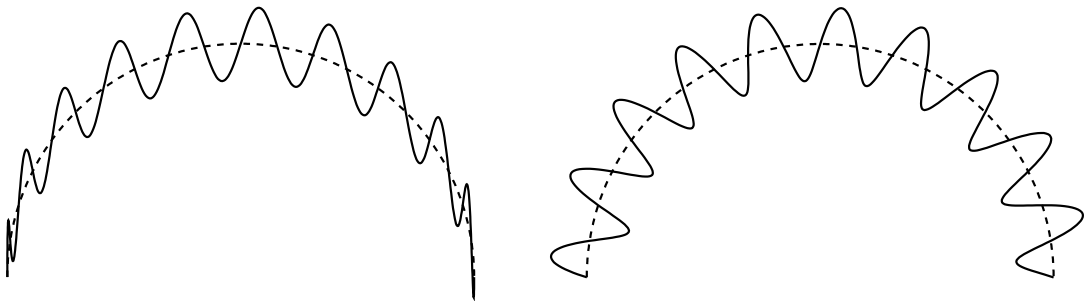


Figure 5.4: Representing the displacements w.r.t. the global coordinate system does not lead to the desired result (*left*). The geometrically intuitive solution is achieved by storing the detail w.r.t. local frames which rotate according to the local tangent plane's rotation of $\mathcal{B}$ (*right*).

5.3 Displacement Vectors

The standard representation for multiresolution details is a displacement of the base surface $\mathcal{B}$, i.e., the detail information is a vector valued displacement function $\mathbf{d} : \mathcal{B} \rightarrow \mathbb{R}^3$, that associates a displacement vector $\mathbf{d}(\mathbf{b})$ with each point $\mathbf{b}$ on the base surface. Hence, the detailed surface $\mathcal{S}$ can be reconstructed from the base surface $\mathcal{B}$ by $\mathcal{S} = \{\mathbf{b} + \mathbf{d}(\mathbf{b}) \mid \mathbf{b} \in \mathcal{B}\}$.

Although the realization of this representation seems to be straightforward, special attention has to be paid to the representation of the displacement field. Expressing the displacements w.r.t. a global coordinate system does not lead to the expected results (cf. Fig. 5.4, left). When the base surface $\mathcal{B}$ is deformed to $\mathcal{B}'$, the displacements have to be rotated according to the local rotations of the base surface's tangent plane in order to guarantee a plausible detail reconstruction $\mathcal{S}'$. Hence, the displacements have to be expressed in so-called *local frames* [FB88, FB95], that consist of the surface normal and two perpendicular tangent vectors (cf. Fig. 5.4, right).

The typical discretization of the displacement field is to restrict the base mesh $\mathcal{B}$ to have the same connectivity as the detailed surface $\mathcal{S}$, such that each vertex $v_i \in \mathcal{V}_\mathcal{S}$ with position $\mathbf{p}_i := \mathbf{p}_\mathcal{S}(v_i) \in \mathcal{S}$ has an associated base point $\mathbf{b}_i := \mathbf{b}(v_i) := \mathbf{p}_\mathcal{B}(v_i) \in \mathcal{B}$. The corresponding displacement vector $\mathbf{d}_i := \mathbf{d}(v_i) := (\mathbf{p}_i - \mathbf{b}_i)$ is then stored in the local frame of $\mathcal{B}$ at the point $\mathbf{b}_i$. This detail representation was used by Zorin et al. [ZSS97]

in the context of multiresolution subdivision surfaces and by Guskov et al. [GSS99] for multiresolution hierarchies on arbitrary irregular meshes.

The problem of these general displacement vectors is the tangential component of the local frame encoding. Using the normal vector as one axis of the local frame is geometrically very intuitive, but the two vectors spanning the tangent plane are not uniquely determined. Therefore some heuristic has to be found to fix the possible rotation of the local frame around the normal vector, e.g., by choosing the first tangent vector to be the projection of the first incident edge into the tangent plane. However, depending on the transformation $\mathcal{B} \mapsto \mathcal{B}'$ the same heuristic applied to $\mathcal{B}'$ might not lead to intuitive results. The second problem is that the tangential component does not carry geometric information, but can be regarded as a parameterization artifact instead. Elongated displacement vectors due to large tangential components may therefore cause stability problems or lead to non-intuitive reconstructions, as discussed in [KVS99].

Suppressing the tangential component and enforcing the displacement vectors to be parallel to the normal of the base surface leads to so-called *normal displacements*. As the displacements are in general not parallel to the surface normal, generating normal displacements has to involve some kind of resampling. Shooting rays in normal direction from each base vertex $\mathbf{b}_i \in \mathcal{B}$ and deriving new vertex positions $\mathbf{p}_i \in \mathcal{S}$ at their intersections with the detailed surface leads to a resampling of the latter [GVSS00, LMH00]. Because $\mathcal{S}$ is a detailed surface with high frequency features, such a resampling is likely to introduce alias artifacts.

Therefore Kobbelt et al. [KVS99] go the other direction: For each vertex position $\mathbf{p}_i \in \mathcal{S}$ they find a base point $\mathbf{b}_i \in \mathcal{B}$ (now *not* necessarily a vertex of $\mathcal{B}$!), such that the displacements are normal to $\mathcal{B}$, i.e., $\mathbf{p}_i = \mathbf{b}_i + h_i \cdot \mathbf{n}(\mathbf{b}'_i)$. This avoids a resampling of $\mathcal{S}$ and therefore allows for the preservation of all of its sharp features. First, a globally continuous linear normal field on $\mathcal{B}$ is generated by barycentric interpolation of its vertex normals, similar to Phong shading for high quality rendering. This guarantees that for each point $\mathbf{p}_i \in \mathcal{S}$ a base point $\mathbf{b}_i$ for a normal displacement exists, and it can be found efficiently by a local search based on Newton iterations.

These normal displacements are then encoded by their length h_i and by the base point $\mathbf{b}_i$, which is represented parametrically by a triangle index and its barycentric coordinates within that triangle. After modifying the base surface, the new base point $\mathbf{b}'_i \in \mathcal{B}'$ is determined by this parametric information, and the corresponding point

$\mathbf{p}'_i \in \mathcal{S}'$ is reconstructed by $\mathbf{p}'_i = \mathbf{b}'_i + h_i \cdot \mathbf{n}(\mathbf{b}'_i)$. Once the normal displacements have been generated in the decomposition phase, the required per-frame reconstruction operator is extremely efficient, since it basically involves computing the linear normal field on the deformed base surface, that is needed anyway for rendering the modified surface.

5.4 Displacement Volumes

A major problem of the well established displacement vectors is that they are handled individually, i.e., they are not coupled in any way. While this approach usually leads to sufficient detail reconstructions for translational or rotational modifications, it results in an unnatural change of volume as soon as the base surface is bent. Consider the prisms that are spanned by the original triangles of $\mathcal{S}$ over the base surface $\mathcal{B}$: Bending the base surface changes their opening angles and thereby alters the prism volumes. Since the volume enclosed between the base surface and the detailed surface is intuitively supposed to stay constant, this behavior does not fully satisfy the plausibility requirements of detail preservation (cf. Fig. 5.5).

A more severe problem of uncoupled displacement vectors is that they do not provide any mechanism to prevent self-intersections. This problem comes in two different forms: *global* and *local* self-intersections. The global form is a variant of the general collision detection problem, that occurs when the deforming surface touches itself. Obviously, the detection and handling of global self-intersections has to be taken care of by the freeform editing operator, since the semantics of a global collision depends on the design intended and go beyond the task of *plausible* detail preservation.

The *local* self-intersection phenomenon, however, has a different nature. As shown in Fig. 5.6, these difficulties typically arise when the base surface is deformed in a concave manner. Where a local self-intersection occurs, the surface is not colliding with itself, but it is folding over itself. Expressed in terms of the prisms spanned by the displacement vectors, local self-intersections occur when one or more of these prisms degenerate. Notice that for global self-intersections usually no individual prisms degenerate. Local self-intersections are primarily due to the detail vector displacement and consequently have to be fixed by the reconstruction operator.

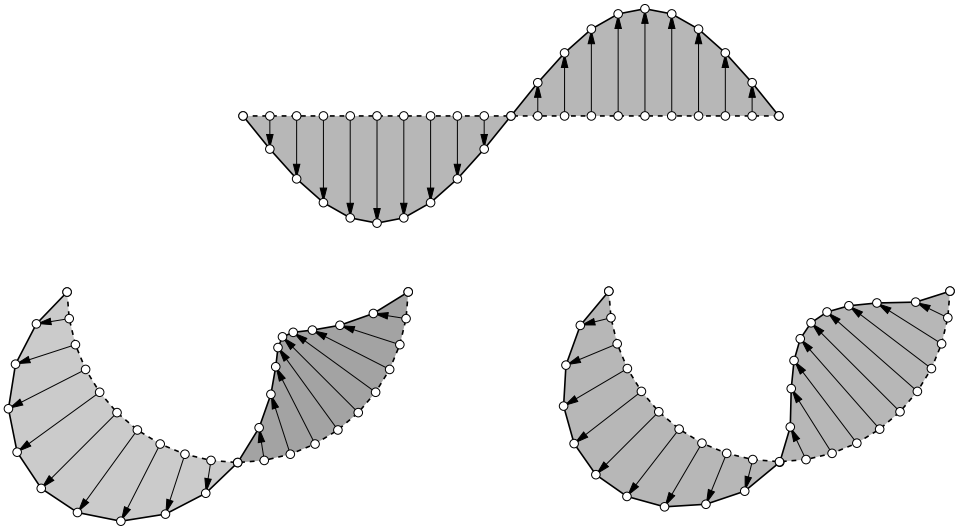


Figure 5.5: A multiresolution deformation of a sine wave is done by bending its base line (*dashed*) and reconstructing the corresponding detailed surface (*solid*). Since displacement *vectors* are handled individually, the resulting surface shows an unnatural change of the volume enclosed between base and detailed surface (*bottom left*). Displacement *volumes* provide a natural coupling of the displacements, that results in a more natural behavior and prevents local self-intersections (*bottom right*).

An obvious way to address this issue is to shift the displacement vectors in tangential direction. However, this has to be done in a way that adheres to the plausibility of the detail preservation. Adjusting the displacements individually or propagating the tangential shift by some diffusion operator applied to the displacement vectors will most probably distort the geometric detail in a non-plausible way.

Both problems, the unnatural change of volume and local self-intersections, are addressed by our new detail encoding scheme, that is based on displacement *volumes* instead of displacement *vectors* [BK03d]. Each triangle of the original detailed mesh $\mathcal{S}$ spans a prism over the base surface $\mathcal{B}$, and the volumes of these prisms are used as detail coefficients. For a modified base surface $\mathcal{B}'$ the reconstruction operator then has to find a new detailed mesh $\mathcal{S}'$ that has the same connectivity as $\mathcal{S}$ and spans the same prism volumes.

This notion of volume preservation provides a physical interpretation for the plausibility of the detail preservation: The detail is supposed to mimic the behavior of elastic

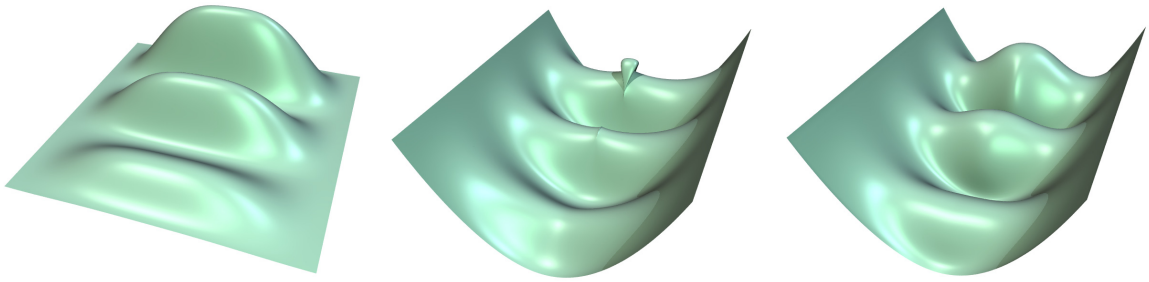


Figure 5.6: Multiresolution editing enables global deformations with intuitive detail preservation. However, detail reconstruction based on displacement *vectors* may lead to a non-plausible change of volume and even to local self-intersections for concave modifications (*center*). Displacement *volumes* instead reconstruct a more natural, non-intersecting surface (*right*).

but incompressible materials. The multiresolution model will deform like a soft but incompressible layer attached to a rigid skeleton (cf. Fig. 5.5). Displacement volumes can also effectively avoid local self-intersections (where the surface of a prism would interpenetrate itself), since prisms can shear, i.e., their top triangles can move tangentially, without changing their volume (cf. Fig. 5.6).

A similar idea was used by Lee et al. [LTW95], who presented a multi-layer tissue model for facial animation, that is based on a mass-spring system extended by an approximate volume preservation. Volume differences due to facial deformations are compensated for by adjusting the prisms' heights, i.e., by shifting vertices in normal direction only. However, this means that exactly the tangential movements required to prevent local self-intersections are suppressed.

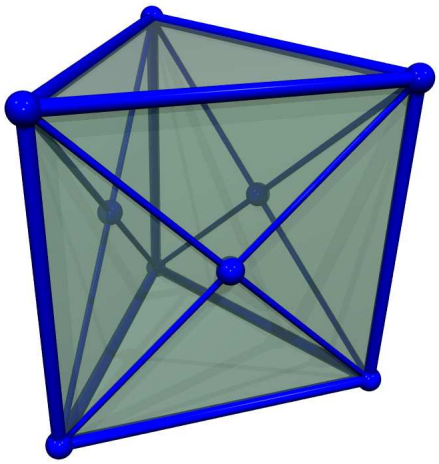
Another method to preserve the geometric detail in a physically plausible manner would be using general finite element methods (FEM). However, this requires substantial computations, that are usually simplified by linearizing elasticities, which, in turn, leads to problems for large rotational modifications. Moreover, the standard FEM formulation is not well-defined when simulating strict incompressibility. As a consequence, the so called *mixed formulation* has to be used for exact volume preservation, leading to more complicated systems of equations [Bat95].

In order to achieve a high quality surface reconstruction in CAD-like editing applications, the system of volume constraints should be solved exactly, instead of computing an approximation only. Moreover, for a generally applicable approach we cannot rely on the quality and regularity of the meshes to be processed, but instead have to be able to robustly handle complex irregular meshes and allow for arbitrarily large scale modifications. As shown in the next sections, these requirements are satisfied by displacement volumes. Given an original surface $\mathcal{S}$ and its low frequency base surface $\mathcal{B}$, the first step is the detail analysis, which amounts to compute and store the prism volumes (Sect. 5.4.1). The corresponding reconstruction operator and its implementation is then described in Sect. 5.4.2.

5.4.1 Volumetric Detail Representation

In the initial state, i.e., before the modification, we can assume that the geometric difference between $\mathcal{S}$ and $\mathcal{B}$ can be represented by normal displacements, as described in Sect. 5.3. Hence, it is possible to find a base point $\mathbf{b}_i \in \mathcal{B}$ for each vertex position $\mathbf{p}_i \in \mathcal{S}$, such that $\mathbf{d}_i = (\mathbf{p}_i - \mathbf{b}_i)$ is perpendicular to $\mathcal{B}$.

Each triangle $(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k)$ of $\mathcal{S}$ together with the corresponding base points $(\mathbf{b}_i, \mathbf{b}_j, \mathbf{b}_k)$ on $\mathcal{B}$ spans a triangular prism. Notice that the quadrilateral faces on the sides of these prisms are non-planar in general and are therefore consistently split into four triangles each by inserting the centroid of their vertices. This guarantees that neighboring prisms use the same tessellation of their common quadrilateral face, and hence no artificial asymmetries are introduced.



After this splitting, the boundary surface of each prism is given by 14 triangles, and the volume of the prism can easily be calculated as a sum of oriented tetrahedra volumes (spanned by the origin and the respective triangle) by

$$V \;=\; \frac{1}{6}\sum_{i=1}^{14} \det\left[\mathbf{u}_i, \mathbf{v}_i, \mathbf{w}_i\right] \quad ,$$

where the $\mathbf{u}_i$, $\mathbf{v}_i$, and $\mathbf{w}_i$ are the coordinate vectors of the corners of the respective triangles. The geometric detail information which is lost when switching from $\mathcal{S}$ to $\mathcal{B}$ is stored as the initial volumes V_j^* for each triangle $f_j \in \mathcal{F}_\mathcal{S}$ of $\mathcal{S}$.

If during the reconstruction phase the volume of a prism is to be changed by shifting one of its vertices $\mathbf{p}_j \in \mathcal{S}$, it is most effective to move it into the direction of the volume gradient, since this yields the maximum volume change for the minimum vertex displacement. Let $\mathbf{u}_0, \ldots, \mathbf{u}_4$ be a cyclic enumeration of the prism corners that are directly connected to $\mathbf{p}_j$, then the volume gradient is

$$\nabla_j V \;:=\; \frac{\partial V}{\partial \mathbf{p}_j} \;=\; \frac{1}{6}\sum_{i=0}^{4} \mathbf{u}_i \times \mathbf{u}_{(i+1) \bmod 5} \quad , \tag{5.1}$$

and the vertex $\mathbf{p}_j$ has to be shifted by

$$\mathbf{r}_j \;=\; \varepsilon\, \frac{\nabla_j V}{\|\nabla_j V\|^2}$$

if the prism volume is to be modified by ε.

5.4.2 Volumetric Detail Reconstruction

After the base surface $\mathcal{B}$ is deformed to $\mathcal{B}'$ by the editing operator of the multiresolution modeling framework, the detailed surface $\mathcal{S}'$ has to be reconstructed from $\mathcal{B}'$ and the geometric details, i.e., the displacement volumes. This means that we have to find a mesh $\mathcal{S}'$, such that the volumes V_j of the prisms spanned between $\mathcal{S}'$ and $\mathcal{B}'$ are identical to the volumes V_j^* enclosed between the original surfaces $\mathcal{S}$ and $\mathcal{B}$.

The correlation between $\mathcal{B}$ and $\mathcal{B}'$ has to be established by the editing operator. In our case, it is enough to know the positions of the new base points $\mathbf{b}_i' \in \mathcal{B}'$ that correspond to the base points $\mathbf{b}_i \in \mathcal{B}$. If the editing operator does not change the surface tessellation and hence the connectivities of $\mathcal{B}$ and $\mathcal{B}'$ are identical, then the correlation between $\mathbf{b}_i$ and $\mathbf{b}_i'$ is simply given by the barycentric coordinates of $\mathbf{b}_i$ with respect to its containing triangle in $\mathcal{B}$.

Volume Preservation

With the transformed base points $\mathbf{b}'_i \in \mathcal{B}'$ the new prisms $\left(\mathbf{p}'_i, \mathbf{p}'_j, \mathbf{p}'_k, \mathbf{b}'_i, \mathbf{b}'_j, \mathbf{b}'_k\right)$ can be defined, that depend on the yet unknown vertex positions $\mathbf{p}'_i \in \mathcal{S}'$. Since for each triangle f_j in $\mathcal{S}'$ the volume V_j of its corresponding prism should equal the initial value V_j^*, the global volume error can be measured by the functional

$$E\left(\mathcal{S}'\right) \;:=\; \sum_{f_j \in \mathcal{F}_{\mathcal{S}'}} \left(V_j^* - V_j\right)^2 \;\;,$$

that accumulates the squared errors of all prism volumes. The detail reconstruction problem then basically amounts to the minimization of this error functional, which can be done by a gradient descent method [PFTV92]. The required gradient of $E\left(\mathcal{S}'\right)$ w.r.t. a vertex $\mathbf{p}_i \in \mathcal{S}'$ consists of the partial derivatives of $\nabla_i V_j$ from (5.1):

$$\frac{\partial E}{\partial \mathbf{p}_i} \;=\; -\sum_{j \in \mathcal{P}_i} 2\,\left(V_j^* - V_j\right)\,\nabla_i V_j \;\;,$$

where the sum is built over all prisms incident to $\mathbf{p}_i$.

Finding good starting values for the iterative gradient descent is not trivial, since self-intersections in the initial configuration should be avoided. Extremely bad starting values will otherwise cause the iterative minimization to get stuck in a local minimum. A reasonable assumption is that the modified base surface $\mathcal{B}'$ has no self-intersections itself. Since the base points $\mathbf{b}'_i$ are lying on this surface, and since there is a one-to-one correspondence with the vertices $\mathbf{p}'_i$, starting from an initial mesh $\mathcal{S}'$ with $\mathbf{p}'_i = \mathbf{b}'_i$ will yield a clean initial configuration. This mesh actually corresponds to the solution of the volume preservation if all target volumes are scaled down to zero.

Based on this observation, the iterative volume preservation scheme is interleaved with scaling steps of the prism volumes. All target prism volumes V_j^* are initially scaled down by a factor $0 < h_0 < 1$ and the iterative volume preservation scheme is applied using $\mathbf{b}'_i$ as starting values. Upon convergence the factor is increased to $h_1 < h_2 < \ldots$ until the scaling 1 is reached. After each volume scaling step $h_k \mapsto h_{k+1}$, the positions $\mathbf{p}'_i$ of the previous round can be used as starting values for the next round.

Regularization

An analysis of the iterative volume preservation scheme reveals that the solution $\mathcal{S}'$ is not well-defined. Simply counting the degrees of freedom shows that each vertex of $\mathcal{S}'$

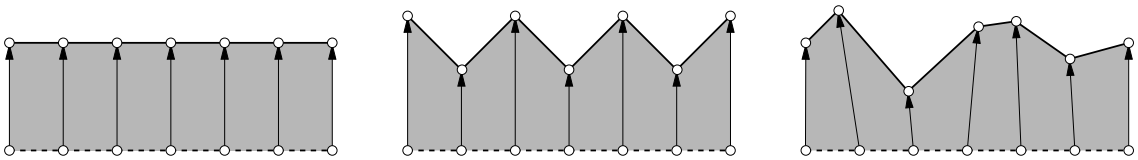


Figure 5.7: Constraining the prism volumes still leaves one degree of freedom per vertex. All three configurations preserve the target volumes, but may contain perturbations in normal direction (*center*) or tangential direction (*right*). These perturbations usually affect the highest frequency band.

yields three free parameters, while each triangle of $\mathcal{S}'$ puts one constraint. Since the number of triangles is approximately two times the number of vertices, it turns out that for a mesh $\mathcal{S}'$ with m vertices (and hence $2m$ triangles) the solution is underdetermined by $3m - 2m = m$ degrees of freedom. As a consequence, the above iterative scheme will converge to a solution, but not necessarily to the best one. Hence, a regularization force has to be added which pushes the iterative scheme towards a better solution of this underdetermined optimization problem.

A suitable regularization force for the displacement volume reconstruction can be found by examining the set of candidate meshes $\mathcal{S}'$ satisfying the volume constraints (cf. Fig. 5.7). Intuitively, each volume constraint fixes the average height of the corresponding prism's top face over the base face, which is equivalent to fixing the height of the centroid of the top triangle. As a consequence, small perturbations in normal direction lead to meshes which also satisfy the volume constraints, but exhibit rotations of the triangles around their centroids (cf. Fig. 5.7, center). Additional perturbations in tangential direction may be compensated for by adjusting the offset heights (cf. Fig. 5.7, right). This reveals that the variations among different volume preserving candidates are mostly on the highest frequency band and are therefore easily eliminated by a properly designed low-pass filter.

In order to reduce the influence of the base surface $\mathcal{B}'$ on the action of the regularization force, the low-pass filter is applied to the displacement vectors $\mathbf{d}'_i := \mathbf{p}'_i - \mathbf{b}'_i$ instead of to the points $\mathbf{p}'_i$. Since its impact on the convergence behavior of the volume preservation should also be limited, different filters are applied to the tangential component of the displacements and to the length component. Both filters make use of the fact that the

deformation of the base surface is smooth and hence local variations of displacements caused by local bending of the base surface are small.

A natural regularization is to push the minimization towards a volume preserving solution $\mathcal{S}'$ that has the least distortion in surface metric from the original surface $\mathcal{S}$. As shown in [PP93, DMA02], a discrete harmonic parameterization can be computed by Laplacian smoothing of the two-dimensional parameter values using the metric (i.e., the Laplace-Beltrami operator) of the 3D surface (Sect. 3.2.1). In the remeshing section is was also shown that tangential smoothing can be used to improve the regularity of a triangulation (Sect. 3.2.3). These two approaches are combined into a *tangential* Laplacian smoothing operator on $\mathcal{S}'$ using the surface metric of the *original* surface $\mathcal{S}$:

$$\mathbf{p}'_i \;\leftarrow\; \mathbf{p}'_i + \lambda \left(I - \mathbf{n}\left(\mathbf{p}'_i\right) \mathbf{n}\left(\mathbf{p}'_i\right)^T \right) \Delta_{\mathcal{S}}\left(\mathbf{p}'_i\right) \quad .$$

This relaxation operator moves the vertices $\mathbf{p}'_i$ in their respective tangent planes in order to minimize the metric distortion to the original surface $\mathcal{S}$.

For the regularization of the normal components, i.e., the lengths of the displacements $d'_i := \|\mathbf{p}'_i - \mathbf{b}'_i\|$, a correlation between neighboring base points has to be found. Let $d_i := \|\mathbf{p}_i - \mathbf{b}_i\|$ be the length of the displacements in the initial configuration $(\mathcal{S}, \mathcal{B})$ before the modification. Since the volume enclosed between these surfaces is to be preserved by $(\mathcal{S}', \mathcal{B}')$, the thickness of the incompressible layer is locally a function of the base surface stretch. Since the base surface stretch smoothly varies over $\mathcal{B}'$ (smooth deformation), the scaling of the d'_i will also vary smoothly, i.e., there exists a smooth scalar function $s : \mathcal{B} \to \mathbb{R}$, such that the lengths of the displacements after the deformation are approximately $d'_i \approx s_i\, d_i$ with $s_i := s(\mathbf{b}_i)$.

It follows that the regularizing filter for the displacement lengths should push the solution d'_i towards $s_i\, d_i$ for some unknown but smooth function s. Instead of minimizing the absolute differences $d'_i - s_i\, d_i$, the relative differences $d'_i/d_i - s_i$ are minimized. To obtain a smoothing filter, the Laplace operator is applied, this time using the metric of the original base surface $\mathcal{B}$:

$$\Delta_{\mathcal{B}}\left(\frac{d'_i}{d_i} - s_i\right) \;=\; \Delta_{\mathcal{B}}\left(\frac{d'_i}{d_i}\right) - \Delta_{\mathcal{B}}\left(s_i\right) \;\approx\; \Delta_{\mathcal{B}}\left(\frac{d'_i}{d_i}\right) \quad ,$$

where the term $\Delta_{\mathcal{B}}(s)$ can be neglected under the assumption that s is smooth. The requirement $\Delta_{\mathcal{B}}\left(d'_i/d_i\right) = 0$ immediately leads to the simple filter

$$d'_i \;\leftarrow\; d'_i + \lambda\, d_i\, \Delta_{\mathcal{B}}\left(\frac{d'_i}{d_i}\right) \quad .$$

This filter regularizes the length of the displacement vectors by taking the original lengths d_i into account. Notice that the lengths d_i are used in the regularization only: They help to stabilize, but they do not affect the volume preservation.

A standard method to combine the regularization forces with the volume optimization would be using Lagrangian multipliers, leading to a constrained minimization problem. However, for efficiency reasons several iterations of (unconstrained) volume optimization are interleaved with one regularization step. Similar to augmented Lagrangian methods the weight of the volume optimization over the regularization is increased during the optimization process.

The initial lack of constraints in the volume preservation may result in solutions containing self-intersections. These self-intersections, however, represent high frequencies in the surface and therefore are easily avoided by the regularization process. By consequence, when starting from a clean initial configuration, the regularization forces drive the iterative scheme to a volume preserving solution without perturbations in surface metric and displacement lengths, and therefore without self-intersections. Although we have no theoretical guarantees for a removal of all self-intersections, the described approach worked robustly in all our examples.

Implementation

The volume optimization as well as the regularization are relaxation methods. A well known result from numerical analysis states that these types of processes tend to rapidly smooth out high-frequency errors, but their convergence rate for the lower frequencies of the error is impractically slow [Hac86].

Therefore a hierarchical cascading multi-grid approach is used to increase the overall rate of convergence (see Chap. 7). Starting from $\mathcal{S}$, multiple levels of decreasing *topological* complexity are constructed by mesh decimation (see Sect. 3.2.2). Using the solutions computed on coarse levels as initial values for the optimization on finer levels leads to an efficient solver for the volume optimization. The complexity of the resulting hierarchical reconstruction operator is close to linear in the number of prisms, i.e., in the number of triangles of $\mathcal{S}$. One multigrid cycle can solve for about 14k prism volume constraints per second on a 2.8GHz Pentium4 processor. Since each prism is decomposed into 14 tetrahedra, this corresponds to about 200k tetrahedron volumes per second. As

described in the last section, the hierarchical volume optimization has to be solved on several volume scales $h_0 < \ldots < h_k = 1$ in order to robustly handle self-intersections, with k typically ranging from 5 to 8 depending on the complexity of the modification.

However, even when using this hierarchical solver, the overall computational complexity is too high for this reconstruction operator to be used for interactive mesh editing of complex models at interactive response times. In [MDM+02] deformations were simulated on a rather coarse volumetric tetrahedral mesh, but a finer triangle mesh was used to represent the skin surface of the model. In a similar way the volume optimization can be restricted to the coarse levels of the multigrid hierarchy only, such that the positions $\mathbf{p}_i'$ on the finest level are derived by the regularization forces alone, resulting in a speed-up of about 20% in our experiments.

We therefore propose to use this simpler technique (or even switch to displacement vectors) during the user's mouse motion in order to achieve faster response times, and to switch back to the exact computation once the user releases the mouse. In addition, since the volume optimization is based on a straightforward but rather inefficient gradient descent method, further performance gains can be expected by using a more sophisticated minimization scheme in the future.

5.5 Results

In this section the general behavior of the displacement volumes is shown on examples for synthetic and real-world datasets. In general, one can differentiate between convex and concave modifications. A convex modification of the base surface increases the opening angles of the volume prisms, causing the respective volumes to grow and the detailed surface to stretch. Therefore the volume preservation typically decreases the offset's height in these areas in order to decrease the volumes down to their original values. Concave modifications of $\mathcal{B}$ compress the volume prisms by decreasing their opening angles. Depending on the detail length and the local curvature of $\mathcal{B}'$, this may lead to self-intersections of the detailed surface $\mathcal{S}'$. The volume preservation will therefore have to expand the prisms both in normal and tangential directions (cf. Fig. 5.3).

Fig. 5.8 shows a multiresolution bending of a cylinder and a cuboid. The models in the top row are reconstructed by normal displacements and show an unnatural increase

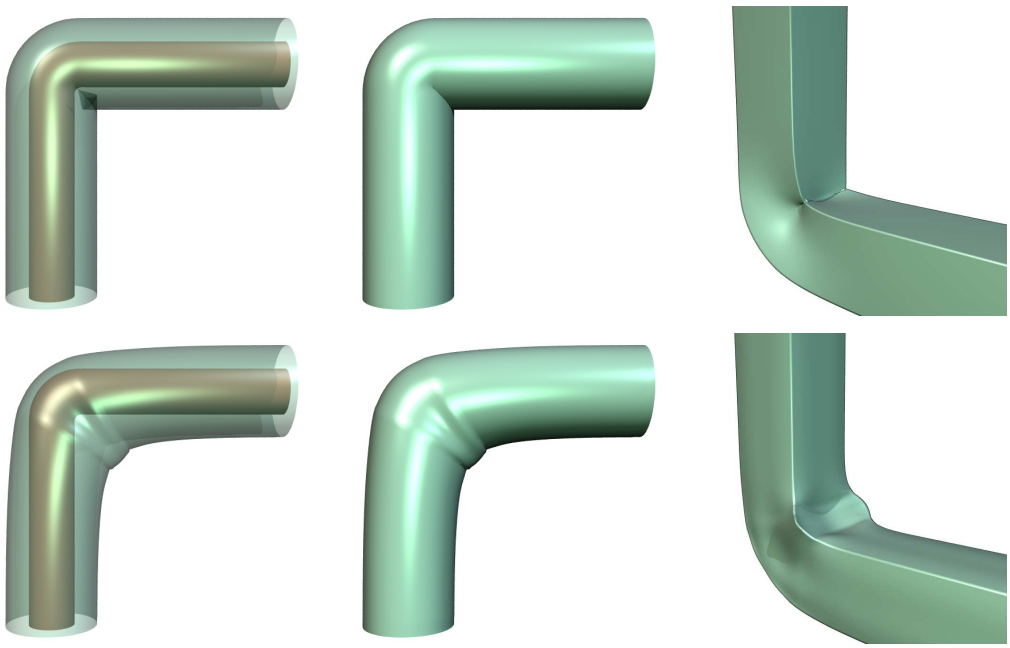


Figure 5.8: A cylinder and a cuboid bended by 90 degrees. Displacement vectors lead to unnatural changes in volume and self-intersections (*top row*), while displacement volumes manage to solve both problems (*bottom row*). In addition to this, displace volumes preserve high-frequency geometric details (*bottom right*).

of volume in the convex parts and self-intersections in concave regions. In the first column the detailed surfaces $\mathcal{S}'$ are rendered transparently in order to also show the base surface $\mathcal{B}'$. Displacement volumes avoid local self-intersections and preserve the volume, resulting in a more plausible detail reconstruction. The cuboid example additionally demonstrates that the reconstruction operator correctly handles high-frequency geometric detail, since the sharp edges are preserved and deformed in a very natural manner.

The two models shown in Fig. 5.8 are synthetic regular triangulations of moderate complexity. The following examples demonstrate the effectiveness and robustness of the volumetric detail representation for complex irregular meshes. Fig. 5.9 shows a scanned toy model of Tinky-Winky. When bending its arm, the normal-displaced surface self-intersects almost immediately, since the layer between base and detailed surface is rather thick and hence the displacement vectors are long. The depicted position contains severe

self-intersections in the normal-displaced setting, that are completely removed by the volumetric reconstruction operator.

In the example shown in Fig. 5.10, the left leg of Michelangelo's David is bent, which was cut out of a decimated version of the model and consists of 33k triangles. Again, normal displacements lead to self-intersections, while displacement volumes do not. Although this example is not anatomically correct (no different material properties are used to simulate different tissues), it effectively avoids self-intersections.

These examples show that the basic idea behind displacement volumes — keeping the volume enclosed between $\mathcal{S}$ and $\mathcal{B}$ constant — results in a more natural detail preservation of the deformed surfaces, mimicking the behavior of elastic but incompressible materials. In combination with the properly designed regularization force, displacement volumes also effectively avoid local self-intersections in the reconstructed detailed surface. Depending on the application, this can be extremely important, since self-intersecting surfaces are not orientable, i.e., they do not provide a clearly defined interior and exterior. As a consequence, there is no implicit representation for them, which prevents the use of all volumetric techniques presented in the first part of this thesis.

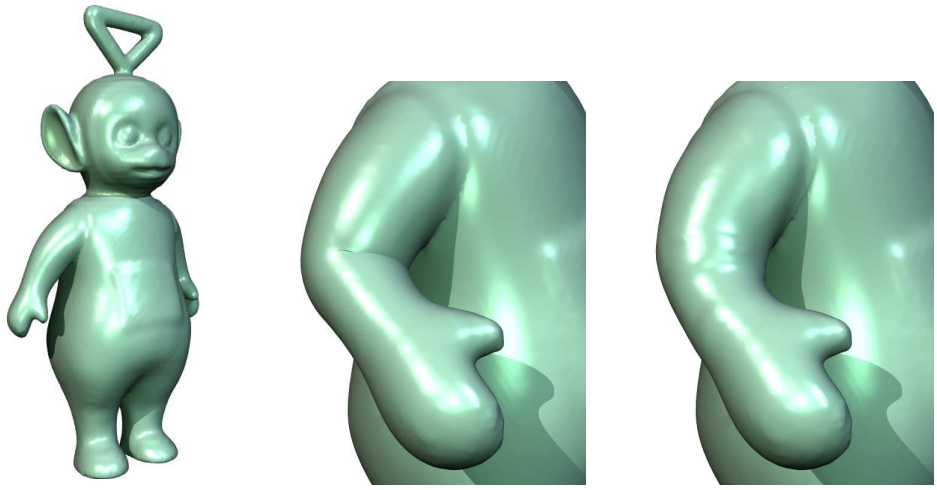


Figure 5.9: A scanned toy model was modified to bend its arm. Here displacement vectors create self-intersections immediately, while displacement volumes enable also larger scale modifications with natural detail preservation.

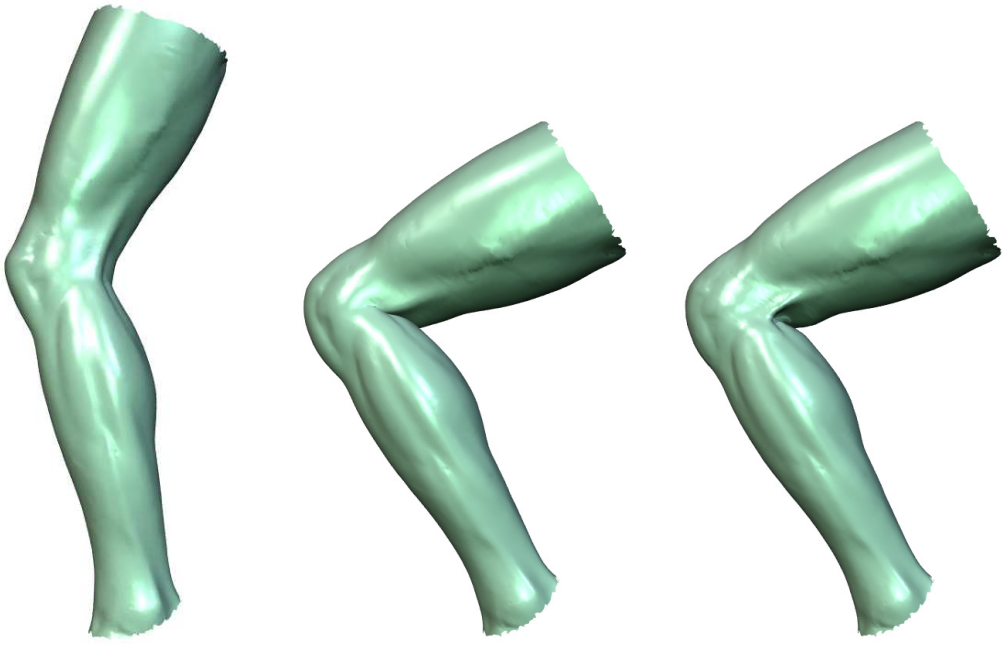


Figure 5.10: The left leg of Michelangelo's David is bended. In contrast to displacement vectors displacement volumes effectively avoid self-intersections at the hollow of the knee.

6 Freeform Surface Editing

After introducing different multiresolution surface representations in Chap. 5, we will now discuss the freeform editing operator as the last missing piece for a complete multiresolution modeling framework (see Sect. 5.1). The *boundary constraint modeling* approach we are going to present provides flexible and precise shape control and allows for real-time deformations even of highly complex meshes [BK04a].

All the methods presented in this chapter can be used to generate smooth surfaces, or to deform a given surface in a smooth manner. However, they do not take special care of sharp surface features and do not provide any means of local frame detail preservation. Consequently, these freeform editing techniques should be integrated into a multiresolution modeling framework, as described in the last chapter, in order to allow for a physically plausible deformation of technical datasets.

The fundamental problem of freeform editing is that the models to be deformed as well as the shape deformations to be applied to them can become highly complex. Surfaces obtained by reverse engineering of physical objects or by tessellating CAD models may easily consist of several millions of triangles. Although mesh decimation can be used to lower this complexity, the requirement of faithfully preserving the geometric features puts a limit to the simplification process. In a similar way the geometric complexity of the intended deformation is growing in parallel to the amount of geometric detail contained in the surface.

In the resulting complex space of possible geometric shapes, esthetically pleasing surfaces are sometimes lying surprisingly close to unacceptable ones. As a consequence, the designer usually has to explore the different possible shape deformations in an interactive manner. Unfortunately, the user interfaces for controlling such shape deformations are still very limited on today's systems. Although there exist sophisticated interaction technologies like haptic input devices, immersive displays [SPS01], or two-handed input metaphors [LKG+03], they did not replace the standard modeling workstation, that is

still controlled by a mouse and a 2D screen and can be found in every industrial design company.

To compensate for the limited user interface, intuitive modeling metaphors for conveying the intended shape deformation to the modeling system have to be found and used. The *control point* paradigm of spline and subdivision surfaces is a well-accepted technique, that has proven to provide intuitive means of geometry specification and deformation (see Sect. 2.1.1 and Sect. 2.1.2). In an interactive modeling session, the designer usually drags control points in 3-space, which provide three degrees of freedom each to model local translations. For more complex shape deformations, like rotations, several control points have to be dragged sequentially or simultaneously. An easier and more convenient user interface is offered by more general manipulator objects, so-called *control handles*, that provide nine degrees of freedom for specifying translations, rotations, and scalings.

In order to derive a mathematical formulation for the surface deformation process, let us denote by $\mathcal{S}$ the given shape, that the designer wants to modify into another shape $\mathcal{S}'$. Notice that if the freeform editing operator is eventually integrated into a multiresolution modeling system, it would rather be used to deform the original base surface $\mathcal{B}$ into a new version $\mathcal{B}'$, such that the deformed surface $\mathcal{S}'$ would be reconstructed from $\mathcal{B}'$ and the high frequency geometric details (see Chap. 5). However, for a clearer notation we will denote the original surface and its deformed version by $\mathcal{S}$ and $\mathcal{S}'$, respectively.

The process of deforming $\mathcal{S}$ into $\mathcal{S}'$ by dragging control points or control handles can be formulated as

$$\mathcal{S}' \;=\; \mathcal{S} + B\left(\delta\mathbf{C}\right) \quad , \tag{6.1}$$

where B represents an abstract basis function, and $\delta\mathbf{C}$ represents the change of position and orientation of the control handles. For a modeling tool based on NURBS surfaces, e.g., B would represent the set of tensor-product basis functions and $\delta\mathbf{C}$ the displacement vectors by which we shift the corresponding control points (see Sect. 2.1.1).

A complex shape modification requires the term $B\left(\delta\mathbf{C}\right)$ to become complex, which can be achieved in two fundamentally different ways. The first option is to require a complex change of the control handles $\mathbf{C}$, i.e., a complex user interaction, for instance by providing a complicated manipulator object with many degrees of freedom, or by adjusting a large number of control points. The alternative is to allow the user interaction $\delta\mathbf{C}$ to be simple, but then the necessary complexity has to be incorporated into the basis functions B.

Our goal is to keep the user interaction simple and intuitive, first, in order to compensate for the limited user interface, but also to allow even non-experienced users to perform sophisticated shape deformations. Therefore we will use special basis functions, that are custom-tailored to the intended shape deformation. We will see in the remainder of this chapter, that this means we have to be able to define an arbitrary support region for the shape deformation and to control smoothness, stiffness, and even bending behavior of the surface.

In turn, when restricting to a simple user interaction the possible types of modifications are limited by the abstract basis functions B that our system associates with the control handles. Taking this into consideration, we review existing freeform modeling approaches in the next section.

6.1 Existing Freeform Modeling Approaches

The traditional surface representation for CAGD are spline surfaces, that are controlled by the intuitive control point metaphor and provide high quality smooth surfaces. However, it was shown in Sect. 2.1.1 that spline surfaces are restricted to rectangular domains, and that complex surfaces therefore have to be composed by a large number of (possibly trimmed) spline patches. Subdivision surfaces can be considered as direct generalization of splines to surfaces of arbitrary topology, since the vertices of the coarse subdivision base mesh act like the control points of a spline surface, and, depending of the subdivision scheme, the iterated surface refinement even converges to a spline surface (Sect. 2.1.2).

In both cases, a smooth basis function is associated with every control point, such that each control point translation adds a smooth bump of either rectangular (splines) or polygonal support (subdivision surfaces) to the surface (cf. Fig. 6.1, top left). Every more sophisticated modeling operation has to be composed from these smooth elementary modifications. In the setting of Eq. (6.1), this corresponds to simple basis functions B of fixed support and fixed smoothness, and therefore a highly complex $\delta\mathbf{C}$, i.e., a highly complex user interaction, is necessary. This is one of the reasons why it takes skilled experts to operate a typical CAD system. Nevertheless, it is the widely accepted standard and implemented in many NURBS or subdivision based modeling frameworks.

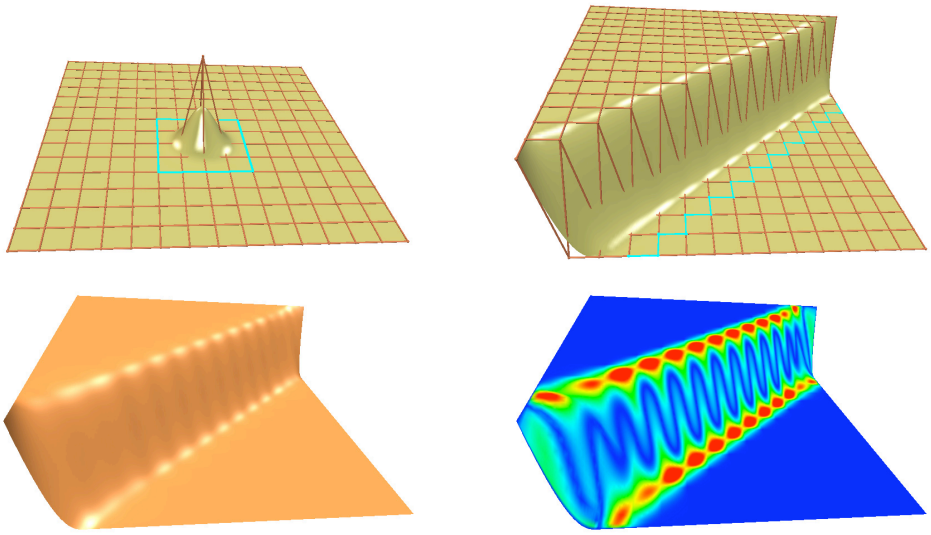


Figure 6.1: A modeling example using a bi-cubic tensor-product spline surface. Each control point is associated with a smooth basis function of fixed rectangular support (*top left*). This fixed support and the fixed regular placement of the control points, resp. basis functions, prevents a precise support specification (*top right*) and can lead to alias artifacts in the resulting surface, that are revealed by more sensitive surface shading (*bottom left*) and a mean curvature plot (*bottom right*).

But there are more problems regarding spline and subdivision modeling besides the complex user interaction. Any modification has to be composed of smooth bumps resulting from the movement of the respective control points. As a consequence, the support of the deformation is the union of the individual basis functions' supports. As the positions of these basis functions are fixed to the initial grid of control points, this prohibits a fine-grained control of the desired support region. Moreover, the composition of fixed basis functions located on a fixed grid might lead to alias artifacts in the resulting surface, as shown in the bottom row of Fig. 6.1. Because the placement of control points also defines the degrees of freedom for modeling operations, the initial patch layout is extremely important and usually has to be done by an experienced designer.

An important limitation of this approach and other modeling frameworks is that the underlying mathematical surface representation is tightly linked to the basis functions that are used for the surface deformation, i.e., to the type and number of control handles. This problem is also more than obvious if our geometry representation is based on

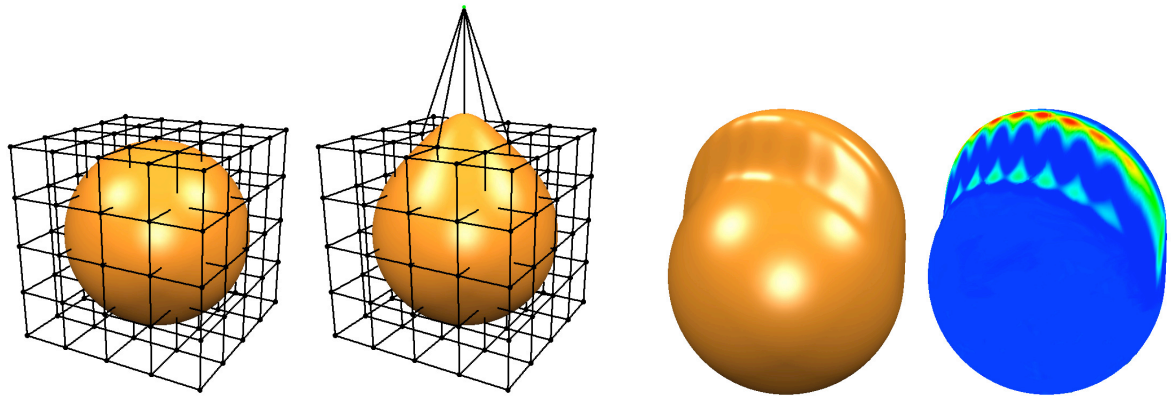


Figure 6.2: In the freeform deformation approach a regular 3D control lattice is used to specify a volumetric displacement function (*left*). Similar to tensor-product spline surfaces, the tri-variate tensor-product splines can also lead to alias artifacts in the deformed surface (*right*).

unstructured triangle meshes, since here, shifting a (control) vertex just adds a tiny hat function to the surface. To overcome this limitation, the deformation basis functions consequently have to be independent of the actual surface representation.

The standard method to achieve this separation of surface representation and surface deformation is to use a volumetric deformation function $\mathbf{d} : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ for transforming each surface sample: $\mathcal{S}' = \{\mathbf{d}(\mathbf{p}) \mid \mathbf{p} \in \mathcal{S}\}$. The most prominent example for this kind of approaches is the freeform deformation technique [SP86, Coq90, MJ96], where the deformation function is represented by a tri-variate tensor-product spline function, that can intuitively be deformed by a regular 3D control lattice (cf. Fig. 6.2, left).

While this is an intuitive modeling metaphor, it does not provide significantly more degrees of freedom and still leads to smooth bumps over simple support regions. Analogously to tensor-product spline surfaces, the volumetric spline basis functions are again located on a fixed regular grid, resulting in the same kind of alias artifacts in the deformed surface (cf. Fig. 6.2, right). In terms of Eq. (6.1), volumetric freeform deformation also corresponds to simple basis functions B and consequently requires complex control point adjustments. Additionally, the support of a volumetric modification is difficult to predict and control, as it is determined by intersecting the support of volumetric basis functions with the surface to be modified.

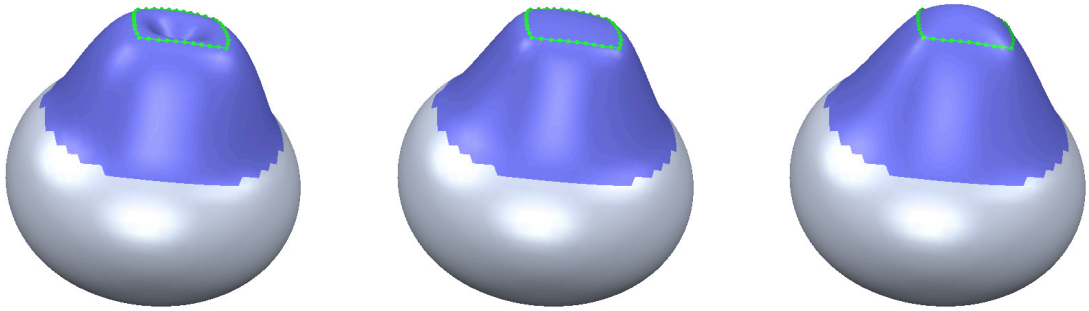


Figure 6.3: A sphere is deformed by lifting a closed handle polygon. Propagating this translation of the control handle based on geodesic distance causes a dent in the interior of the handle polygon (*left*). Using a partition-of-unity approach avoids the dent by rigidly transforming of the interior part (*center*). The most intuitive solution of a smooth interpolation (*right*), however, cannot be achieved with this kind of approach.

Another large class of deformation approaches directly transforms a user-defined handle region and propagates this deformation outwards over the mesh [SF98, PKKG03, BK03a]. This corresponds to defining a scalar field on the surface that is used to damp the handle transformation and whose values are 1 at the handle region (full deformation) and decrease as a function of Euclidean or geodesic distance from the handle. As shown in Fig. 6.3, such distance-based propagation cannot yield the geometrically most intuitive solution, that would be the interpolation of the (transformed) handle region by a high quality smooth surface.

The necessary smooth deformation of a surface with prescribed boundary conditions is most elegantly modeled by an energy minimization principle [MS92, WW92, KCVS98]. The surface is assumed to behave like a physical skin, which stretches and bends as forces are acting on it. Mathematically, this behavior can be captured by an energy functional which penalizes stretch or bending. Then the optimal surface is the one that minimizes this energy while satisfying all the prescribed boundary conditions. The advantage of this formulation is that it allows to take arbitrary boundary conditions into account and that the optimal solution is known to have certain smoothness properties. When changing the boundary conditions, the optimal surface changes accordingly and this is why we call this approach *boundary constraint modeling* (BCM).

Because of its flexibility and high surface quality we also use the boundary constraint modeling approach. We adopt the modeling metaphor of Kobbelt et al. [KCVS98] and extend it to meet the central requirements of our modeling system. The very recent boundary constraint modeling approaches of [LSCO$^+$04, SCOL$^+$04, YZX$^+$04] are also very similar to [KCVS98], since they are based on the same partial differential equations and hence yield comparable results. However, these approaches differ in the way the multiresolution details are encoded and how the boundary constraints are specified by the designer. In contrast to them, we are able to obtain more flexible and more precise shape control by explicitly controlling the smoothness and bending behavior of the surface. Additionally, a pre-computation of a set of linear basis functions allows us to achieve real-time feedback even when modifying complex surface areas.

6.2 Boundary Constraint Modeling

As shown in Eq. (6.1), a freeform modification is performed by adding an abstract basis function B to the current surface $\mathcal{S}$. To be able to control even complex deformations by a simple user interaction, this basis function has to be custom-tailored to the intended shape deformation. For the specification of a particular modification's basis function, we have to define its *support*, i.e., the region of the surface $\mathcal{S}$ that should be affected by the modification, and its *characteristic shape*.

We will see below that our BCM approach allows us to specify interpolation constraints for an arbitrary set of vertices. As a consequence, this allows for a maximally precise specification of the support region on a per-vertex basis. The support is chosen to be an arbitrary surface region (a set of vertices), that may be convex or non-convex, with an arbitrary boundary curve, such that it can easily be aligned to any feature on the surface. A simple and intuitive way to let the designer specify this region is to draw onto the surface either its outline or the complete region using some painting metaphor.

In order to map the control of the modification to a 9-DoF manipulator object, the user selects a second region, the *handle region*, in the interior of the support region (cf. Fig. 6.4, left). The manipulator is then rigidly attached to this surface patch, such that moving the manipulator moves the surface patch accordingly. The remaining part of the surface, i.e., support region minus handle region, is supposed to smoothly bend according to the translation, rotation and scaling of the handle region (cf. Fig. 6.4).

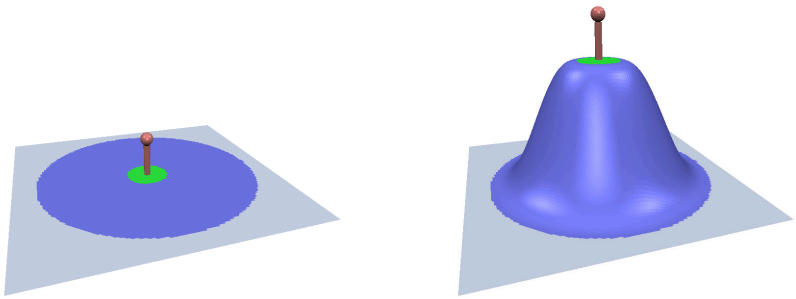


Figure 6.4: In our modeling metaphor, we define a custom tailored basis function by selecting a support region (*blue/dark*) and a handle region (*green/light*). If the handle region is transformed (by moving the manipulator object) the support region is bent to smoothly interpolate its inner and outer boundaries.

6.2.1 Constrained Surface Optimization

A physically plausible bending behavior of the surface is achieved by considering the fixed vertices and handle vertices as interpolation constraints and determining the remaining degrees of freedom (the support region) by a constrained minimization of a so-called *fairness functional* [MS92, WW92, KCVS98]. This functional punishes high curvature and therefore its minimization leads to surfaces that interpolate the constraints, but are otherwise free of unnecessary details, thereby following the *principle of simplest shape* [Sap94]. During a modeling session, each time the designer transforms the handle by moving the manipulator, the boundary constraints for the optimization are changed and the support region is re-computed to minimize its bending energy.

A geometrically very intuitive measure for the bending energy of a surface $\mathcal{S}$ is the so-called *thin-plate energy*, that is computed by integrating over the squared principal curvatures of $\mathcal{S}$:

$$E_{TP}(\mathcal{S}) \;=\; \int_{\mathbf{x}\in\mathcal{S}} \kappa_1^2(\mathbf{x}) + \kappa_2^2(\mathbf{x}) \; d\mathbf{x} \;. \tag{6.2}$$

However, this functional is highly non-linear, since it is based on intrinsic surface curvatures, and therefore its minimization is computationally too expensive for interactive modeling applications.

For efficiency reasons, E_{TP} is therefore approximated by replacing the intrinsic curvatures κ_1 and κ_2 by second order partial derivatives w.r.t. a surface parameterization $\mathbf{f} : \Omega \to \mathbb{R}^3$:

$$\tilde{E}_{TP}(\mathbf{f}) = \int_{\mathbf{x}\in\Omega} \|\mathbf{f}_{uu}(\mathbf{x})\|^2 + 2\|\mathbf{f}_{uv}(\mathbf{x})\|^2 + \|\mathbf{f}_{vv}(\mathbf{x})\|^2 \, d\mathbf{x} \ . \tag{6.3}$$

Since both energy functionals are, e.g., invariant to rigid motions, proper boundary constraints have to be imposed on them. These constraints are given by the fixed vertices and handle vertices of the surface, that define C^0 and C^1 interpolation constraints at the boundary $\delta\Omega$ of the support region. If we denote by $\mathcal{BC}$ the space of functions $\mathbf{f} : \Omega \to \mathbb{R}^3$ that satisfy the given boundary constraints, then we are looking for the optimal surface $\mathcal{S}^*$ defined by its parameterization

$$\mathbf{f}^* := \operatorname*{argmin}_{\mathbf{f}\in\mathcal{BC}} \tilde{E}_{TP}(\mathbf{f}) \ .$$

In order to efficiently compute the solution of the above minimization problem, variational calculus is applied to derive the Euler-Lagrange PDE that characterizes the minimizer $\mathbf{f}^*$ of $\tilde{E}_{TP}$ [Kob97]:

$$\Delta^2 \mathbf{f}^*(\mathbf{x}) = 0, \quad \forall\, \mathbf{x} \in \Omega \setminus \delta\Omega \ , \tag{6.4}$$

again with proper C^1 constraints on the boundary $\delta\Omega$. Hence, the optimal surface $\mathcal{S}^*$ can directly be computed by solving (6.4), which is usually more efficient than a minimization of the energy functional (6.3).

The quality of the parameterization-dependent approximation $\tilde{E}_{TP}$ to the exact geometric intrinsic E_{TP} — and hence the fairness of the resulting surface $\mathcal{S}^*$ — strongly depends on the parameterization used. This parameterization should be chosen as close as possible to isometric, since then the second partial derivatives yield a good approximation to the intrinsic principal curvatures. In the optimal case of an isometric parameterization, the functionals E_{TP} and $\tilde{E}_{TP}$ are identical.

Finding a close-to-isometric surface parameterization $\mathbf{f}^*$ for the yet unknown solution $\mathcal{S}^*$ of the minimization problem is not possible. However, following the general idea of the data-dependent fairness functionals of Greiner et al. [Gre94, GLW96], the initial surface $\mathcal{S}$ (instead of a planar region $\Omega \subset \mathbb{R}^2$) can be used as parameter domain. If the optimal surface $\mathcal{S}^*$ is not too far from the initial $\mathcal{S}$, i.e., if the surface deformation is not

too extreme, then the parameterization $\mathbf{f}^* : \mathcal{S} \rightarrow \mathcal{S}^*$ will be close to isometric. This can, e.g., be achieved by decomposing any large scale deformation into a sequence of smaller ones.

When $\mathcal{S}$ is used as parameter domain, the parameterizations $\mathbf{f}$ are functions over the reference surface $\mathcal{S}$, and therefore the partial derivatives of $\mathbf{f}$ have to be computed using the gradient $\nabla_{\mathcal{S}}$ induced by the metric of $\mathcal{S}$ [dC76]. Analogously, the Laplace operator used in the Euler-Lagrange PDE (6.4) should also be based on this surface metric, i.e., it should be replaced by the Laplace-Beltrami operator $\Delta_{\mathcal{S}}\,\mathbf{f} = \mathrm{div}_{\mathcal{S}} \nabla_{\mathcal{S}}\,\mathbf{f}$ w.r.t. the reference surface $\mathcal{S}$:

$$\Delta_{\mathcal{S}}^2\,\mathbf{f}^*\,(\mathbf{x}) = 0, \quad \forall\,\mathbf{x} \in \Omega \setminus \delta\Omega \ . \tag{6.5}$$

Notice that the resulting minimizer surfaces satisfying $\Delta_{\mathcal{S}}^2\,\mathbf{f} = 0$ are also limiting surfaces of explicit bi-Laplacian smoothing

$$\mathbf{f}\,(v_i) \ \leftarrow \ \mathbf{f}\,(v_i) - \lambda\,\Delta_{\mathcal{S}}^2\,\mathbf{f}\,(v_i)$$

as well as of the fourth order surface diffusion flow [DMSB99]

$$\frac{\partial\,\mathbf{f}\,(v_i)}{\partial t} \ = \ -\lambda\,\Delta_{\mathcal{S}}^2\,\mathbf{f}\,(v_i) \quad ,$$

since the local update vectors, i.e., the squared Laplacians, vanish for $\mathcal{S}^*$. This tight connection between Laplacian surface smoothing (see Sect. 3.2.1) and constrained surface optimization gives another reason why both approaches yield high quality smooth surfaces.

But in contrast to the implicit fairing approach, the Laplace-Beltrami is always computed w.r.t. the initial reference surface $\mathcal{S}$, i.e., the surface metric is not updated, which corresponds to the simplification of the non-linear E_{TP} (6.2) to the linearized $\tilde{E}_{TP}$ (6.3). However, this can also be regarded as an advantage, since the resulting surface $\mathcal{S}^*$ will have a low metric distortion to $\mathcal{S}$. The physical interpretation is that $\mathcal{S}^*$ on the one hand minimizes its bending energy, but on the other hand is also influenced by the material stretching due to the deformation of the initial state $\mathcal{S}$.

Besides the thin-plate energy, that punishes high surface curvature, one can also consider other functionals of different order, which results in a different bending behavior, corresponding to a different *stiffness* of the surface. The *membrane* energy

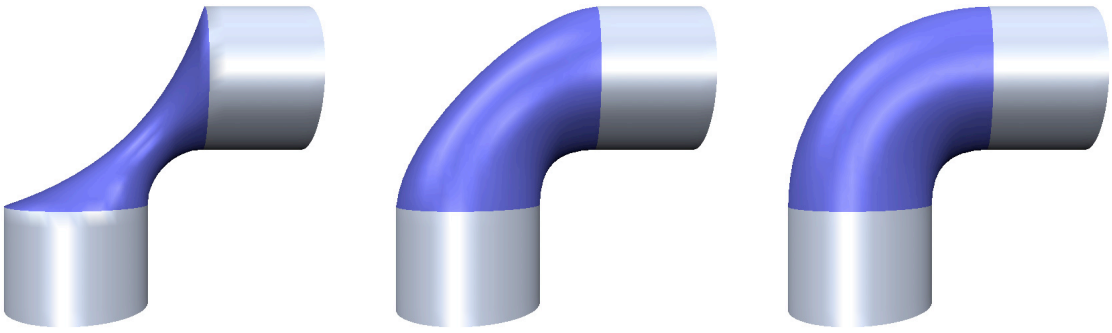


Figure 6.5: The order k of the energy functional and of the corresponding Euler-Lagrange PDE $\Delta_{\mathcal{S}}^{k}\,\mathbf{f}=0$ defines the stiffness of the surface in the support region and the maximum smoothness C^{k-1} of the boundary conditions. From left to right: membrane surface ($k=1$), thin-plate surface ($k=2$), minimum variation surface ($k=3$).

$\int_{\mathbf{x}\in\Omega}\left\|\mathbf{f}_u\left(\mathbf{x}\right)\right\|^2+\left\|\mathbf{f}_v\left(\mathbf{x}\right)\right\|^2\,d\mathbf{x}$ measures surface area and their minimizers can be characterized analogously by the second order PDE $\Delta_S\,\mathbf{f}=0$. The so-called *minimum variation surfaces* [MS92] minimize the integral over curvature derivatives and therefore yield surfaces with a uniform (mean) curvature distribution. They can be computed by solving the sixth order PDE $\Delta_S^3\,\mathbf{f}=0$. The different surface types obtained by these three energy functionals are shown in Fig. 6.5, where it can also be seen that surfaces derived from $\Delta_S^k\,\mathbf{f}=0$ interpolate boundary constraints of order C^{k-1}.

6.2.2 Linear System Derivation

Since our underlying surface representation is a triangle mesh, the Euler-Lagrange PDEs $\Delta_{\mathcal{S}}^{k}\,\mathbf{f}=0$ have to be discretized. Like in Sect. 3.2.1 the Laplace-Beltrami discretization of [DMSB99, MDSB03] is used and higher order Laplacians are defined recursively by

$$\begin{aligned}\Delta_{\mathcal{S}}^{k}\,\mathbf{f}\left(v_v\right) &:= \Delta_{\mathcal{S}}\left(\Delta_{\mathcal{S}}^{k-1}\,\mathbf{f}\left(v_i\right)\right),\\ \Delta_{\mathcal{S}}^{0}\,\mathbf{f}\left(v_i\right) &:= \mathbf{f}\left(v_i\right),\end{aligned}\tag{6.6}$$

such that the kth order Laplacian of a vertex v_i depends on its k-ring neighborhood.

With this discrete Laplace operator, the PDE $\Delta_{\mathcal{S}}^k \mathbf{f} = 0$ together with its boundary constraints leads to the following sparse linear system to be solved:

$$\left(\begin{array}{c} \Delta_{\mathcal{S}}^k \\ \hline \begin{array}{ccc} 0 & I & 0 \\ 0 & 0 & I \end{array} \end{array}\right) \left(\begin{array}{c} P \\ F \\ H \end{array}\right) = \left(\begin{array}{c} 0 \\ F \\ H \end{array}\right) \quad , \tag{6.7}$$

where $P = (\mathbf{p}_1, \ldots, \mathbf{p}_n)^T \in \mathbb{R}^{n \times 3}$ is the vector of free vertices in the support region, $F = (\mathbf{f}_1, \ldots, \mathbf{f}_l)^T \in \mathbb{R}^{l \times 3}$ are the fixed vertices outside the support region and $H = (\mathbf{h}_1, \ldots, \mathbf{h}_m)^T \in \mathbb{R}^{m \times 3}$ are the vertices of the handle region. These sets of vertices correspond to the *blue*, *gray* and *green* surface regions shown in Fig. 6.4, respectively. Solving this system yields the optimal surface $\mathcal{S}^*$ as a sampling of its parameterization at the vertex positions $\mathbf{p}_i = \mathbf{f}^*(v_i)$.

Since the vertex sets F and H are fixed, they impose the boundary conditions on the system. Notice that the necessary C^{k-1} boundary constraints are not specified explicitly, e.g., by prescribing the values of $\Delta_{\mathcal{S}}^j \mathbf{f}$, $0 \leq j < k$, at the boundary. Instead, they are implicitly prescribed by the first k rings of fixed vertices and handle vertices around the support region, since these define the higher order Laplacians. As a consequence, only the first k rings of constraints $F \cup H$ are required for the solution of Eq. (6.7).

Also notice that multiple independent handle regions, each controlled by its own manipulator widget, are no problem for the presented approach. In that case, the set of handle vertices H just has to be split into several components, such that each of them can be transformed independently by the designer. Because this is just a different user interface for specifying the boundary constraints, the energy minimization as well as the linear system (6.7) are not affected by this generalization.

The optimality conditions and the boundary conditions are combined into one equation for the sake of a simpler notation only. For the actual solution of the linear system, the constraints F and H should be moved to the right hand side of the system, which eliminates their corresponding rows and columns from the matrix, such that only the upper left $n \times n$ block remains. Whenever the designer moves the handle region, the vertices in H change their position and provide a new right hand side for the linear system. By solving it again the new vertex positions in P are computed as a linear function of H and F. We analyze the structure of these linear systems and propose robust and efficient ways of solving them in Chap. 7.

The modeling framework described so far basically resembles the approach of Kobbelt et al. [KCVS98]. However, one important difference is the choice of the Laplace operator and its discretization. The uniform Laplace operator used in [KCVS98] is a bad approximation in the case of irregular tessellations, where it causes tangential vertex movements within the surface and low fairness due to geometric artifacts. In contrast, the data-dependent Laplace-Beltrami operator $\Delta_{\mathcal{S}}$, that is discretized using the cotangent weights of the reference surface $\mathcal{S}$, is almost independent of the triangulation and leads to high quality surface of superior fairness (cf. Fig. 6.6).

Since the approach of Kobbelt et al. [KCVS98] was primarily targeted at conceptual design, it is not readily suitable for deformations of technical models in engineering applications. In the following we therefore propose extensions that yield more precise control over the surface and allow for the real-time solution of the presented linear systems even for complex models.

6.2.3 Boundary Smoothness

When specifying the deformation basis function B, a very important characteristic is its boundary smoothness, that determines how smooth the deformed surface region blends with the fixed part of the surface and the transformed handle region.

When prescribing the required k rings of fixed vertices, surfaces derived by solving $\Delta_{\mathcal{S}}^{k}\,\mathbf{f} = 0$ can be shown to interpolate boundary conditions of the order C^{k-1}, which can clearly be seen in Fig. 6.5. However, there are many cases in which this default behavior does not lead to the desired results, e.g., when different smoothness values for the inner and outer boundaries are intended. Fig. 6.7 shows examples for minimum variation surfaces derived from $\Delta_{\mathcal{S}}^{3}\,\mathbf{f} = 0$ with C^0 and C^2 boundary constraints.

Since k rings of boundary constraints are required to guarantee a C^{k-1} boundary smoothness, a lower order continuity can be achieved by somehow neglecting the influence of the outer rings of boundary constraints. But as all k rings of constrained vertices are necessary to properly compute the kth order Laplacians of the outmost ring of free vertices P, these constraints cannot simply be removed. One the other hand, the recursive computation of $\Delta_{\mathcal{S}}^{k}\,\mathbf{f}$ for the free vertices P only requires the lower order Laplacians $\Delta_{\mathcal{S}}^{k-1}\,\mathbf{f}, \ldots, \Delta_{\mathcal{S}}^{0}\,\mathbf{f}$ at the *first* ring of constrained vertices — which are, in turn, defined by the other $k-1$ rings of constraints.

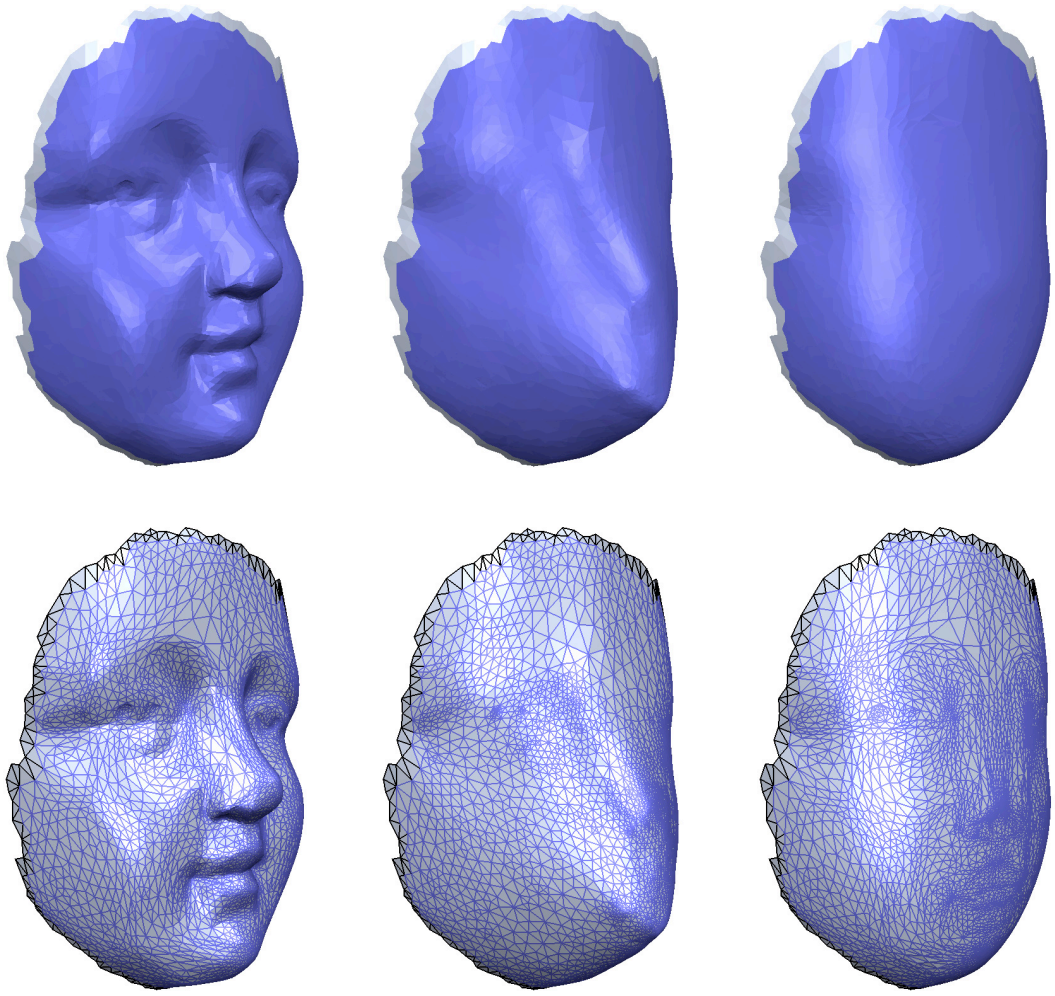


Figure 6.6: The original surface $\mathcal{S}$ (*left*) has an irregular tessellation with higher vertex density in the region of the mouth and the nose. This mesh is smoothed by minimizing $\tilde{E}_{TP}$, i.e., by solving Eq. (6.7). Using the *uniform* Laplace discretization Δ_{uni} in this linear system results in geometric surface artifacts and strong tangential movements within the surface (*center*). The cotangent discretization of the data-dependent Laplace-Beltrami operator $\Delta_{\mathcal{S}}$ avoids tangential movements and leads to the expected fair surface (*right*).

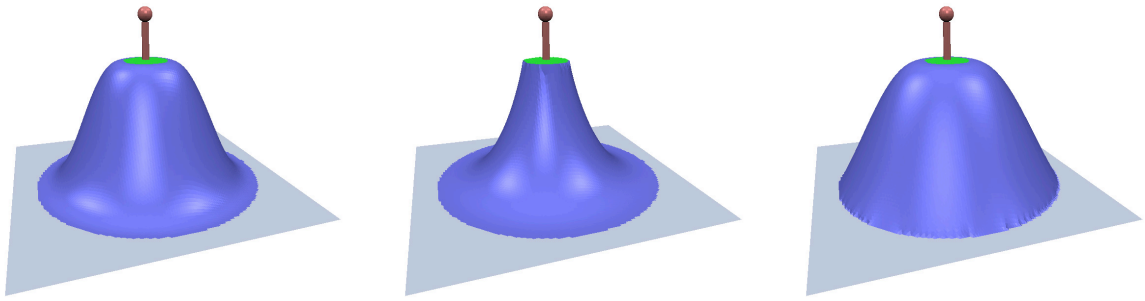


Figure 6.7: In our framework the smoothness conditions at the inner and outer boundary of the support region can be controlled independently and continuously blended between C^0 and C^2. From left to right: C^2 at inner and outer boundary, C^0 at inner and C^2 at outer, as well as C^2 at inner and C^0 at outer boundary.

However, by explicitly setting $\Delta_{\mathcal{S}}^{j}\mathbf{f}(f_i) = \Delta_{\mathcal{S}}^{j}\mathbf{f}(h_i) = 0$, $s < j < k$, for the *boundary* vertices $F \cup H$, only the first s rings of constrained vertices affect the solution and the remaining outer $k - s$ rings are neglected. As a consequence, the resulting boundary continuity is C^s.

A more fine-grained control of the boundary smoothness can be achieved by blending the continuities between C^0 and C^{k-1} using a real-valued smoothness parameter $c(v_i) \in [0, k-1]$ for the constrained vertices $v_i \in F \cup H$. The recursive definition of the higher order Laplacian from Eq. (6.6) is extended by damping values $\lambda_i \in [0,1]$ in order to take this per-vertex smoothness value into account:

$$\begin{aligned}
\bar{\Delta}_{\mathcal{S}}^{k}\mathbf{f}(v_i) &:= \Delta_{\mathcal{S}}\left(\lambda_{k-1}(v_i) \cdot \bar{\Delta}_{\mathcal{S}}^{k-1}\mathbf{f}(v_i)\right) \\
\bar{\Delta}_{\mathcal{S}}^{1}\mathbf{f}(v_i) &:= \Delta_{\mathcal{S}}\mathbf{f}(v_i) \\
\\
\lambda_k(v_i) &:= \begin{cases} 1, & c(v_i) > k \\ c(v_i) - k, & k-1 \le c \le k \\ 0, & c(v_i) < k-1 \end{cases} .
\end{aligned}$$

For instance, minimum variation surfaces are computed based on third order Laplacians. When using the modified Laplace operator

$$\bar{\Delta}_{\mathcal{S}}^{3}\mathbf{f}(v_i) = \Delta_{\mathcal{S}}(\lambda_2 \cdot \Delta_{\mathcal{S}}(\lambda_1 \cdot \Delta_{\mathcal{S}}\mathbf{f}(v_i))) ,$$

the boundary smoothness can continuously be blended between C^0 and C^1 ($\lambda_2 = 0$, $\lambda_1 \in [0,1]$) and between C^1 and C^2 ($\lambda_2 \in [0,1]$, $\lambda_1 = 1$).

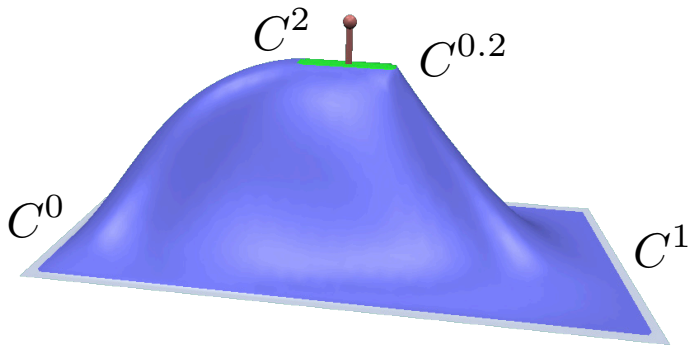


Figure 6.8: The continuous boundary smoothness can be controlled on a per-vertex basis. This allows to specify different continuities for arbitrary segments of the inner and outer boundaries of the support region.

This continuous boundary smoothness can precisely be controlled on a per-vertex basis, such that different continuities can easily be specified for different segments of the inner and outer boundary of the support region, as shown in Fig. 6.8. The user interaction for specifying the boundary smoothness is still easy and intuitive: First a (segment of a) boundary is selected and then the desired smoothness is chosen within $[0, k-1]$ using a slider widget.

6.2.4 Anisotropic Bending

One of the major goals of our freeform modeling framework is a simple yet powerful user interface, allowing the designer to specify even complex deformations by just a few intuitive parameters. The support and handle regions can easily and precisely be defined by painting them onto the surface, and their choice intuitively controls the extend and the "fullness" of the deformation, respectively. The characteristic shape of the surface is mainly determined by the surface stiffness (i.e., the energy functional) and by the boundary continuities. However, the framework described so far may still lead to rather counter-intuitive deformation results in the case of an anisotropically shaped support region, as we will show below.

Each time the designer changes the boundary constraints by dragging the manipulator, the deformed surface $\mathcal{S}'$ is derived by minimizing the chosen energy functional, i.e.,

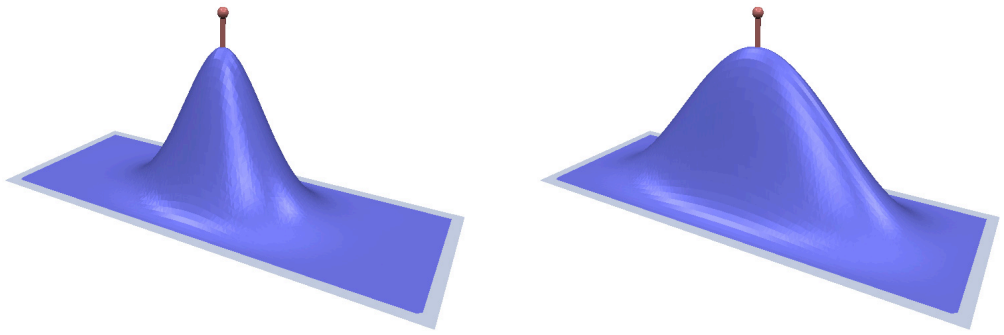


Figure 6.9: A simple modeling example using isotropic (*left*) and anisotropic (*right*) basis functions. The better adaptation of the anisotropic shape deformation to the support's shape results in a more natural bending behavior.

by solving the Laplacian system (6.7) for the new right-hand side. Consequently, the Laplace operator and its discretization have a large impact on the resulting surface (cf. Fig. 6.6), and the Laplace-Beltrami $\Delta_{\mathcal{S}}$ w.r.t. the original undeformed surface $\mathcal{S}$ was shown to yield high quality fair surfaces.

Just as the two-dimensional Laplace operator governs the uniform heat propagation in the plane, the above Laplace-Beltrami operator models a geodesically uniform propagation of the handle transformation over the support region. Due to this geodesically isotropic diffusion, the resulting optimal surface will also bend isotropically, i.e., the elementary modifications are mainly shaped like circular bumps, even if the shape of the support is anisotropically stretched (cf. Fig. 6.9, left).

Because the shape of the support region is considered as a design parameter for specifying the basis function of the intended deformation, a bending behavior that better adapts to the shape of the support would be much more intuitive. As shown in Fig. 6.9, the basis functions look more natural if the impact of the handle transformation is propagated through the support region, such that its "iso-contours" hit the outer boundary at the same time and with approximately the same slope.

Since the reason for this behavior is the isotropic Laplace operator, one way to achieve an anisotropic deformation would be to use an anisotropic formulation of the Laplacian. This, however, would complicate the overall computation and have a negative impact on the possible rate of interactivity. In contrast, we propose to switch back to a two-dimensional parameter domain Ω and to properly adjust this domain, respectively the

surface parameterizations $\mathbf{f}$, such that the Laplacian $\Delta_\Omega \mathbf{f}$ w.r.t. this domain results in the desired anisotropic bending behavior (cf. Fig. 6.9, right). The overall process is depicted in Fig. 6.10.

In a first step a parameterization of the support region $\mathbf{f} : \Omega \rightarrow \mathcal{S}$ is computed, such that the resulting optimal surface $\mathbf{f}^*$ with $\Delta_\Omega \mathbf{f}^* = 0$ is close to the isometric result derived by solving $\Delta_\mathcal{S} \mathbf{f} = 0$. This initial parameterization of the support region is computed using the Least Squares Conformal Map (LSCM) approach of Lévy et al. [LPRM02], yielding as parameter domain Ω a planar triangulation with identical connectivity, i.e., a flattened version of the support region. Because the resulting conformal parameterization minimizes (angular) distortion, the two-dimensional domain Ω will be anisotropically stretched in a similar way as the support region is in 3D.

Then a principal component analysis on the planar triangulation Ω is computed in order to scale it along its principal axes, such that its diameter is approximately the same in each direction. The scaled parameter domain Ω' now corresponds to parameterizations $\mathbf{f} : \Omega' \rightarrow \mathcal{S}$ that are no longer conformal, but contain a certain amount of stretch. This, however, is exactly the stretching that is intended for the anisotropic deformation. Hence, using Ω' as the underlying parameter domain and solving $\Delta_{\Omega'} \mathbf{f} = 0$ for the optimal surface will result in the desired anisotropic bending behavior.

The special parameterizations $\Omega' \rightarrow \mathcal{S}$ can therefore be considered to "factor out" the anisotropy, such that w.r.t. these parameterizations the standard *isotropic* Laplace operator can be used. This means, that once the parameterization is generated, the computational complexity for deriving the optimal surface is the same as in the simple isotropic case. As the only difference the cotangent weights of the discrete Laplacian are computed on the planar triangulation Ω instead of being derived from the 3D surface $\mathcal{S}$.

Since the special parameterization has to be computed only once after selecting the support region (instead of each time the handle is moved), it can be regarded as a pre-computation process. Nevertheless, its computation should be sufficiently fast to be useful in an interactive modeling application. Computing the initial LSCM parameterization requires the solution of a large sparse linear least squares system. In order to accelerate the iterative solver used in [LPRM02], Ray and Lévy [RL03] propose a hierarchical multi-grid solver. However, using a sparse Cholesky solver as described in Chap. 7 has the same asymptotic linear complexity, but is faster by a factor of about 4 compared to the results presented in [RL03] (see Sect. 7.3.7).

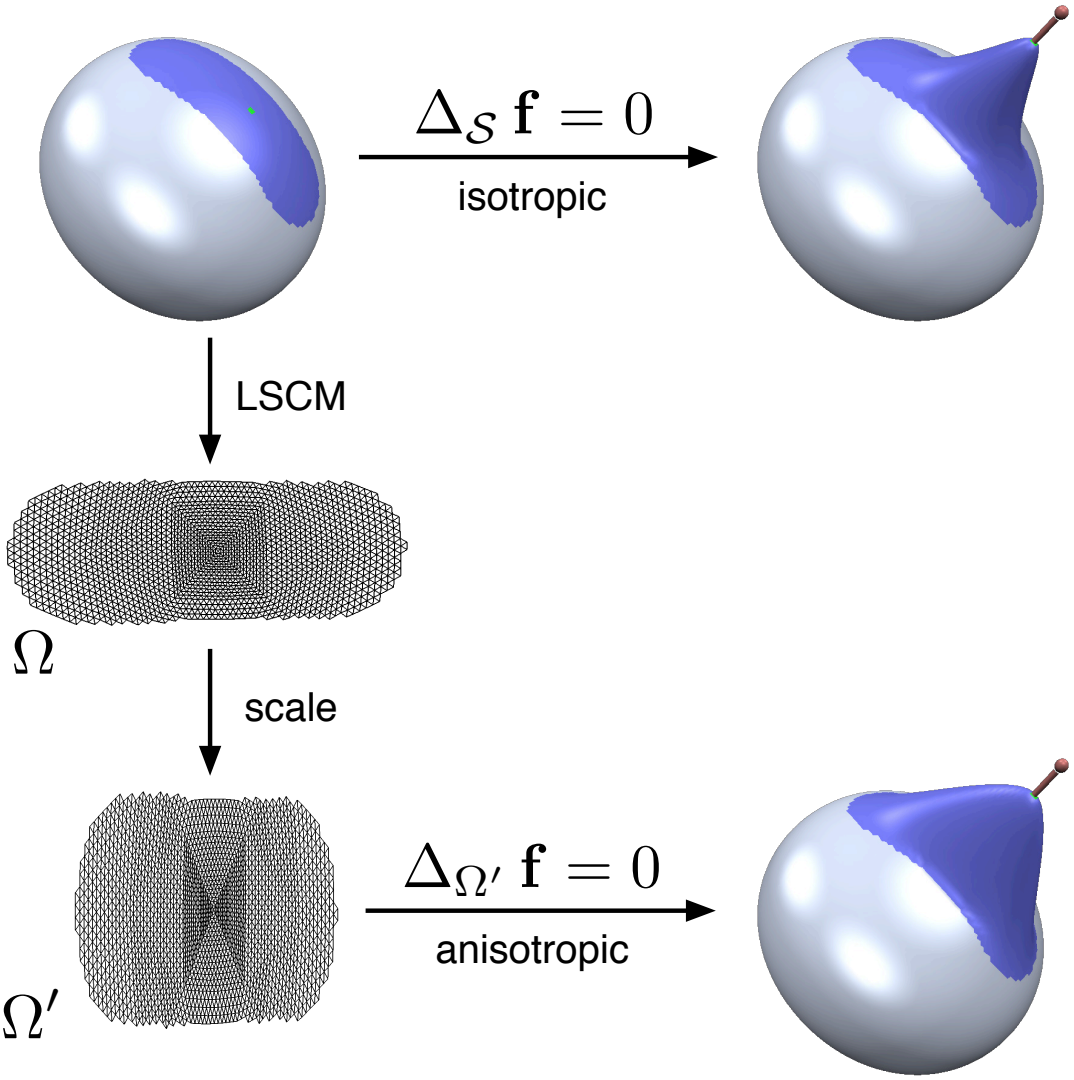


Figure 6.10: The isotropic Laplace-Beltrami operator $\Delta_{\mathcal{S}}$ leads to isotropic deformations even for anisotropically shaped support regions (*top row*). Starting from a conformal parameterization $\Omega \rightarrow \mathcal{S}$, a scaled version is computed to have an isotropic support region $\Omega' \subset \mathbb{R}^2$ (*left column*). The special parameterizations $\Omega' \rightarrow \mathcal{S}$ factor out the anisotropy, such that the isotropic Laplace w.r.t. the domain Ω' results in the more natural anisotropic bending behavior (*bottom row*).

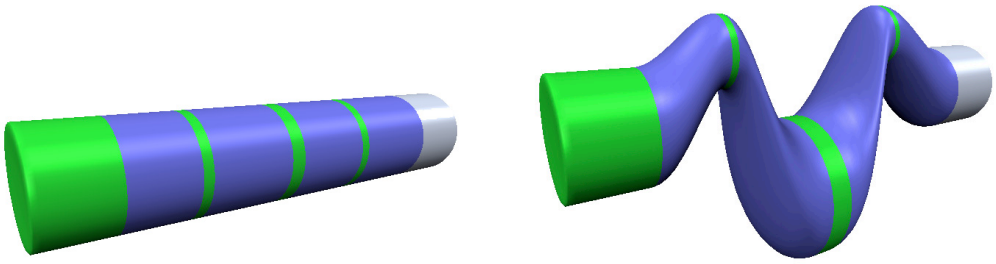


Figure 6.11: Non-disk shaped *anisotropic* deformation are possible using multiple handles. The left cap (*green/light*) is defined as a handle component that is not to be moved. By this the union of support and handle regions is a topological disk.

Notice that this parameterization restricts the shape of the support region to topological disks. However, it is possible to generate modifications with more general support regions by adding "dummy" handle regions that are not to be moved, such that the union of support and handle regions is eventually homeomorphic to a disk (cf. Fig. 6.11). Another possibility would be not to scale the two-dimensional domain Ω, but to use a pre-scaled version of $\mathcal{S}$ as parameter domain, in which the support region has uniform isotropic extend, leading to anisotropic deformation w.r.t. Euclidean distance in 3D. However, if the support is not sufficiently flat (e.g., at edges or corners), the results are less intuitive compared to the *geodesically* anisotropic deformation proposed above. Notice that arbitrary complex support regions can be used if we restrict to the isotropic Laplacian, because this completely avoids the parametrization step.

6.2.5 Precomputed Basis Functions

The freeform modeling technique described so far requires to solve the linear system (6.7) for the free vertex positions P whenever the designer moves the manipulator (and hence the handle region). Even if a highly efficient solver is used for this task (see Chap. 7), the frame rate will not be sufficiently high, especially when surfaces of minimum curvature variation are computed and the number of vertices in the support region is on the order of 10^4 or higher.

By pre-computing a special set of basis functions, that directly correspond to the degrees of freedom of the manipulator, the per-frame computational costs can be reduced significantly. If we denote by L the matrix of Eq. (6.7), the solution of this system can be

expressed explicitly in terms of the inverse matrix L^{-1}. Hence, the set of basis functions is represented by (a combination of) the column vectors of L^{-1}. The explicit solution of (6.7) is

$$\begin{pmatrix} P \\ F \\ H \end{pmatrix} = L^{-1} \begin{pmatrix} 0 \\ F \\ H \end{pmatrix} = L^{-1} \begin{pmatrix} 0 \\ F \\ 0 \end{pmatrix} + L^{-1} \begin{pmatrix} 0 \\ 0 \\ H \end{pmatrix} , \tag{6.8}$$

where the first term on the right hand side is constant, since the fixed vertices F are constant, and the second term depends on the vertices in the handle region. In a similar manner, James and Pai [JP99] as well as Debrun et al. [DMA02] also used pre-computed basis functions based on the columns of inverse matrices in order to accelerate the solution of boundary constrained problems. However, for a complex deformation with a large number of handle vertices $m = |H|$, this pre-computation amounts to solving the linear system m times, which is computationally far too expensive.

But notice that in our case, due to the custom-tailored basis functions, we can restrict to simple user interactions, i.e., to simple transformations of the control handle region. During interactive shape editing, a 9-DoF manipulator is used, which provides an intuitive interface to control an affine map $\mathbf{t} : \mathbb{R}^3 \rightarrow \mathbb{R}^3$, that is applied to the vertices H of the handle region. We can define a local affine coordinate system, spanned by four affinely independent points $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, and $\mathbf{d}$, that is attached to the manipulator in its initial state. Then there exists a matrix $Q \in \mathbb{R}^{m \times 4}$ ($m = |H|$) containing the handle points' affine coordinates w.r.t. the affine frame $(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) \in \mathbb{R}^{3 \times 4}$:

$$H = Q \, (\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})^T \quad .$$

Due to affine invariance, the transformed handle points $\mathbf{t}(H)$ can also be expressed by applying the affine map $\mathbf{t}$ to the affine frame, i.e.,

$$\mathbf{t}(H) = \mathbf{t}\left(Q \, (\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})^T\right) = Q \, (\mathbf{t}(\mathbf{a}), \mathbf{t}(\mathbf{b}), \mathbf{t}(\mathbf{c}), \mathbf{t}(\mathbf{d}))^T \quad .$$

As a consequence we can rewrite (6.8) as

$$\begin{pmatrix} P \\ F \\ H \end{pmatrix} = L^{-1} \begin{pmatrix} 0 \\ F \\ 0 \end{pmatrix} + L^{-1} \begin{pmatrix} 0 \\ 0 \\ Q \end{pmatrix} (\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})^T \quad .$$

This means that we only have to solve the system (6.7) in a pre-processing step for 7 different right hand sides: the three columns of $(0, F, 0)^T$ and the four columns of

$(0,0,Q)^T$. This results in a pre-computed constant term $C := L^{-1}\,(0,F,0)^T$ and a "basis function" matrix $B := L^{-1}\,(0,0,Q)^T$. Whenever the designer moves the manipulator, the affine frame is (affinely) transformed to $(\mathbf{a}',\mathbf{b}',\mathbf{c}',\mathbf{d}')^T$, and the solution of the linear system (6.7) can simply be computed by

$$\begin{pmatrix} P' \\ F \\ H \end{pmatrix} \;=\; C + B\;(\mathbf{a}',\mathbf{b}',\mathbf{c}',\mathbf{d}')^T \quad , \tag{6.9}$$

which can be done in real-time even for highly complex meshes.

Writing Eq. (6.9) in terms of surface updates and removing the constraints from the notation leads to

$$P' \;=\; P + B\;\underbrace{(\delta\mathbf{a},\delta\mathbf{b},\delta\mathbf{c},\delta\mathbf{d})^T}_{=:\delta\mathbf{C}}.$$

Comparing this to Eq. (6.1), we see that B is exactly the matrix representation of the abstract basis function for the deformation. Its four columns correspond to the four points of the affine frame, i.e., to the degrees of freedom of the affine handle transformation T.

As mentioned in Sect. 6.2.2, multiple independent handle regions can be used by splitting H into several components $H_1,\ldots,H_n$, such that each handle can be transformed by its own manipulator widget. In this case, a separate basis function B_i is computed for each handle region H_i, such that the pre-computation requires to solve the linear system (6.7) $3+4n$ times. The solution $\mathcal{S}^*$ is then computed accordingly by evaluating the basis functions using their corresponding (modified) affine frames:

$$P' \;=\; C + \sum_{i=1}^{n} B_i\,(\mathbf{a}_i',\mathbf{b}_i',\mathbf{c}_i',\mathbf{d}_i')^T \quad .$$

6.3 Results

We integrated the presented freeform modeling technique into a multiresolution modeling framework as described in Chap. 5. In addition to the freeform editing, the decomposition operator is also based on BCM, as shown in Fig. 6.12. The multiresolution details are encoded either as normal displacements (high performance) or as displacement volumes (higher quality).

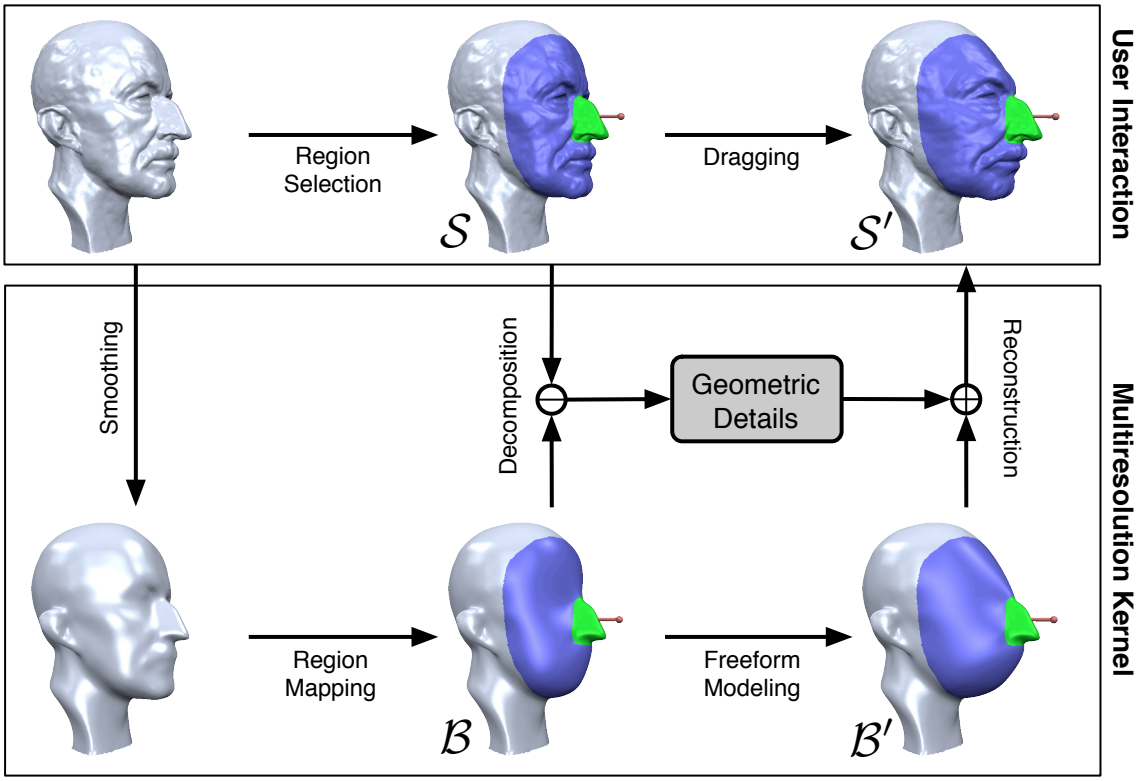


Figure 6.12: Overview of the complete multiresolution modeling framework. In a preprocess the original surface $\mathcal{S}$ is smoothed to get a base surface without high frequencies (*lower left*), which provides smoother and hence more reliable boundary conditions. After the region selection, an energy minimization is used to remove the remaining middle frequencies from the support region on the base surface, yielding the surface $\mathcal{B}$, whose difference to $\mathcal{S}$ is then encoded as geometric details $\mathcal{D} = \mathcal{S} \ominus \mathcal{B}$. Whenever the handle is moved, the support region of the base surface is re-computed by solving Eq. (6.7), i.e., by evaluating the basis function (6.9), yielding the modified base surface $\mathcal{B}'$. The deformed detailed surface $\mathcal{S}'$ is finally reconstructed from $\mathcal{B}'$ and the geometric details: $\mathcal{S}' = \mathcal{B}' \oplus \mathcal{D}$.

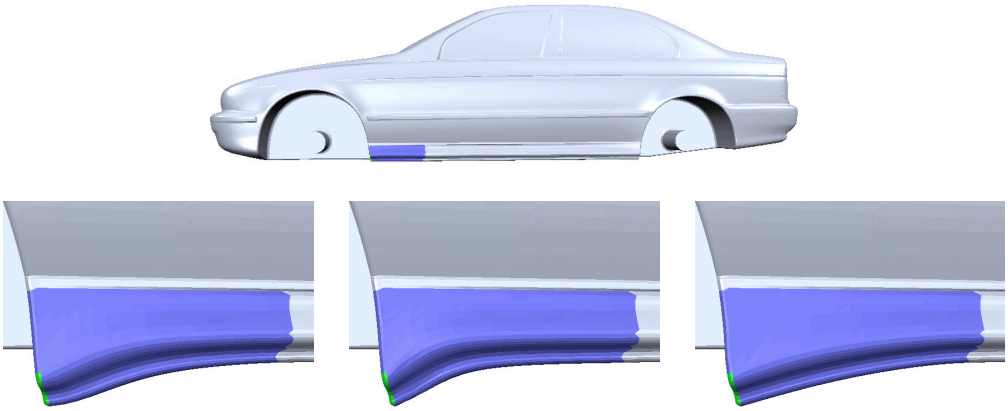


Figure 6.13: Multiresolution deformation of the sillboard using flexible boundary conditions and the anisotropic bending behavior. After lowering the sillboard, the unwanted point of inflection (*bottom left*) is avoided by reducing the continuity constraint at the handle region to C^0. Switching from the isotropic discretization of the Laplace operator (*bottom center*) to the anisotropic one finally leads to the intended natural displacement propagation (*bottom right*).

The actual multiresolution modeling is performed in two steps of interaction. First, the designer defines the basis functions for the deformation by

1. selecting support and handle regions,
2. controlling the stiffness by the choice of the energy functional,
3. specifying the (segment-wise) boundary continuities,
4. and choosing either isotropic or anisotropic bending behavior.

After that, each handle region can be transformed by a manipulator widget, leading to a new base surface $\mathcal{B}'$ and the deformed surface $\mathcal{S}'$ is finally reconstructed by adding the geometric details onto $\mathcal{B}'$.

In an industrial evaluation typical shape deformations in the context of CFD simulation for conceptual car design were successfully tested. One of these deformations is shown in Fig. 6.13, where the sillboard of a car is to be lowered. Exploiting the flexibility provided by continuous boundary smoothness avoids the generation of an unwanted point of inflection along the feature line. In order to achieve a natural displacement propagation over the support region, the anisotropic Laplace discretization is required.

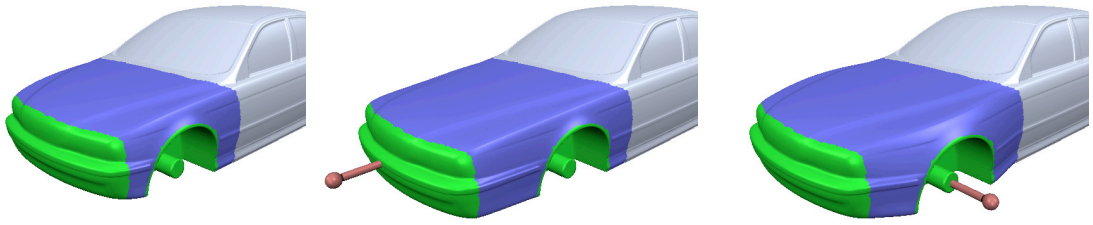


Figure 6.14: Using multiple independent handle components allows to stretch the hood while also rigidly preserving the circular shape of the wheel houses. This kind of deformation would be very difficult using a volumetric deformation technique like freeform deformation.

All fine surface features are well preserved due to the multiresolution decomposition and reconstruction.

Fig. 6.14 shows an example of a complex modification of a car's hood. Multiple independent handle regions are placed at the wheel houses and the grill, thereby allowing to stretch the hood while keeping the wheel houses circular. The active region consists of 35k vertices (support and handles), where the number of support vertices, i.e., the unknowns P of the linear system, is 17k. The pre-processing includes $3 + 3 \cdot 4 = 15$ solutions of Eq. (6.7) (in order to compute the basis functions), the multiresolution decomposition and detail encoding. Using the efficient solvers described in Chap. 7, the total precomputation time was less than 4s on an Intel Pentium4 3GHz. In each frame the base surface $\mathcal{B}'$ is computed by evaluating the three basis function (8ms), the surface $\mathcal{S}'$ is reconstructed by normal displacements (18ms), and the final normal field is computed for surface rendering (11ms). Including the rendering of this 250k triangle model, the surface editing can be done with 13fps.

A possible extension of this modeling metaphor would be to use non-affine handle transformations, e.g., to select a curve on the surface and to deform this handle curve in a non-rigid manner. Although this prevents the use of pre-computed basis functions, the general framework can still be applied, since only the boundary constraints are specified in a different manner. As a consequence, the linear system (6.7) has to be solved each frame, which can be done efficiently using the factorization-based solvers presented in the next chapter. However, although both techniques have linear complexity, the per-frame solution slows down the computation of the BCM surfaces by a factor of about 10, such that the total effective frame rate is halved.

7 Numerical Aspects

The last chapter described the theoretical foundations of our boundary constraint freeform modeling metaphor [BK04a] by introducing the different energy functionals as well as the Euler-Lagrange PDEs characterizing their minimizer surfaces, and by discretizing these PDEs to linear systems of Laplacians, which have to be solved in order to compute the optimal BCM surface. In this chapter we analyze the structure of these Laplacian systems, since they are frequently used in surface smoothing [Tau95, DMSB99], surface parameterization [PP93, DMA02], and surface modeling [KCVS98, LSCO$^+$04, SCOL$^+$04, YZX$^+$04, BK04a].

In the context of our multiresolution modeling framework, we have to solve the linear system of Eq. (6.7) of first, second, or third order Laplacians either in each frame, or several times for the pre-computation of the deformation's basis functions. In both cases, the main goals from a numerical point of view are *robustness* and *efficiency*, such that the surface deformation works flawlessly even on numerically demanding meshes and is sufficiently fast to be used in interactive applications.

We start by discussing numerical robustness issues in Sect. 7.1, and after an analysis of general Laplacian systems (Sect. 7.2) we describe and compare different classes of linear system solvers in Sect. 7.3, since those are responsible for the computational efficiency of our interactive modeling system.

7.1 Robustness

The numerical robustness of computations on discrete triangle meshes is mostly related to the shape of the triangles: Equilateral triangles allow for stable computations, while for degenerate triangles neither area nor derivative information like normal vectors can

be evaluated robustly. For our multiresolution modeling system, this causes the editing operator as well as the reconstruction operator to break down for degenerate meshes.

The failure of the reconstruction operator is obvious, since normal displacements require a well-defined normal field on the base surface to derive the (modified) detailed surface as a displacement in normal direction from the base surface (see Sect. 5.3).

The freeform editing operator has to solve the linear system $\Delta_{\mathcal{S}}^{k} P = B$ for proper boundary constraints B, with k typically ranging from 1 to 3. The matrix $\Delta_{\mathcal{S}}$ contains in each row the cotangent values and Voronoi areas used to discretize the Laplace-Beltrami $\Delta_{\mathcal{S}} \mathbf{f}(v_i)$ of a vector-valued function $\mathbf{f}$, which is in most cases the geometric realization $\mathbf{f}(v_i) = \mathbf{p}(v_i) =: \mathbf{p}_i$ (see Sect. 3.2.1). In the context of conformal parameterizations, this matrix was shown to be non-singular and positive definite as long as no triangle areas are vanishing [PP93]. In the presence of degenerate triangles, however, the matrix is singular, and hence the freeform editing operator cannot compute a solution.

As we described in Chap. 5, and as it is depicted in Fig. 5.3 and Fig. 6.12, two different surfaces are involved in a general multiresolution modeling framework: The detailed surface $\mathcal{S}$ and the base surface $\mathcal{B}$. Notice that these two surfaces play fundamentally different roles in the modeling process. The former is the mesh the designer interacts with, while the latter is generally hidden from the user and is internally used for computing the deformations. As a consequence, the requirements on these two surfaces also differ. The original surface $\mathcal{S}$ has to provide a high-quality approximation to the actual surface geometry and represents its fine details and sharp features by a possibly hand-crafted triangulation. On the other hand, since all numerical computations are performed on the base mesh $\mathcal{B}$ only, its structure is mainly responsible for robustness and efficiency.

Our main observation in [BK04b] is that the tessellations of $\mathcal{S}$ and $\mathcal{B}$ are not restricted to be identical if a suitable representation for the multiresolution details is chosen. As shown in Chap. 5, both normal displacements and displacement volumes do not require the base points $\mathbf{q}_i \in \mathcal{B}$, corresponding to the vertex positions $\mathbf{p}_i \in \mathcal{S}$, to be vertices of the base surface. They can instead be arbitrary surface points on $\mathcal{B}$, i.e., they are also allowed to lie in the interior of triangles.

As a consequence, the triangulation of the base surface is independent of the triangulation of the detailed surface and can therefore be considered as an additional degree of freedom, which can be adjusted in order to improve the robustness of the multiresolution modeling process. Notice that the base surface $\mathcal{B}$ is smooth by construction and

hence contains no high-frequency details. By consequence, a remeshing or resampling of $\mathcal{B}$ can be performed without introducing geometric aliasing artifacts. In contrast, the original surface $\mathcal{S}$ generally does contain sharp features, therefore a naïve remeshing is prohibitive, as it could destroy a feature-aligned triangulation.

Applying the isotropic remeshing technique presented in Sect. 3.2.3 to the base surface results in a highly regular tessellation with close-to-equilateral triangles. Since this effectively removes degenerate triangles, the Laplace matrix of this new tessellation is regular and positive definite. The inner triangle angles are close to 60°, such that the respective cotangent weights are positive, which guarantees stability and convergence. Additionally, the discretization of the Laplace-Beltrami yields a better approximation to the its exact continuous counterpart if the triangulation does not contain obtuse angles [MDSB03].

Hence, isotropic remeshing of the smooth base surface is clearly preferable from a numerical point of view, since it improves the matrix conditioning and increases the overall robustness. The discrete Laplace-Beltrami used for computing the optimal surfaces was demonstrated in Fig. 6.6 not to depend on the triangulation of the surface (in contrast to the uniform Laplace discretization). Hence, the isotropic remeshing can safely be used, since while increasing numerical stability, it will not affect the shape of the resulting surface.

The remeshing also allows for a highly efficient implementation, since the base surface is known to be smooth and therefore no special care has to be taken to preserve sharp features, such that meshes of about 100k triangles can be remeshed in about 5s. In a typical modeling session, the isotropic remeshing has to be performed only once after loading the model and generating the base surface by a pre-smoothing process (cf. Fig. 6.12). It can therefore be considered as an additional pre-processing step, such that the computational overhead caused by it is usually negligible.

7.2 Laplacian Systems

The performance of our modeling system is determined by how fast we can solve the corresponding linear systems, which depends on the kind of solver we use for this task.

As the class of applicable solvers depends on the properties of the corresponding matrix, we take a closer look at the Laplacian matrices $\Delta_{\mathcal{S}}^k$ first.

In each row the matrix $\Delta_{\mathcal{S}}$ contains the weights for the discretization of the Laplace-Beltrami of a function $f : \mathcal{S} \rightarrow \mathbb{R}$ at one vertex v_i (see Sect. 3.2.1):

$$\Delta_{\mathcal{S}} f(v_i) \;=\; \frac{2}{A(v_i)} \sum_{v_j \in N_1(v_i)} (\cot\alpha_{ij} + \cot\beta_{ij}) (f(v_j) - f(v_i)) \quad .$$

Again, f denotes the signal to be smoothed, and in order to compute a smooth surface, it is chosen component-wise to be the geometric realization function $\mathbf{p}(v_i) = (x(v_i), y(v_i), z(v_i))^T$. In matrix notation the vector of the Laplacians of $f(v_i)$ can be written as

$$\begin{pmatrix} \vdots \\ \Delta_{\mathcal{S}} f(v_i) \\ \vdots \end{pmatrix} \;=\; D \cdot M \cdot \begin{pmatrix} \vdots \\ f(v_i) \\ \vdots \end{pmatrix} \quad ,$$

where D is a diagonal matrix containing the normalization factors $D_{ii} = 2/A(v_i)$, and M is a symmetric matrix of cotangent weights

$$M_{ij} \;=\; \begin{cases} 0 & i \neq j \,,\; j \notin N_1(v_i) \\ \cot\alpha_{ij} + \cot\beta_{ij}, & i \neq j \,,\; j \in N_1(v_i) \\ -\sum_{v_j \in N_1(v_i)} (\cot\alpha_{ij} + \cot\beta_{ij}) & i = j \end{cases} \quad .$$

Since the Laplacian of a vertex v_i is defined *locally* in terms of its one-ring neighbors, the matrix M is highly sparse and has non-zeros in the ith row only on the diagonal and in those columns corresponding to v_i's one-ring neighbors $N_1(v_i)$. Due to the Euler characteristic for triangle meshes, this results in about 7 non-zeros per row in average. Analogously, higher order Laplacian matrices $\Delta_{\mathcal{S}}^k$ have non-zeros for the k-ring neighbors $N_k(v_i)$, which are, e.g., about 19 for $k = 2$ and 37 for $k = 3$.

For a closed mesh without boundaries, Laplacian systems $\Delta_{\mathcal{S}}^k P = B$ of any order k can be turned into symmetric ones by moving the first diagonal matrix D to the right-hand side:

$$M(DM)^{k-1} P \;=\; D^{-1} B \quad . \tag{7.1}$$

Boundary constraints are typically employed by restricting the positions of certain vertices C (e.g., the handle and fixed vertices in Sect. 6.2.2), which corresponds to eliminating their respective rows and columns and hence keeps the matrix symmetric. The case

of meshes with boundaries is equivalent to a patch bounded by constrained vertices and therefore also results in a symmetric matrix. Pinkal and Polthier [PP93] additionally showed that this system is positive definite, such that the efficient solvers presented in the next section can be applied.

In order to use the continuous per-vertex boundary continuity introduced in Sect. 6.2.3, the scaling matrices Λ_j with $(\Lambda_j)_{ii} = \lambda_j(v_i)$ have to be incorporated, such that the linear system (7.1) changes to

$$M(\Lambda_{k-1}DM)\cdots(\Lambda_1 DM)P \;=\; D^{-1}B \;\;.$$

Notice that even if the scaling factors $\lambda_j(v_i) \neq 1$ for the constrained vertices $v_i \in C$ only and if all constraints C are moved to the right-hand side, the above matrix is only symmetric if all Λ_j are equal. This is always the case for thin-plate surfaces ($k = 2$), but for minimum variation surfaces ($k = 3$) it is only true for C^0 and C^2 boundary continuity, since then both scaling factors $\lambda_1(v_i) = \lambda_2(v_i)$ are either 0 or 1, respectively. Scaling values which are equal, but different from 0 or 1, have no geometric meaning and are therefore useless.

As a consequence, we can mainly concentrate on symmetric positive definite Laplacian systems in the following discussion, but we also have to consider the non-symmetric case for the segment-wise boundary continuities.

7.3 Linear System Solvers

In this section we describe different types of solvers for sparse linear systems. Within this class of systems, we will further concentrate on symmetric positive definite (so-called *spd*) matrices, like for instance the Laplacian systems analyzed in the last section, since exploiting their special structure allows for the most efficient and most robust implementations. However, the general case of a non-symmetric indefinite system is outlined afterwards in Sect. 7.3.5.

Following our discussion from [BBK05], we propose the use of direct solvers for sparse spd systems, since their superior efficiency — although well known in the field of high performance computing — is often neglected in geometry processing applications. After reviewing the commonly known and used direct and iterative solvers, we introduce sparse direct solvers and point out their advantages.

An important point to be considered is whether the linear systems are solved just once or several times, e.g., for different right hand sides. Since most problems are separable w.r.t. the coordinate components, they can be solved component-wise for x, y, and z using the same system matrix. Multiple right-hand side problems also naturally occur in applications where the user interactively changes boundary constraints, e.g., in surface editing. Notice that there is another situation for solving a sequence of similar systems: when decomposing a non-linear problem into a sequence of linear systems, the values of the matrix entries usually change in each iteration, but its structure, i.e., the pattern of non-zero elements $\{(i,j) \,|\, A_{ij} \neq 0\}$, stays the same. In both cases — solving for multiple right hand sides or matrices of identical structure — this additional information should be exploited as much as possible, e.g., by investing pre-computation time in some kind of factorization or preconditioning.

For the following discussion we restrict ourselves to sparse spd problems $A\mathbf{x} = \mathbf{b}$, with $A = A^T \in \mathbb{R}^{n\times n}$, $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$, and denote by $\mathbf{x}^*$ the exact solution $A^{-1}\mathbf{b}$. The general case of non-symmetric indefinite systems is then outlined in Sect. 7.3.5.

7.3.1 Dense Direct Solvers

Direct linear system solvers are based on a factorization of the matrix A into matrices of simpler structure, e.g., triangular, diagonal, or orthogonal matrices. This structure allows for an efficient solution of the factorized system. As a consequence, once the factorization is computed, it can be used to solve the linear system for several different right hand sides.

The most commonly used examples for *general* matrices A are, in the order of increasing numerical robustness and computational effort, the LU factorization, QR factorization, or the singular value decomposition. However, in the special case of a spd matrix the Cholesky factorization $A = LL^T$, with L denoting a lower triangular matrix, should be employed, since it exploits the symmetry of the matrix and can additionally be shown to be numerically very robust due to the positive definiteness of the matrix A [GL89b].

On the downside, the asymptotic time complexity of all dense direct methods is $O(n^3)$ for the factorization and $O(n^2)$ for solving the system based on the pre-computed factorization. Since for the problems we are targeting at, n can be of the order of 10^5, the

total cubic complexity of dense direct methods is prohibitive. Even if the matrix A is highly sparse, the naïve direct methods enumerated here are not designed to exploit this structure, hence the factors are dense matrices in general (cf. Fig. 7.2, top row).

7.3.2 Iterative Solvers

In contrast to dense direct solvers, iterative methods are able to exploit the sparsity of the matrix A. Since they additionally allow for a simple implementation [PFTV92], iterative solvers are the de-facto standard method for solving sparse linear systems in the context of geometric problems. A detailed overview of iterative methods with valuable implementation hints can be found in [BBC+94].

Iterative methods compute a converging sequence $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(i)}$ of approximations to the solution $\mathbf{x}^*$ of the linear system, i.e., $\lim_{i\to\infty} \mathbf{x}^{(i)} = \mathbf{x}^*$. In practice, however, one has to find a suitable criterion to stop the iteration if the current solution $\mathbf{x}^{(i)}$ is accurate enough, i.e., if the norm of the error $\mathbf{e}^{(i)} := \mathbf{x}^* - \mathbf{x}^{(i)}$ is less than some ε. Since the solution $\mathbf{x}^*$ is not known beforehand, the error has to be estimated by considering the residual $\mathbf{r}^{(i)} := A\mathbf{x}^{(i)} - \mathbf{b}$. These two are related by the *residual equations* $A\mathbf{e}^{(i)} = \mathbf{r}^{(i)}$, leading to an upper bound $\left\|\mathbf{e}^{(i)}\right\| \leq \|A^{-1}\| \cdot \left\|\mathbf{r}^{(i)}\right\|$, i.e., the norm of the inverse matrix has to be estimated or approximated in some way (see [BBC+94]).

The simplest examples for iterative solvers are the Jacobi and Gauss-Seidel methods. They belong to the class of static iterative methods, whose update steps can be written as $\mathbf{x}^{(i+1)} = M\mathbf{x}^{(i)} + \mathbf{c}$ with constant M and $\mathbf{c}$, such that the solution $\mathbf{x}^*$ is the fixed point of this iteration. An analysis of the eigenstructure of the update matrices M reveals that both methods rapidly remove the high frequencies of the error, but the iteration stalls if the error is a smooth function. By consequence, the convergence to the exact solution $\mathbf{x}^*$ is usually too slow in practice. As an additional drawback these methods only converge for a restricted set of matrices, e.g., for diagonally dominant ones.

Non-stationary iterative solvers are more powerful, and for spd matrices the method of conjugate gradients (CG) [HS52, GL89b] is suited best, since it provides guaranteed convergence with monotonically decreasing error. For a spd matrix A the solution of $A\mathbf{x} = \mathbf{b}$ is equivalent to the minimization of the quadratic form

$$\phi(\mathbf{x}) := \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x} \ .$$

The CG method successively minimizes this functional along a set of linearly independent search directions $\mathbf{p}^{(i)}$, such that

$$\mathbf{x}^{(i)} = \operatorname{argmin}\left\{\phi(\mathbf{x}) \,\middle|\, \mathbf{x} \in \mathbf{x}_0 + \operatorname{span}\left\{\mathbf{p}^{(1)}, \ldots, \mathbf{p}^{(i)}\right\}\right\} \quad .$$

Due to the nestedness of these spaces the error decreases monotonically, and the exact solution $\mathbf{x}^* \in \mathbb{R}^n$ is found after at most n steps (neglecting rounding errors). Minimizing ϕ by gradient descent results in inefficient zigzag paths in steep valleys of ϕ, which correspond to strongly differing eigenvalues of A. In order to cancel out the effect of A's eigenvalues on the search directions $\mathbf{p}_i$, those are chosen to be *A-conjugate*, i.e., orthogonal w.r.t. the scalar product induced by A: $\mathbf{p}_j^T A \mathbf{p}_i = 0$ for $i \neq j$ [She94]. The computation of and minimization along these optimal search directions can be performed efficiently and with a constant memory consumption.

The complexity of each CG iteration is mainly determined by the matrix-vector product $A\mathbf{x}$, which is of order $O(n)$ if the matrix is sparse. Given the maximum number of n iterations, the total complexity is $O(n^2)$ in the worst case, but it is usually better in practice.

As the convergence rate mainly depends on the spectral properties of the matrix A, a proper pre-conditioning scheme should be used to increase the efficiency and robustness of the iterative scheme. This means that a slightly different system $\tilde{A}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}$ is solved instead, with $\tilde{A} = PAP^T$, $\tilde{\mathbf{x}} = P^{-T}\mathbf{x}$, $\tilde{\mathbf{b}} = P\mathbf{b}$, using a regular pre-conditioning matrix P, that is chosen such that $\tilde{A}$ is well conditioned [GL89b, BBC+94]. However, the matrix P is restricted to have a simple structure, since an additional linear system $P\mathbf{z} = \mathbf{r}$ has to be solved each iteration.

The iterative conjugate gradients method manages to decrease the computational complexity from $O(n^3)$ to $O(n^2)$ for sparse matrices. However, this is still too slow to compute exact (or sufficiently accurate) solutions of large linear systems, in particular if the systems are numerically ill-conditioned, like for instance the higher order Laplacian systems used in variational surface modeling [KCVS98, BK04a].

7.3.3 Multigrid Iterative Solvers

As mentioned in the last section, one characteristic problem of most iterative solvers is that they are *smoothers*: they attenuate the high frequencies of the error $\mathbf{e}^{(i)}$ very

fast, but their convergence stalls if the error is a smooth function. This fact is exploited by multigrid methods, that build a fine-to-coarse hierarchy $\{\mathcal{M} = \mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_k\}$ of the computation domain $\mathcal{M}$ and solve the linear system hierarchically from coarse to fine [Hac86, BHM00].

After a few (pre-)smoothing iterations on the finest level $\mathcal{M}_0$ the high frequencies of the error are removed and the solver becomes inefficient. However, the remaining low frequency error $\mathbf{e}_0 = \mathbf{x}^* - \mathbf{x}_0$ on $\mathcal{M}_0$ corresponds to higher frequencies when restricted to the coarser level $\mathcal{M}_1$ and therefore can be removed efficiently on $\mathcal{M}_1$. Hence the error is solved for using the residual equations $A\mathbf{e}_1 = \mathbf{r}_1$ on $\mathcal{M}_1$, where $\mathbf{r}_1 = R_{0\to1}\mathbf{r}_0$ is the residual on $\mathcal{M}_0$ transfered to $\mathcal{M}_1$ by a restriction operator $R_{0\to1}$. The result is prolongated back to $\mathcal{M}_0$ by $\mathbf{e}_0 \leftarrow P_{1\to0}\mathbf{e}_1$ and used to correct the current approximation: $\mathbf{x}_0 \leftarrow \mathbf{x}_0 + \mathbf{e}_0$. Small high-frequency errors due to the prolongation are finally removed by a few post-smoothing steps on $\mathcal{M}_0$. The recursive application of this two-level approach to the whole hierarchy can be written as

$$\Phi_i \;=\; S_\mu \, P_{i+1\to i} \, \Phi_{i+1} \, R_{i\to i+1} \, S_\lambda \;\;,$$

with λ and μ pre- and post-smoothing iterations, respectively. One recursive run is known as a *V-cycle* iteration.

Another concept is the method of *nested iterations*, that exploits the fact that iterative solvers are very efficient if the starting value is sufficiently close to the actual solution. One starts by computing the exact solution on the coarsest level $\mathcal{M}_k$, which can be done efficiently since the system $A_k\mathbf{x}_k = \mathbf{b}_k$ corresponding to the restriction to $\mathcal{M}_k$ is small. The prolongated solution $P_{k\to k-1}\mathbf{x}_k^*$ is then used as starting value for iterations on $\mathcal{M}_{k-1}$, and this process is repeated until the finest level $\mathcal{M}_0$ is reached and the solution $\mathbf{x}_0^* = \mathbf{x}^*$ is computed.

The remaining question is how to iteratively solve on each level. The standard method is to use one or two V-cycle iterations, leading to the so-called *full multigrid* method. However, one can also use an iterative smoothing solver (e.g., Jacobi or CG) on each level and completely avoid V-cycles. In the latter case the number of iterations m_i on level i must not be constant, but instead has to be chosen as $m_i = m\,\gamma^i$ to decrease exponentially from coarse to fine [BD96]. Besides the easier implementation, the advantage of this *cascading multigrid* method is that once a level is computed, it is not involved in further computations and can be discarded. A comparison of the three methods in terms of visited multigrid levels is given in Fig. 7.1.

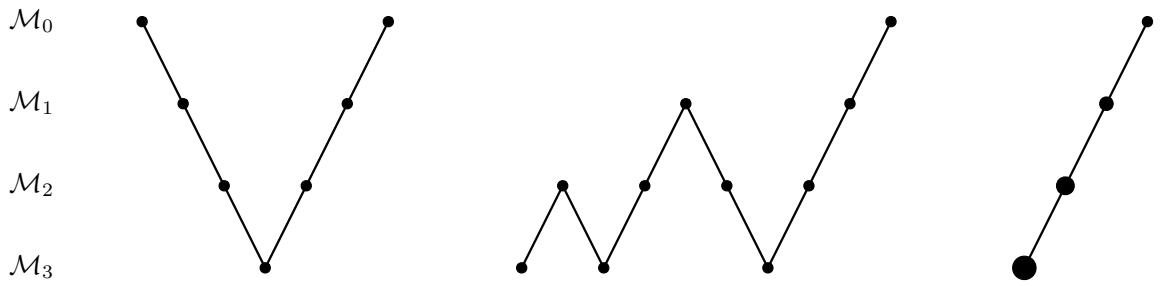


Figure 7.1: A schematic comparison in terms of visited multigrid levels for V-cycle (*left*), full multigrid with one V-cycle per level (*center*), and cascading multigrid (*right*).

Due to the logarithmic number of hierarchy levels $k = O(\log n)$ the full multigrid method and the cascading multigrid method can both be shown to have linear asymptotic complexity, as opposed to quadratic for non-hierarchical iterative methods. However, they cannot exploit synergy for multiple right hand sides, which is why factorization-based approaches are clearly preferable in such situations, as we will show in the next section.

Since in our case the discrete computational domain $\mathcal{M}$ is an irregular triangle mesh instead of a regular 2D or 3D grid, the coarsening operator for building the hierarchy is based on mesh decimation techniques [KCS98]. The shape of the resulting triangles is important for numerical robustness, and the edge lengths on the different levels should mimic the case of regular grids. Therefore the decimation usually removes edges in the order of increasing lengths, such that the hierarchy levels have uniform edge lengths and triangles of bounded aspect ratio.

The simplification from one hierarchy level $\mathcal{M}_i$ to the next coarser one $\mathcal{M}_{i+1}$ should additionally be restricted to remove a *maximally independent set* of vertices, i.e., no two removed vertices $v_j, v_l \in \mathcal{M}_i \setminus \mathcal{M}_{i+1}$ are connected by an edge $e_{jl} \in \mathcal{M}_i$. In [AKS05b] some more efficient alternatives to this standard Dobkin-Kirkpatric hierarchy are described. In order to achieve higher performance, we do not change the simple way the hierarchy is constructed, but instead solve the linear system on every second or third level only, and use the prolongation operator alone on all in-between levels.

The linear complexity of multi-grid methods allows for the highly efficient solution even of very complex systems. However, the main problem of these solvers is their quite involved implementation, since special care has to be taken for the hierarchy building, for special multigrid pre-conditioners, and for the inter-level conversion by restriction and prolongation operators. A detailed overview of these techniques is given in [AKS05b].

Additionally, the number of iterations per hierarchy level have to be chosen: This includes the number of V-cycles and pre- and post-smoothing iterations per V-cycle for the full multigrid method, or m and γ for the cascading multigrid approach. These numbers have to be chosen either by heuristic or experience, since they not only depend on the problem (structure of A), but also on its specific instance (values of A). Nevertheless, if iterative solvers are to be used, multigrid methods are the only way to achieve acceptable running times when solving large systems, as has been shown in [KCVS98, RL03, AKS05b].

7.3.4 Sparse Direct Solvers

The use of direct solvers for large sparse linear systems is often neglected, since naïve direct methods have complexity $O(n^3)$, as described above. The problem is that even when the matrix A is sparse, the factorization will not preserve this sparsity, such that the resulting Cholesky factor is a dense lower triangular matrix.

However, an analysis of the factorization process reveals that a *band-limitation* of the matrix A will be preserved. Following [GL81], we define the bandwidth $\beta(A)$ in terms of the bandwidth of its ith row

$$\beta(A) := \max_{1 \le i \le n} \{\beta_i(A)\} \quad \text{with} \quad \beta_i(A) := i - \min_{1 \le j \le i} \{j \mid A_{ij} \neq 0\} \ .$$

If the matrix $A = LL^T$ has bandwidth $\beta(A)$ then so has its factor L. An even stricter bound is that also the so-called *envelope* or *profile*

$$\mathrm{Env}(A) := \{(i,j) \mid 0 < i - j \le \beta_i(A)\}$$

is preserved, i.e., no additional non-zeros (so-called *fill-in* elements) are generated outside the envelope.

This additional structure can be exploited in both the factorization and the solution process, such that their complexities reduce from $O(n^3)$ and $O(n^2)$ to linear complexity in the number of non-zeros $\text{nz}(A)$ of A [GL81]. Since usually $\text{nz}(A) = O(n)$, this is the same linear complexity as for multigrid solvers. However, in the graphics-related examples we will show in Sect. 7.3.7, sparse direct methods turned out to be more efficient compared to multigrid solvers, in particular for multiple right-hand side problems.

Since we assume the matrices to be sparse, but not band-limited or profile-optimized, the first step is to minimize the matrix envelope, which can be achieved by symmetric row and column permutations $A \leftarrow P^T AP$ using a permutation matrix P, i.e., a re-ordering of the mesh vertices. Although this problem is NP complete, several good heuristics exist, of which we will present the most commonly used in the following. All of these methods work on the undirected *adjacency graph* $\text{Adj}(A)$ corresponding to the non-zeros of A, i.e., two nodes $i, j \in \{1, \ldots, n\}$ are connected by an edge if and only if $A_{ij} \neq 0$.

The standard method for envelope minimization is the *Cuthill-McKee* algorithm [CM69], that picks a start node and renumbers all its neighbors by traversing the adjacency graph in a breadth-first manner, using a greedy selection in order of increasing valence. It has further been proven in [LS76] that reverting this permutation leads to better re-orderings, such that usually the *reverse Cuthill-McKee* method (RCMK) is employed. The result $P^T AP$ of this matrix re-ordering is depicted in the second row of Fig. 7.2.

Since no special pivoting is required for the Cholesky factorization, the non-zero structure of its matrix factor L can symbolically be derived from the non-zero structure of the matrix A alone, or, equivalently, from its adjacency graph. The graph interpretation of the Cholesky factorization is to successively eliminate the node with the lowest index from the graph and connect all its immediate neighbors to each other. The additional edges e_{ij} generated in this so-called *elimination graph* correspond to the fill-in elements $L_{ij} \neq 0 = A_{ij}$.

In order to minimize fill-in the strategy of the *minimum degree* algorithm (MD) and its variants [GL89a, Liu85] is to remove the nodes with smallest valence first from the elimination graph, since this causes the least number of additional pairwise connections. Many efficiency optimizations of this method exist, the most prominent of which is the *super-nodal* approach: instead of removing eliminated nodes from the graph, neighboring eliminated nodes are clustered to so-called super-nodes, allowing for more efficient graph

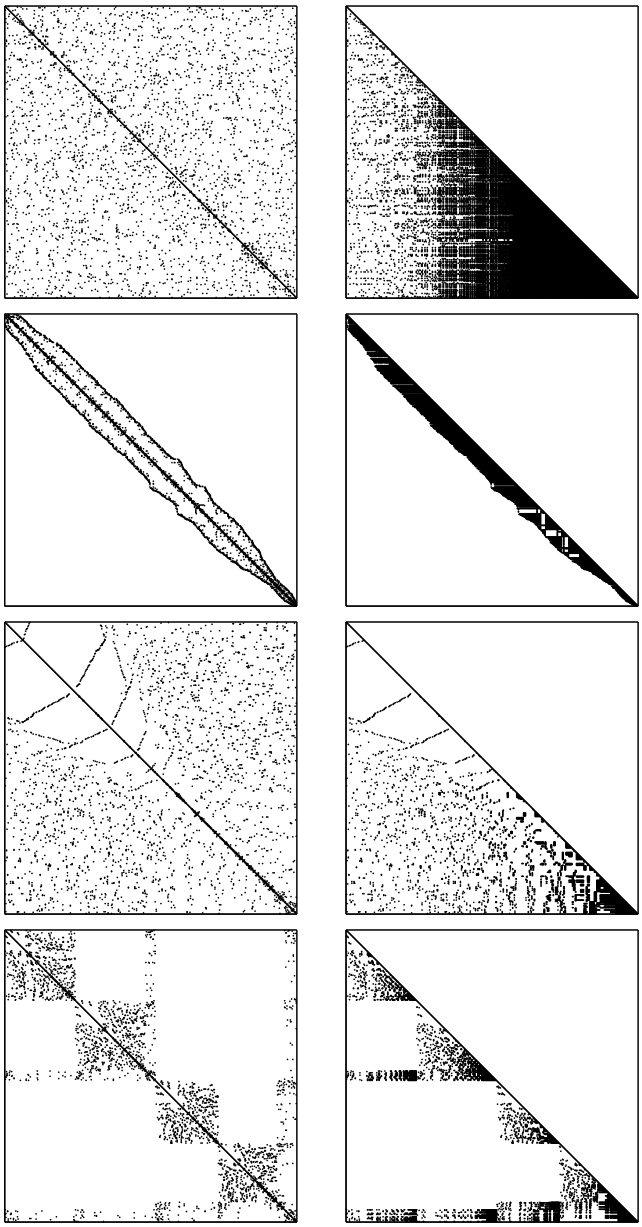


Figure 7.2: The top row shows the non-zero pattern of a typical 500×500 matrix A and its Cholesky factor L, corresponding to a Laplacian system on a triangle mesh. Although A is highly sparse (3502 non-zeros), the factor L is dense (36k non-zeros). The reverse Cuthill-McKee algorithm minimizes the envelope of the matrix, resulting in 14k non-zeros of L (*2nd row*). The minimum degree ordering avoids fill-in during the factorization, which decreases the number of non-zeros to 6203 (*3rd row*). The last row shows the result of a nested dissection method (7142 non-zeros), that allows for parallelization due to its block structure.

updates. The resulting minimum degree re-orderings do not lead to some kind of a band-structure (which implicitly limits fill-in), but instead directly minimize the fill-in of L (cf. Fig. 7.2, third row).

The last class of re-ordering approaches is based on graph partitioning. Consider a matrix A whose adjacency graph has m separate connected components. Such a matrix can be restructured to a block-diagonal matrix of m blocks, such that the factorization can be performed on each block individually. If the adjacency graph is connected, a small subset S of nodes, whose elimination would separate the graph into two components of roughly equal size, is found by one of several heuristics [KK98]. This graph-partitioning results in a matrix consisting of two large diagonal blocks (two connected components) and $|S|$ rows representing their connection (separator S). Recursively repeating this process leads to the method of *nested dissection* (ND), resulting in matrices of the typical block structure shown in the bottom row of Fig. 7.2. Besides the obvious fill-in reduction, these systems also allow for easy parallelization of both the factorization and the solution.

For the comparison of the different matrix re-ordering strategies a rather small matrix was used in Fig. 7.2 to allow for clearer visualization. On an analogous $5k \times 5k$ matrix the number of non-zeros $\mathrm{nz}(L)$ decreases from 2.3M to 451k, 106k, and 104k by applying the RCMK, MD, and ND method, respectively. The timings to obtain those re-orderings are 17ms, 12ms, and 38ms. It can further be observed that for larger systems the nested dissection method [KK98] generally leads to the best results.

One important advantage of the Cholesky factorization is that the non-zero structure of the factor L can be determined from $\mathrm{Adj}(A)$ without any numerical computations. This allows us to setup of an efficient *static* data structure for L *before* the actual *numerical factorization*, which is therefore called *symbolic factorization*. Since suitable data structures and proper memory layout are crucial for efficient numerical computations, this two-step factorization process allows for significant optimizations.

Analogously to the dense direct solvers, the factorization can be exploited to solve for different right hand sides in a very efficient manner. In addition to this, whenever the matrix A is changed, such that its non-zero structure $\mathrm{Adj}(A)$ is preserved, then the matrix re-ordering as well as the symbolic factorization can obviously be re-used. Solving the modified system therefore only requires to re-compute the numerical factorization and performing the back-substitution, which typically saves about 50% of the total

computation time for solving the modified system. As we will show in Sect. 7.3.7, this allows for an efficient implementation of a large class of algorithms that decompose a non-linear problem into a sequence of similar linear ones, like for instance the implicit fairing approach [DMSB99] or the Levenberg-Marquardt optimization for non-linear problems [PFTV92, GMW81].

Another advantage of sparse direct methods is that no additional parameters have to be chosen in a problem-dependent manner, as for instance the different numbers of iterations for the multigrid solvers. The only degree of freedom is the matrix re-ordering, but this only depends on the symbolic structure of the problem and therefore can be chosen quite easily. For more details and implementation notes the reader is referred to the book of George and Liu [GL81]; a highly efficient implementation is publicly available in the TAUCS library [TCR03].

7.3.5 Non-Symmetric Indefinite Systems

When the assumptions about the symmetry and positive definiteness of the matrix A are not satisfied, optimal methods like the Cholesky factorization or conjugate gradients cannot be used. In this section we shortly outline which techniques are applicable instead.

From the class of iterative solvers the bi-conjugate gradients algorithm (BiCG) is typically used as a replacement of the conjugate gradients method [PFTV92]. Although working well in most cases, BiCG does not provide any theoretical convergence guarantees and has a very irregular non-monotonically decreasing residual error for ill-conditioned systems. On the other hand, the GMRES method converges monotonically with guarantees, but its computational cost and memory consumption increase in each iteration [GL89b]. As a good trade-off, the stabilized Bi-CGSTAB [BBC$^+$94] represents a mixture between the efficient BiCG and the smoothly converging GMRES; it provides a much smoother convergence and is reasonably efficient and easy to implement.

When considering dense direct solvers, the Cholesky factorization cannot be used for general matrices. Therefore the LU factorization is typically employed (instead of QR or SVD), since it is similarly efficient and also extends well to sparse direct methods. However, (partial) row and column pivoting is essential for the numerical robustness of

the LU factorization, since this avoids zeros on the diagonal during the factorization process.

Similarly to the Cholesky factorization, it can be shown that the LU factorization also preserves the band-width and envelope of the matrix A. Techniques like the minimum degree algorithm generalize to non-symmetric matrices as well. But as for dense matrices, the banded LU factorization relies on partial pivoting in order to guarantee numerical stability. In this case, two competing types of permutations are involved: symbolic permutations for matrix re-ordering and pivoting permutations ensuring numerical robustness. As these permutations cannot be handled separately, a trade-off between stability and fill-in minimization has to be found, resulting in a significantly more complex factorization process.

As a consequence, the re-ordering depends on the numerical values of the matrix entires, such that an exact symbolic factorization like in the Cholesky case is not possible. In order to nevertheless be able to setup a static data structure, a more conservative envelope is typically used, such that pivoting within this structure is still possible. A highly efficient implementation of a sparse LU factorization is provided by the SuperLU library [DEG+99].

7.3.6 Comparison

In the following we compare the different kinds of linear system solvers for Laplacian as well as for bi-Laplacian systems. All timings reported in this and the next section were taken on a 3.0GHz Pentium4 running Linux. The iterative solver (CG) from the `gmm++` library [RP05] is based on the conjugate gradients method and uses an incomplete LDL^T factorization as preconditioner. Our cascading multigrid solver (MG) performs preconditioned conjugate gradient iterations on each hierarchy level and additionally exploits SSE instructions in order to solve for up to four right-hand sides simultaneously. The direct solver (LL^T) of the TAUCS library [TCR03] employs nested dissection re-ordering and a sparse complete Cholesky factorization. Although our linear systems are symmetric, we also compare to the popular SuperLU solver [DEG+99], which is based on a sparse LU factorization, for the sake of completeness.

Iterative solvers have the advantage over direct ones that the computation can be stopped as soon as a sufficiently small error is reached, which — in typical computer

graphics applications — does not have to be the highest possible precision. In contrast, direct methods always compute the exact solution up to numerical round-off errors, which in our application examples actually was more precise than required. The stopping criteria of the iterative methods have therefore been chosen to yield sufficient results, such that their quality is comparable to that achieved by direct solvers. The resulting residual errors were allowed to be about one order of magnitude larger than those of the direct solvers. While the latter achieved an average residual error of 10^{-7} and 10^{-5} for Laplacian and bi-Laplacian systems, respectively, the iterative solvers were stopped at an error of 10^{-6} and 10^{-4}.

Table 7.1 shows timings for the different solvers on Laplacian systems $\Delta_{\mathcal{S}} P = B$ of 10k to 50k and 100k to 500k unknowns, i.e., free vertices P. For each solver three columns of timings are given:

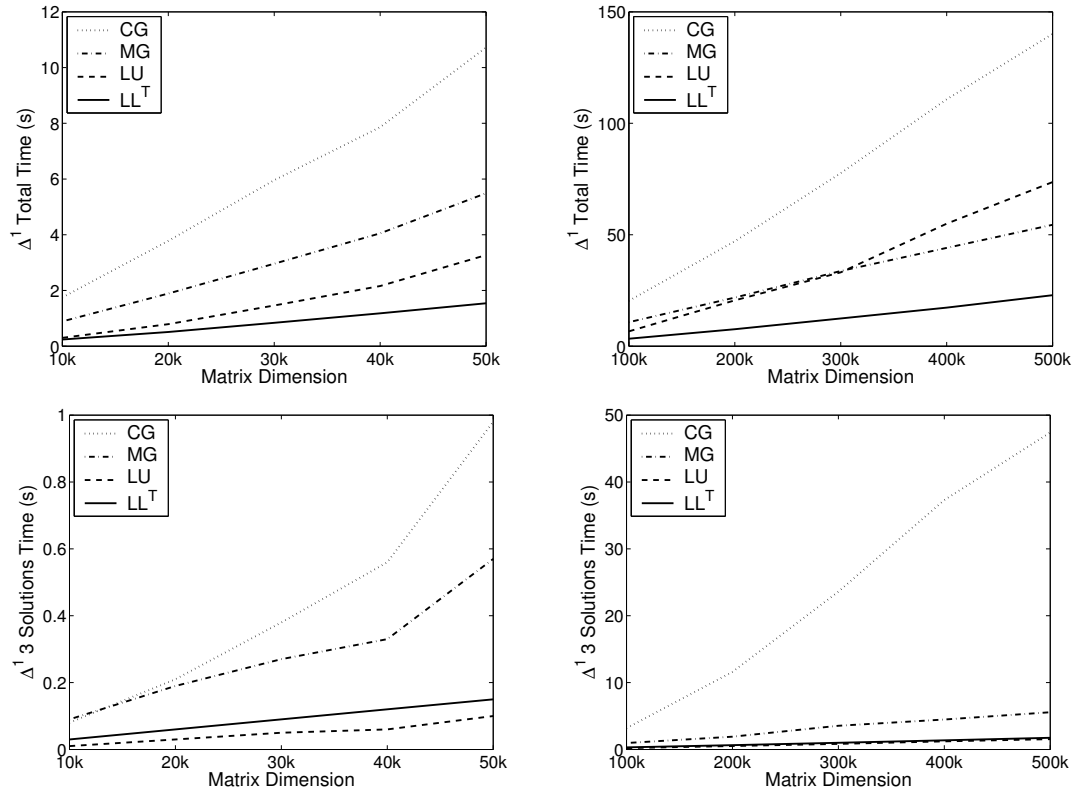
Setup: Computing the cotangent weights for the Laplace discretization and building the matrix structure (done per-level for the multigrid solver).

Precomputation: Preconditioning (*iterative*), building the hierarchy by mesh decimation (*multigrid*), matrix re-ordering and sparse factorization (*direct*).

Solution: Solving the linear system for three different right-hand sides corresponding to the x, y, and z components of the free vertices P.

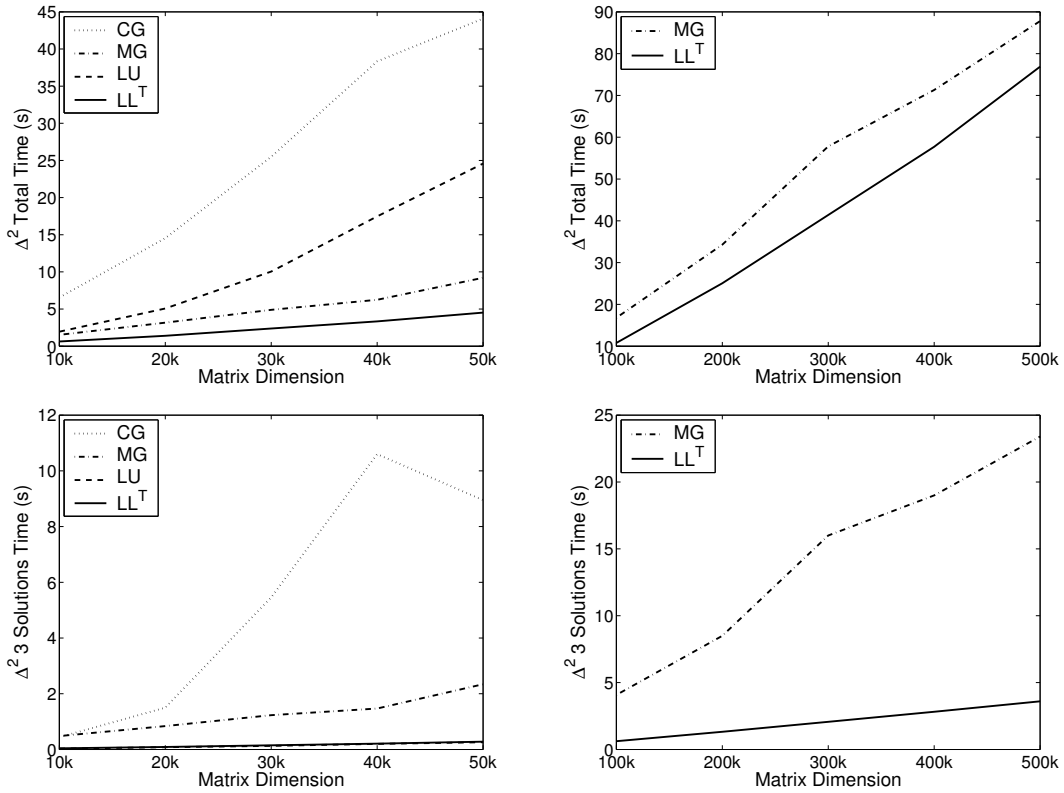
Due to its effective preconditioner, which computes a sparse incomplete factorization, the iterative solver scales almost linearly with the system complexity. However, for large and thus ill-conditioned systems it breaks down. Notice that without preconditioning the solver would not converge for the larger systems.

The experiments clearly verify the linear complexity of multigrid and sparse direct solvers. Once their sparse factorizations are pre-computed, the computational costs for actually solving the system are about the same for the LU and Cholesky solver. However, they differ significantly in the factorization performance, because the numerically more robust Cholesky factorization allows for more optimizations, whereas pivoting is required for the LU factorization to guarantee robustness. This is the reason for the breakdown of the LU solver, such that the multigrid solver is more efficient in terms of total computation time for the larger systems.



Size	Iterative	Multigrid	LU	Cholesky
10k	0.11/1.56/0.08	0.15/0.65/0.09	0.07/0.22/0.01	0.07/0.14/0.03
20k	0.21/3.36/0.21	0.32/1.38/0.19	0.14/0.62/0.03	0.14/0.31/0.06
30k	0.32/5.26/0.38	0.49/2.20/0.27	0.22/1.19/0.05	0.22/0.53/0.09
40k	0.44/6.86/0.56	0.65/3.07/0.33	0.30/1.80/0.06	0.31/0.75/0.12
50k	0.56/9.18/0.98	0.92/4.00/0.57	0.38/2.79/0.10	0.39/1.00/0.15
100k	1.15/16.0/3.19	1.73/8.10/0.96	0.79/5.66/0.21	0.80/2.26/0.31
200k	2.27/33.2/11.6	3.50/16.4/1.91	1.56/18.5/0.52	1.59/5.38/0.65
300k	3.36/50.7/23.6	5.60/24.6/3.54	2.29/30.0/0.83	2.35/9.10/1.00
400k	4.35/69.1/37.3	7.13/32.5/4.48	2.97/50.8/1.21	3.02/12.9/1.37
500k	5.42/87.3/47.4	8.70/40.2/5.57	3.69/68.4/1.54	3.74/17.4/1.74

Table 7.1: Comparison of different solvers for Laplacian systems $\Delta_S P = B$ of 10k to 50k and 100k to 500k free vertices P. The three timings for each solver represent matrix setup, pre-computation, and three solutions for the x, y, and z components of P. The graphs in the upper row show the total computation times (sum of all three columns). The center row depicts the solution times only (3rd column), as those typically determine the per-frame cost in interactive applications.



Size	Iterative	Multigrid	LU	Cholesky
10k	0.33/5.78/0.44	0.40/0.65/0.48	0.24/1.68/0.03	0.24/0.35/0.04
20k	0.64/12.4/1.50	0.96/1.37/0.84	0.49/4.50/0.08	0.49/0.82/0.09
30k	1.04/19.0/5.46	1.40/2.26/1.23	0.77/9.15/0.13	0.78/1.45/0.15
40k	1.43/26.3/10.6	1.69/3.08/1.47	1.07/16.2/0.20	1.08/2.05/0.21
50k	1.84/33.3/8.95	2.82/4.05/2.34	1.42/22.9/0.26	1.42/2.82/0.28
100k	—	4.60/8.13/4.08	2.86/92.8/0.73	2.88/7.29/0.62
200k	—	9.19/16.6/8.50	—	5.54/18.2/1.32
300k	—	17.0/24.8/16.0	—	8.13/31.2/2.07
400k	—	19.7/32.6/19.0	—	10.4/44.5/2.82
500k	—	24.1/40.3/23.4	—	12.9/60.4/3.60

Table 7.2: Comparison of different solvers for bi-Laplacian systems $\Delta_S^2 P = B$ of 10k to 50k and 100k to 500k free vertices P. The three timings for each solver represent matrix setup, pre-computation, and three solutions for the components of P. The graphs in the upper row again show the total computation times, while the center row depicts the solution times only (3rd column). For the larger systems, the iterative solver and the sparse LU factorization fail to compute a solution.

Interactive applications often require to solve the same linear system for several right-hand sides (e.g. once per frame), which typically reflects the change of boundary constraints due to user interaction. For such problems the solution times, i.e., the third columns of the timings, are more relevant, as they correspond to the per-frame computational costs. Here the precomputation of a sparse factorization pays off and the direct solvers are clearly superior to the multigrid method.

Table 7.2 shows the same experiments for bi-Laplacian systems $\Delta_S^2 X = B$ of the same complexity. In this case, the matrix setup is more complex, the matrix condition number is squared, and the sparsity decreases from 7 to 19 non-zeros per row.

Due to the higher condition number the iterative solver takes much longer and even fails to converge on large systems. In contrast, the multigrid solver converges robustly without numerical problems; notice that constructing the multigrid hierarchy is almost the same as for the Laplacian system (up to one more ring of boundary constraints). The computational costs required for the sparse factorization are proportional to the increased number of non-zeros per row. The LU factorization additionally has to incorporate pivoting for numerical stability and failed for larger systems. In contrast, the Cholesky factorization worked robustly in all our experiments.

If we focus on the solution times for the bi-Laplacian systems and compare them to the Laplacian systems, we observe that the direct solver scales with the sparsity of the matrix, while the number of iterations required for the multigrid solver depends on the (squared) matrix condition. In our experiments it turned out that the performance gap between multigrid and direct methods is even larger for bi-Laplacian systems.

We also analyzed the memory consumption of the multigrid method and the sparse Cholesky solver, although both methods were optimized more for performance than for memory requirements. The memory consumption of the multigrid method is mainly determined by the meshes representing the different hierarchy levels. In contrast, the memory required for the Cholesky factorization depends significantly on the sparsity of the matrix, too. On the 500k example the multigrid method and the direct solver need about 1GB and 600MB for the Laplacian system, and about 1.1GB and 1.2GB for the bi-Laplacian system. Hence, the direct solver would not be capable of factorizing Laplacian systems of higher order on current PCs, while the multigrid method would succeed.

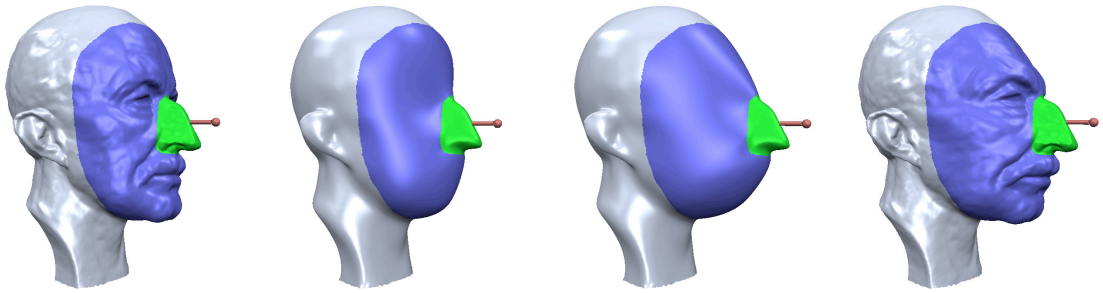


Figure 7.3: Multiresolution modeling allows a low-frequency change of the global shape based on the change of a smooth base surface, that is computed by solving a bi-Laplacian system $\Delta_S^2 P = B$.

These comparisons show that direct solvers are a valuable and efficient alternative to multigrid methods even if the linear systems are highly complex. In all our experiments the sparse Cholesky solver was faster than the multigrid method, and if the system has to be solved for multiple right-hand sides, the precomputation of a sparse factorization is even more beneficial.

7.3.7 Applications

In this section we show several geometry processing applications that benefit from the use of sparse direct solvers. Most applications are based on solving Laplacian or bi-Laplacian systems, thus their characteristic behavior for different complexities or different solvers can be transferred from the experiments of the last section.

Surface Modeling

The first application is our freeform modeling approach (see Chap. 6), which requires to compute (the change of) a smooth base surface by solving higher order Laplacian systems $\Delta_S^k P = B$, $k \in \{2, 3\}$, three times for the x, y, and z coordinates of the unconstrained (*dark/blue*) vertices P (cf. Fig. 7.3). Each time the designer drags some points on the surface, the boundary constraints change and the linear system has to be solved for another right-hand side in order to compute the deformed surface. As a consequence, this approach greatly benefits from the sparse factorization solvers. The

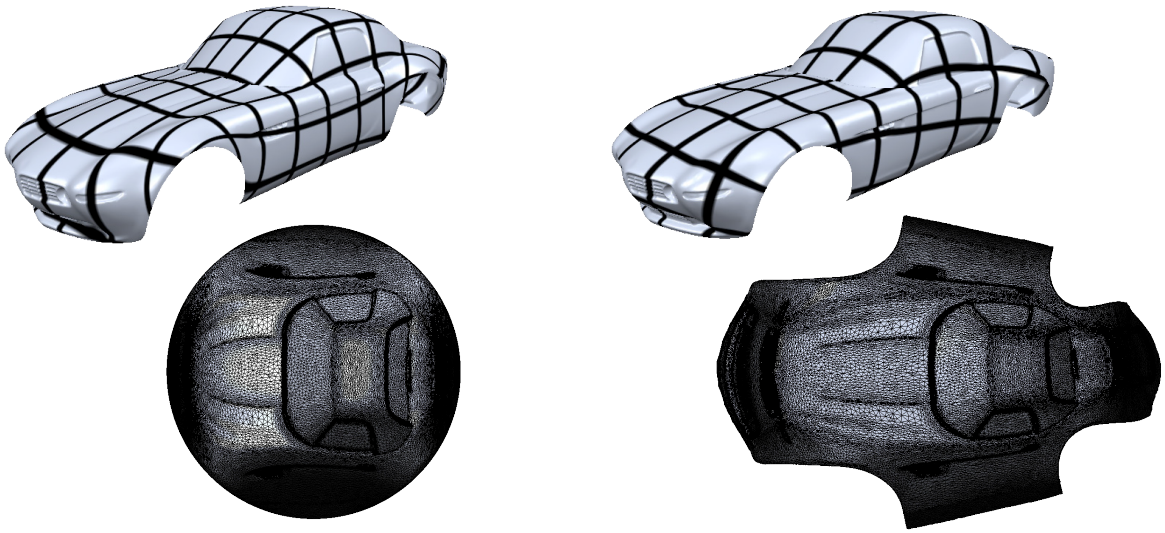


Figure 7.4: Two different parameterizations of a car model: discrete conformal parameterization with fixed boundary (*left*), least squares conformal map with free boundary (*right*). Both parameterizations are computed by solving a sparse spd system for the free 2D parameter values associated to the mesh vertices.

precomputation of basis functions for the deformation (Sect. 6.2.5) also requires to solve the linear system for several right-hand sides, such that this precomputation gets more efficient, too.

Conformal Parameterization

Computing a conformal parameterization [PP93, DMA02] with fixed boundary vertices requires the solution of a Laplacian system $\Delta_{\mathcal{S}} P = B$ for x and y (cf. Fig. 7.4, left). In [AKS05b] a highly elaborate multigrid solver has been derived by evaluating different kinds of multigrid hierarchies and preconditioning strategies. This solver was then used for the parameterization of large meshes, where it takes only 37s for 580k free vertices on a 2.8GHz Pentium4. This time includes loading the system from disk, building the hierarchy, and solving the system for the x coordinate [Aks05a]. Our implementation based on the sparse Cholesky solver takes 28s for for the parameterization of 600k vertices on a 3.0GHz Pentium4, including matrix setup, re-ordering, factorization, and two solutions.

Least Squares Conformal Maps

In the approach of [LPRM02] a conformal parameterization is not computed by minimizing the discrete Dirichlet energy, but instead by solving a system of Cauchy-Riemann equations for each face (cf. Fig. 7.4, right). Since the number of faces F is about twice the number of vertices V, this system is overdetermined and hence solved in the least squares sense using the normal equations, leading to a spd matrix of dimension $2V \times 2V$, which is similar in structure to a Laplacian matrix. Since the iterative solver used in the original paper [LPRM02] was not capable of parameterizing large meshes, the use of multigrid methods was proposed in [RL03]. On an 1.2GHz Pentium4 their hierarchical approach takes 18s, 31s, and 704s for meshes of 18k, 36k, and 560k vertices, respectively. On a comparable machine (Athlon 1.2GHz) the direct sparse solver is about 4–5 times faster; on the 3.0GHz machine these parameterizations can be computed in 1.4s, 3.2s, and 95s, respectively.

Implicit Smoothing

In the implicit fairing approach [DMSB99] meshes are smoothed by an integration of the PDE $\partial \mathbf{x}_i / \partial t = \lambda \Delta_{\mathcal{S}} \mathbf{x}_i$, leading to the so-called *mean curvature flow* (see Sect. 3.2.1). Using semi-implicit integration, this non-linear problem is decomposed into a sequence of linear ones, such that in each time-step the Laplace discretization $\Delta_{P^{(i)}}$ is updated and the Laplacian system $(I - \lambda \Delta_{P^{(i)}})\, P^{(i+1)} = P^{(i)}$ is solved. In this case the matrix reordering and the symbolic factorization can be kept and just the numerical factorization and the solution have to be computed. In our experiments this saved 40%–60% of the solver time per iteration.

Discussion

In this section we discussed and compared different classes of linear system solvers for large sparse symmetric positive matrices, and pointed out that sparse direct solvers are a valuable alternative to the usually employed multigrid methods, since they turned out to be more efficient and easier to use in all our experiments. Although the class of spd matrices seems to be quite restricted, many frequently encountered geometry processing problems lead to this kind of systems or can easily be reformulated in this form. As we demonstrated in our experiments, all these applications benefit considerably from the use of sparse direct solvers.

8 Conclusion

In this thesis we proposed techniques for the generation and optimization of triangle meshes as well as methods for high quality surface deformation. In the following we summarize our main contributions, present the results of an industrial evaluation, and conclude with a discussion of promising future research directions.

The goal of the first part of this work was the approximation of technical datasets by triangle meshes, which should be of sufficiently high quality to be used in numerical simulations. In this context, the quality of a triangle mesh approximation is determined by several factors: First, it has to provide a close approximation of the original surface geometry, since otherwise the results of numerical simulations are meaningless. Second, its mesh topology, i.e., the surface tessellation, has to be carefully controlled and adjusted in order to not contain numerically critical degenerate triangles.

One constraint for successfully approximating technical datasets is the faithful reconstruction of both sharp geometric features and anisotropically curved blend regions. Our feature-sensitive mesh generation and resampling algorithms represent important contributions to this research field and were shown to yield superior results with a minimum amount of geometric aliasing or normal noise.

Our feature-preserving isotropic remeshing approach optimizes the tessellation of a triangle mesh for close-to-equilateral triangles, which allows for numerically stable computations. While our results are comparable to existing high quality isotropic remeshing techniques, the running times are considerably faster, since computationally expensive parameterizations are not required. Guaranteeing an upper bound on the geometric deviation throughout all mesh optimization processes is crucial, and our GPU-accelerated tolerance volumes represent an efficient technique to do so. Since it does not interfere with the geometry processing algorithms, this framework is generally applicable.

The multiresolution surface deformation techniques presented in the second part of this thesis allow for high quality deformations that preserve all important surface details in an intuitive and natural manner. Our multiresolution surface representation based on displacement volumes provides physically more plausible detail reconstructions compared to the standard displacement vector representation. Moreover, it effectively avoids local self-intersections, which is crucial for producing and preserving orientable manifold surfaces.

The "heart" of a multiresolution modeling framework is the editing operator, since it is responsible for the overall flexibility, smoothness, and ease of use. Our boundary constraint modeling approach is based on mathematically well understood concepts of surface optimization and results in deformations which interpolate arbitrary per-vertex displacements (high flexibility) and otherwise exhibit minimal bending energy (high smoothness). Its fine-grained control of boundary continuities and anisotropic surface behavior turned out to be very important in practical evaluations. The proposed precomputation of deformation basis functions finally enables shape editing in real-time even of complex surfaces.

Since many geometry processing problems discussed in this thesis can be formulated as the solution of one or several large sparse linear systems, a key ingredient for efficient algorithms is a fast linear system solver. In a detailed comparison of different classes of solvers we found sparse direct solvers to be preferable for our applications, since they are easy to use and provide high performance. Although these methods are well known in the field of high performance computing, they have been rarely applied to geometry processing problems.

Large parts of this thesis were developed during an industrial cooperation with the CFD department of the BMW group in Munich, which provided us with the valuable opportunity to verify that the presented results are not only of theoretical interest, but also solve relevant "real-world" problems in a reliable and efficient manner. The majority of the proposed algorithms have been combined into a flexible and powerful geometry processing toolkit for repairing, optimizing, and multiresolution editing of triangle meshes.

The geometry deformations typically applied for optimizing a car's aero dynamics are usually performed in a CAD system. However, our toolkit now enables CFD engineers to directly prepare, optimize, and deform the triangle mesh to be used for simulations later

on, which consequently avoids expensive surface conversions. Because of the superior flexibility of unstructured triangle meshes compared to NURBS surfaces, and due to our intuitive and efficient modeling metaphor, the desired deformations, which initially took highly skilled CAD specialists up to several days, can now be done by the CFD engineer in a few hours.

Promising directions for future research can be found in all topics addressed in this thesis. In the context of high quality surface generation, anisotropic remeshing and shape approximation techniques produce meshes that very well capture the structure of the underlying surface geometry [ACSD+03, MK04, CSAD04]. As these approaches are computationally quite involved, it would be interesting to investigate which quality can be achieved by an anisotropic version of our efficient isotropic remeshing technique.

One advantage which can also be considered a problem of the presented freeform modeling technique is that it computes a deformation field *on* the surface. While this provides a fine-grained control of the surface deformation, it also requires a certain minimum tessellation or sampling quality in order to guarantee sufficient robustness, as pointed out in Chap. 7. In contrast to this, space deformation techniques do not depend on the underlying surface representation and hence are not affected by its quality aspects. As these methods can additionally be applied to most explicit geometry representations in a unified manner, a promising approach would be to derive a space deformation method based on the same variational shape optimization principles as our surface-based approach, hence providing the same quality as well as flexibility, but avoiding the above mentioned drawbacks.

Bibliography

[AA03] A. Adamson and M. Alexa. Approximating and intersecting surfaces from points. In *Proc. of Eurographics Symposium on Geometry Processing 03*, pages 245–254, 2003.

[AB03] H. Aanæs and J. A. Bærentzen. Pseudo-normals for signed distance computation. In *Proc. of Vision, Modeling and Visualization 03*, pages 407–413, 2003.

[ABK98] N. Amenta, M. Bern, and M. Kamvysselis. A new Voronoi-based surface reconstruction algorithm. In *Proc. of ACM SIGGRAPH 98*, pages 415–422, 1998.

[ACdVDI03] P. Alliez, É. Colin de Verdière, O. Devillers, and M. Isenburg. Isotropic surface remeshing. In *Proc. of Shape Modeling International 03*, pages 49–58, 2003.

[ACSD+03] P. Alliez, D. Cohen-Steiner, O. Devillers, B. Lévy, and M. Desbrun. Anisotropic polygonal remeshing. In *Proc. of ACM SIGGRAPH 03*, pages 485–493, 2003.

[AK89] J. Arvo and D. Kirk. *An introduction to ray tracing*, chapter A survey of ray tracing acceleration techniques, pages 201–262. Academic Press, 1989.

[Aks05a] B. Aksoylu. Personal communication, 2005.

[AKS05b] B. Aksoylu, A. Khodakovsky, and P. Schröder. Multilevel Solvers for Unstructured Surface Meshes. *SIAM Journal on Scientific Computing*, 26(4):1146–1165, 2005.

[AMD02] P. Alliez, M. Meyer, and M. Desbrun. Interactive geometry remeshing. In *Proc. of ACM SIGGRAPH 02*, pages 347–354, 2002.

[Bat95] K.-J. Bathe. *Finite Element Procedures*. Prentice Hall, 1995.

[BBB+97] J. Bloomental, C. Bajaj, J. Blinn, M. Cani-Gascuel, A. Rockwood, B. Wyvill, and G. Wyvill. *Introduction to implicit surfaces*. Morgan Kaufmann Publishers, 1997.

[BBC+94] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[BBK04] S. Bischoff, M. Botsch, and L. Kobbelt. Freeform shape representations for efficient geometry processing. In *Course Notes of Shape Modeling International*, 2004.

[BBK05] M. Botsch, D. Bommes, and L. Kobbelt. Efficient linear system solvers for geometry processing. In *11th IMA conference on the Mathematics of Surfaces*, 2005.

[BBVK04] M. Botsch, D. Bommes, C. Vogel, and L. Kobbelt. GPU-based tolerance volumes for mesh processing. In *Proc. of Pacific Graphics 04*, 2004.

[BD96] F. A. Bornemann and P. Deuflhard. The cascading multigrid method for elliptic problems. *Num. Math.*, 75(2):135–152, 1996.

[BGH+04] I. Buck, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, D. Luebke, T. J. Purcell, and C. Woolley. GPGPU: General-purpose computation on graphics hardware. In *Course notes of ACM SIGGRAPH 04*, 2004.

[BHM00] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial*. SIAM, 2nd edition, 2000.

[BHZK05] M. Botsch, A. Hornung, M. Zwicker, and L. Kobbelt. High quality surface splatting on today's GPUs. In *Proc. of symposium on Point-Based Graphics 05*, pages 17–24, 2005.

[BK01a] M. Botsch and L. Kobbelt. Resampling feature and blend regions in polygonal meshes for surface anti-aliasing. In *Proc. of Eurographics 01*, pages 402–410, 2001.

[BK01b] M. Botsch and L. Kobbelt. A robust procedure to eliminate degenerate faces from triangle meshes. In *Proc. of Vision, Modeling, and Visualization 01*, pages 283–289, 2001.

[BK03a] G. H. Bendels and R. Klein. Mesh forging: editing of 3D-meshes using implicitly defined occluders. In *Proc. of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 207–217, 2003.

[BK03b] S. Bischoff and L. Kobbelt. Sub-voxel topology control for level-set surfaces. In *Proc. of Eurographics 03*, pages 273–280, 2003.

[BK03c] M. Botsch and L. Kobbelt. High-quality point-based rendering on modern GPUs. In *Proc. of Pacific Graphics 03*, pages 335–343, 2003.

[BK03d] M. Botsch and L. Kobbelt. Multiresolution surface representation based on displacement volumes. In *Proc. of Eurographics 03*, pages 483–491, 2003.

[BK04a] M. Botsch and L. Kobbelt. An intuitive framework for real-time freeform modeling. In *Proc. of ACM SIGGRAPH 04*, pages 630–634, 2004.

[BK04b] M. Botsch and L. Kobbelt. A remeshing approach to multiresolution modeling. In *Proc. of Eurographics symposium on Geometry Processing 04*, pages 189–196, 2004.

[Bot05] M. Botsch. Extended marching cubes implementation. http://www-i8.informatik.rwth-aachen.de/software/software.html, 2002–2005.

[BPK04] S. Bischoff, D. Pavic, and L. Kobbelt. Automatic restoration of polygon models. *Preprint*, 2004.

[BRK00] M. Botsch, C. Rössl, and L. Kobbelt. Feature sensitive sampling for interactive remeshing. In *Proc. of Vision, Modeling and Visualization 00*, pages 129–136, 2000.

[BSBK02] M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt. Openmesh — a generic and efficient polygon mesh data structure. In *Proc. of OpenSG symposium 02*, 2002.

[BSK04] M. Botsch, M. Spernat, and L. Kobbelt. Phong splatting. In *Proc. of symposium on Point-Based Graphics 04*, 2004.

[BSM05] M. Botsch, A. Sovakar, and M. Marinov. OpenMesh implementation. http://www.openmesh.org, 2002–2005.

[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 *Proc. of ACM SIGGRAPH 01*, pages 67–76, 2001.

[CKS98] S. Campagna, L. Kobbelt, and H.-P. Seidel. Directed edges — a scalable representation for triangle meshes. *ACM Journal of Graphics Tools*, 3(4), 1998.

[CL94] B. Curless and M. Levoy. Better optical triangulation through spacetime analysis. In *Proc. of the 5th International Conference on Computer Vision*, page 987, 1994.

[CL96] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Proc. of ACM SIGGRAPH 96*, pages 303–312, 1996.

[CM69] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proc. of the 24th ACM National Conference*, pages 157–172, 1969.

[Coq90] S. Coquillart. Extended free-form deformation: a sculpturing tool for 3D geometric modeling. In *Proc. of ACM SIGGRAPH 90*, pages 187–196, 1990.

[Cox89] H. S. M. Coxeter. *Introduction to Geometry*. Wiley, 2nd edition, 1989.

[CRS98] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.

[CSAD04] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. In *Proc. of ACM SIGGRAPH 04*, pages 905–914, 2004.

[CVM+96] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. P. Brooks, Jr., and W. Wright. Simplification envelopes. In *Proc. of ACM SIGGRAPH 96*, pages 119–128, 1996.

[Dav75] P. Davis. *Interpolation and Approximation*. Dover Publications, 1975.

[dC76] M. P. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, 1976.

[DEG+99] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications*, 20(3):720–755, 1999.

[DMA02] M. Desbrun, M. Meyer, and P. Alliez. Intrinsic parameterizations of surface meshes. In *Proc. of Eurographics 02*, pages 209–218, 2002.

[DMSB99] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proc. of ACM SIGGRAPH 99*, pages 317–324, 1999.

[EDD+95] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *Proc. of ACM SIGGRAPH 95*, pages 173–182, 1995.

[Far97] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, 4th edition, 1997.

[FB88] D. R. Forsey and R. H. Bartels. Hierarchical B-spline refinement. In *Proc. of ACM SIGGRAPH 88*, pages 205–212, 1988.

[FB95] D. Forsey and R. H. Bartels. Surface fitting with hierarchical splines. *ACM Transactions on Graphics*, 14(2):134–161, 1995.

[FPRJ00] S. Frisken, R. Perry, A. Rockwood, and T. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proc. of ACM SIGGRAPH 00*, pages 249–254, 2000.

[Gar99] M. Garland. Multiresolution modeling: Survey & future opportunities. In *Eurographics State of the Art Report 99*, 1999.

[GGH02] X. Gu, S. J. Gortler, and H. Hoppe. Geometry images. In *Proc. of ACM SIGGRAPH 02*, pages 355–361, 2002.

[GH97] M. Garland and P. Heckbert. Surface simplification using quadric error metrics. In *Proc. of ACM SIGGRAPH 97*, pages 209–216, 1997.

[GL81] A. George and J. W. H. Liu. *Computer solution of large sparse positive definite matrices*. Prentice Hall, 1981.

[GL89a] A. George and J. W. H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31(1):1–19, 1989.

[GL89b] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 1989.

[GLW96] G. Greiner, J. Loos, and W. Wesselink. Data dependent thin plate energy and its use in interactive surface modeling. In *Proc. of Eurographics 96*, pages 175–186, 1996.

[GMW81] P. R. Gill, W. Murray, and M.H. Wright. *Practical Optimization*. Academic Press, 1981.

[Gre94] G. Greiner. Variational design and fairing of spline surfaces. In *Proc. of Eurographics 94*, pages 143–154, 1994.

[GSS99] I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. In *Proc. of ACM SIGGRAPH 99*, pages 325–334, 1999.

[GVSS00] I. Guskov, K. Vidimce, W. Sweldens, and P. Schröder. Normal meshes. In *Proc. of ACM SIGGRAPH 00*, pages 95–102, 2000.

[Hac86] W. Hackbusch. *Multi-Grid Methods and Applications*. Springer Verlag, 1986.

[HDD+92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Proc. of ACM SIGGRAPH 92*, pages 71–78, 1992.

[HDD+94] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. In *Proc. of ACM SIGGRAPH 94*, pages 295–302, 1994.

[Hof89] C. M. Hoffmann. *Geometric and solid modeling: An introduction*. Morgan Kaufmann Publishers, 1989.

[HS52] M. Hestenes and E. Stiefel. Method of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Stand.*, 49:409–436, 1952.

[JLSW02] T. Ju, F. Lasasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. In *Proc. of ACM SIGGRAPH 02*, pages 339–346, 2002.

[JP99] D. L. James and D. K. Pai. ArtDefo: accurate real time deformable objects. In *Proc. of ACM SIGGRAPH 99*, pages 65–72, 1999.

[Ju04] T. Ju. Robust repair of polygonal models. In *Proc. of ACM SIGGRAPH 04*, pages 888–895, 2004.

[Kau87] A. Kaufman. Efficient algorithms for 3D scan-conversion of parametric curves, surfaces, and volumes. In *Proc. of ACM SIGGRAPH 87*, pages 171–179, 1987.

[KB89] D. Kalra and A. Barr. Guaranteed ray intersections with implicit surfaces. In *Proc. of ACM SIGGRAPH 89*, pages 297–306, 1989.

[KB00] L. Kobbelt and M. Botsch. An interactive approach to point cloud triangulation. In *Proc. of Eurographics 00*, pages 479–487, 2000.

[KB03a] L. Kobbelt and M. Botsch. Feature sensitive mesh processing. In *Proc. of 19th spring conference on Computer graphics*, pages 17–22, 2003.

[KB03b] L. Kobbelt and M. Botsch. Freeform shape representations for efficient geometry processing. In *Proc. of Shape Modeling International 03*, pages 111–118, 2003.

[KB04] L. Kobbelt and M. Botsch. A survey of point-based techniques in computer graphics. *Computers & Graphics*, 28(6):801–814, 2004.

[KBB+00] L. Kobbelt, S. Bischoff, M. Botsch, K. Kähler, C. Rössl, R. Schneider, and J. Vorsatz. Geometric modeling based on polygonal meshes. In *Eurographics Tutorial Notes 00*, 2000.

[KBS00] L. Kobbelt, T. Bareuther, and H.-P. Seidel. Multiresolution shape deformations for meshes with dynamic vertex connectivity. In *Proc. of Eurographics 00*, pages 249–260, 2000.

[KBSS01] L. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel. Feature sensitive surface extraction from volume data. In *Proc. of ACM SIGGRAPH 01*, pages 57–66, 2001.

[KCS98] L. Kobbelt, S. Campagna, and H.-P. Seidel. A general framework for mesh decimation. In *Proc. of Graphics Interface 98*, pages 43–50, 1998.

[KCVS98] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *Proc. of ACM SIGGRAPH 98*, pages 105–114, 1998.

[Ket98] L. Kettner. Using generic programming for designing a data structure for polyhedral surfaces. In *14th Annual ACM Symp. on Computational Geometry*, 1998.

[KK98] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal of Sci. Comput.*, 20(1):359–392, 1998.

[KLS96] R. Klein, G. Liebich, and W. Straßer. Mesh reduction with error control. In *Proc. of Visualization 96*, pages 311–318, 1996.

[Kob97] L. Kobbelt. Discrete fairing. In *Proc. on 7th IMA Conference on the Mathematics of Surfaces*, pages 101–131, 1997.

[Kob03] L. Kobbelt. Freeform shape representations for efficient geometry processing. Invited Talk at Eurographics 2003, 2003.

[KVLS99] L. Kobbelt, J. Vorsatz, U. Labsik, and H.-P. Seidel. A shrink wrapping approach to remeshing polygonal surfaces. In *Proc. of Eurographics 99*, pages 119–130, 1999.

[KVS99] L. Kobbelt, J. Vorsatz, and H.-P. Seidel. Multiresolution hierarchies on unstructured triangle meshes. *Comput. Geom. Theory Appl.*, 14(1-3):5–24, 1999.

[KWT98] M. Kass, A. Witkin, and D. Terzopoulus. Snakes: active contour models. *International Journal of Computer Vision*, 4(1):321–313, 1998.

[LC87] W. E. Lorensen and H. E. Cline. Marching cubes: a high resolution 3D surface construction algorithm. In *Proc. of ACM SIGGRAPH 87*, pages 163–170, 1987.

[Liu85] J. W. H. Liu. Modification of the minimum-degree algorithm by multiple elimination. *ACM Trans. Math. Softw.*, 11(2):141–153, 1985.

[LKE98] C. Lürig, L. Kobbelt, and T. Ertl. Deformable surfaces for feature based indirect volume rendering. In *Proc. of Computer Graphics International 98*, pages 752–760, 1998.

[LKG+03] I. Llamas, B. Kim, J. Gargus, J. Rossignac, and C. D. Shaw. Twister: a space-warp operator for the two-handed editing of 3D shapes. In *Proc. of ACM SIGGRAPH 03*, pages 663–668, 2003.

[LKM01] E. Lindholm, M. Kilgard, and H. Moreton. A user-programmable vertex engine. In *Proc. of ACM SIGGRAPH 01*, pages 149–158, 2001.

[LMH00] A. Lee, H. Moreton, and H. Hoppe. Displaced subdivision surfaces. In *Proc. of ACM SIGGRAPH 00*, pages 85–94, 2000.

[LPC+00] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital Michelangelo project: 3D scanning of large statues. In *Proc. of ACM SIGGRAPH 00*, pages 131–144, 2000.

[LPRM02] B. Lévy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. In *Proc. of ACM SIGGRAPH 02*, pages 362–371, 2002.

[LS76] J. W. H. Liu and A. H. Sherman. Comparative analysis of the Cuthill-McKee and the reverse Cuthill-McKee ordering algorithms for sparse matrices. *SIAM J. Numerical Analysis*, 2(13):198–213, 1976.

[LSCO+04] Y. Lipman, O. Sorkine, D. Cohen-Or, D. Levin, C. Rössl, and H.-P. Seidel. Differential coordinates for interactive mesh editing. In *Proc. of Shape Modeling International 04*, pages 181–190, 2004.

[LSS+98] A. W. F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin. MAPS: Multiresolution adaptive parameterization of surfaces. In *Proc. of ACM SIGGRAPH 98*, pages 95–104, 1998.

[LTW95] Y. Lee, D. Terzopoulos, and K. Waters. Realistic modeling for facial animation. In *Proc. of ACM SIGGRAPH 95*, pages 55–62, 1995.

[MBWB02] K. Museth, D. E. Breen, R. T. Whitaker, and A. H. Barr. Level set surface editing operators. In *Proc. of ACM SIGGRAPH 02*, pages 330–338, 2002.

[MDM+02] M. Müller, J. Dorsey, L. McMillan, R. Jagnow, and B. Cutler. Stable real-time deformations. In *Proc. of the ACM SIGGRAPH symposium on Computer animation*, pages 49–54, 2002.

[MDSB03] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In Hans-Christian Hege and Konrad Polthier, editors, *Visualization and Mathematics III*, pages 35–57. Springer-Verlag, Heidelberg, 2003.

[MJ96] R. MacCracken and K. I. Joy. Free-form deformations with lattices of arbitrary topology. In *Proc. of ACM SIGGRAPH 95*, pages 181–188, 1996.

[MK04] M. Marinov and L. Kobbelt. Direct anisotropic quad-dominant remeshing. In *Proc. of Pacific Graphics 04*, pages 207–216, 2004.

[MS92] H. P. Moreton and C. H. Séquin. Functional optimization for fair surface design. In *Proc. of ACM SIGGRAPH 92*, pages 167–176, 1992.

[MSS94a] C. Montani, R. Scateni, and R. Scopigno. Discretized marching cubes. In *Proc. of Visualization 94*, pages 281–287, 1994.

[MSS94b] C. Montani, R. Scateni, and R. Scopigno. A modified look-up table for implicit disambiguation of marching cubes. *The Visual Computer*, (10):353–355, 1994.

[NH91] G. Nielson and B. Hamann. The asymptotic decider: resolving the ambiguity in marching cubes. In *Proc. of Visualization 91*, pages 83–91, 1991.

[OBA+03] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level partition of unity implicits. In *Proc. of ACM SIGGRAPH 03*, pages 463–470, 2003.

[OF02] J. S. Osher and R. P. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 2002.

[PBP02] H. Prautzsch, W. Boehm, and M. Paluszny. *Bézier and B-Spline Techniques*. Springer Verlag, 2002.

[PFTV92] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 2nd edition, 1992.

[PKKG03] M. Pauly, R. Keiser, L. Kobbelt, and M. Gross. Shape modeling with point-sampled geometry. In *Proc. of ACM SIGGRAPH 03*, pages 641–650, 2003.

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

[PT97] L. A. Piegl and W. Tiller. *The NURBS Book*. Springer, 2nd edition, 1997.

[RL03] N. Ray and B. Levy. Hierarchical Least Squares Conformal Map. In *Proc. of Pacific Graphics 03*, pages 263–270, 2003.

[RP05] Y. Renard and J. Pommier. `Gmm++`: a generic template matrix C++ library. http://www-gmm.insa-toulouse.fr/getfem/gmm_intro, 2005.

[RS01] C. Rezk-Salama. *Volume Rendering Techniques for General Purpose Graphics Hardware*. PhD thesis, University of Erlangen-Nürnberg, 2001.

[SA03] M. Segal and K. Akeley. The OpenGL Graphics System: A Specification (Version 1.5). http://www.opengl.org, 2003.

[SAG03] V. Surazhsky, P. Alliez, and C. Gotsman. Isotropic remeshing of surfaces: a local parameterization approach. In *Proc. of 12th International Meshing Roundtable*, 2003.

[Sam94] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison–Wesley, 1994.

[Sap94] N. S. Sapidis. *Designing Fair Curves and Surfaces: Shape Quality in Geometric Modeling and Computer-Aided Design*. SIAM, 1994.

[SCF+04] T. W. Sederberg, D. L. Cardon, G. T. Finnigan, N. S. North, J. Zheng, and T. Lyche. T-spline simplification and local refinement. In *Proc. of ACM SIGGRAPH 04*, pages 276–283, 2004.

[SCOL+04] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel. Laplacian surface editing. In *Proc. of Eurographics symposium on Geometry Processing 04*, pages 179–188, 2004.

[Set96] J. Sethian. A fast marching level set method for monotonically advancing fronts. In *Proc. of the National Academy of Science*, volume 93, pages 1591–1595, 1996.

[Set99] J. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, 1999.

[SF98] K. Singh and E. Fiume. Wires: A geometric deformation technique. In *Proc. of ACM SIGGRAPH 98*, pages 405–414, 1998.

[SG03] V. Surazhsky and C. Gotsman. Explicit surface remeshing. In *Proc. of Eurographics/ACM SIGGRAPH symposium on Geometry processing 03*, pages 20–30, 2003.

[She94] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie Mellon University, 1994.

[SJ00] G. Schaufler and H. Wann Jensen. Ray tracing point sampled geometry. In *Proc. of Eurographics Workshop on Rendering Techniques 00*, pages 319–328, 2000.

[SOS04] C. Shen, J. F. O'Brien, and J. R. Shewchuk. Interpolating and approximating implicit surfaces from polygon soup. In *Proc. of ACM SIGGRAPH 04*, pages 896–904, 2004.

[SP86] T. W. Sederberg and S. R. Parry. Free-form deformation of solid geometric models. In *Proc. of ACM SIGGRAPH 86*, pages 151–159, 1986.

[SPS01] S. Schkolne, M. Pruett, and P. Schröder. Surface drawing: creating organic 3D shapes with the hand and tangible tools. In *Proc. of the SIGCHI conference on Human factors in computing systems*, pages 261–268. ACM Press, 2001.

[SZBN03] T. W. Sederberg, J. Zheng, A. Bakenov, and A. Nasri. T-splines and T-NURCCs. In *Proc. of ACM SIGGRAPH 03*, pages 477–484, 2003.

[Tau95] G. Taubin. A signal processing approach to fair surface design. In *Proc. of ACM SIGGRAPH 95*, pages 351–358, 1995.

[TCR03] S. Toledo, D. Chen, and V. Rotkin. Taucs: A library of sparse linear solvers. http://www.tau.ac.il/∼stoledo/taucs, 2003.

[TL94] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *Proc. of ACM SIGGRAPH 94*, pages 311–318, 1994.

[VG96] L. Velho and J. Gomez. Approximate conversion of parametric to implicit surfaces. In *Proc. of Eurographics 96*, pages 327–337, 1996.

[VRS03] J. Vorsatz, C. Rössl, and H.-P. Seidel. Dynamic remeshing and applications. In *Proc. of Solid Modeling and Applications*, pages 167–175, 2003.

[WDSB00] Z. Wood, M. Desbrun, P. Schröder, and D. Breen. Semi-regular mesh extraction from volumes. In *Proc. of Visualization 00*, pages 275–282, 2000.

[WK03] J. Wu and L. Kobbelt. Piecewise linear approximation of signed distance fields. In *Proc. of Vision, Modeling, and Visualization 03*, pages 513–520, 2003.

[WK04] J. Wu and L. Kobbelt. A stream algorithm for the decimation of massive meshes. In *Proc. of Graphics Interface 03*, pages 185–192, 2004.

[WW92] W. Welch and A. Witkin. Variational surface modeling. In *Proc. of ACM SIGGRAPH 92*, pages 157–166, 1992.

[WW94] W. Welch and A. Witkin. Free-form shape design using triangulated surfaces. In *Proc. of ACM SIGGRAPH 94*, pages 247–256, 1994.

[YT02] G. Yngve and G. Turk. Robust creation of implicit surfaces from polygonal meshes. *IEEE Transactions on Visualization and Computer Graphics*, 8(4):346–359, 2002.

[YZX+04] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Mesh editing with Poisson-based gradient field manipulation. In *Proc. of ACM SIGGRAPH 04*, pages 644–651, 2004.

[ZG02] S. Zelinka and M. Garland. Permission grids: Practical, error-bounded simplification. In *ACM Transactions on Graphics*, pages 207–229, 2002.

[ZSD+00] D. Zorin, P. Schröder, T. DeRose, L. Kobbelt, A. Levin, and W. Sweldens. Subdivision for modeling and animation. In *Course notes of ACM SIGGRAPH 00*, 2000.

[ZSS97] D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In *Proc. of ACM SIGGRAPH 97*, pages 259–268, 1997.

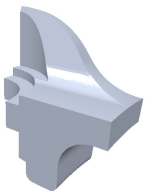
Data Sources

The following list specifies the origins of the models used in this thesis that have not been created by the author nor his colleagues at the RWTH Aachen or his former colleagues at the MPI Saarbrücken:

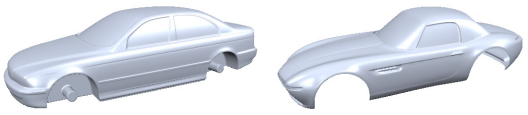
The bunny model is courtesy of the Stanford 3D Scanning Repository, Stanford University, USA.

The David model is courtesy of Marc Levoy, Digital Michelangelo Project, Stanford University, USA.

The Fandisk model is courtesy of Hugues Hoppe, Microsoft Research, USA.



The BMW car models are courtesy of the BMW group, Munich, Germany.



Curriculum Vitae

Personal data

Mario Botsch
Computer Graphics Group
RWTH Aachen, Germany
Phone: ++49-241-8021-817
Email: botsch@cs.rwth-aachen.de

21.01.1974	Born in Bremen, Germany
Oct. 1994 – Apr. 1999	Study of mathematics at the University of Erlangen-Nürnberg. Finished with diploma of mathematics (with honours).
Mai 1999 – Dec. 2000	Ph.D. student at the Max Planck Institute for Computer Science, Saarbrücken, supervised by Dr. Leif Kobbelt and Prof. Dr. Hans-Peter Seidel.
Jan. 2001 – Jul. 2005	Ph.D. student at the Computer Graphics Group, RWTH Aachen, supervised by Prof. Dr. Leif Kobbelt. Finished with degree "Dr. rer. nat." (with honours).

Publications

Diploma Thesis, University of Erlangen-Nürnberg, 1999

Mario Botsch: *3D Gesichtsmodellierung zur Operationsplanung*, supervised by Prof. Dr. Thomas Sauer, Prof. Dr. med. Dr. med. dent. M. Farmand.

Eurographics 2000

Leif Kobbelt, Mario Botsch: *An interactive approach to point cloud triangulation*, pages 479–487.

Vision, Modeling & Visualization 2000

Mario Botsch, Christian Rössl, Leif Kobbelt: *Feature sensitive sampling for interactive remeshing*, pages 129–136.

ACM SIGGRAPH 2001

Leif Kobbelt, Mario Botsch, Ulrich Schwanecke, Hans-Peter Seidel: *Feature sensitive surface extraction from volume data*, pages 57-66.

Eurographics 2001

Mario Botsch, Leif Kobbelt: *Resampling feature and blend regions in polygonal meshes for surface anti-aliasing*, pages 402–410.

Vision, Modeling & Visualization 2001

Mario Botsch, Leif Kobbelt: *A robust procedure to eliminate degenerate faces from triangle meshes*, pages 283–289.

Eurographics Workshop on Rendering 2002

Mario Botsch, Andreas Wiratanaya, Leif Kobbelt: *Efficient high quality rendering of point sampled geometry*, pages 53–64.

OpenSG Symposium 2002

Mario Botsch, Stephan Steinberg, Stephan Bischoff, Leif Kobbelt: *OpenMesh — A generic and efficient polygon mesh data structure.*

Eurographics 2003

Mario Botsch, Leif Kobbelt: *Multiresolution surface representations based on displacement volumes*, pages 483–491.

Pacific Graphics 2003

Mario Botsch, Leif Kobbelt: *High-quality point-based rendering on modern GPUs*, pages 335–343.

Spring Conference on Computer Graphics 2003

Leif Kobbelt, Mario Botsch: *Feature sensitive mesh processing*, pages 17–22.

Shape Modeling International 2003

Leif Kobbelt, Mario Botsch: *Freeform shape representations for efficient geometry processing*, pages 111–118.

ACM SIGGRAPH 2004

Mario Botsch, Leif Kobbelt: *An intuitive framework for real-time freeform modeling*, pages 630–634.

Symposium Geometry Processing 2004

Mario Botsch, Leif Kobbelt: *A remeshing approach to multiresolution modeling*, pages 189–196.

Graphics Interface 2004

Matthias Zwicker, Jussi Räsänen, Mario Botsch, Carsten Dachsbacher, Mark Pauly: *Perspective accurate splatting*, pages 247–254.

Symposium on Point-Based Graphics 2004

Mario Botsch, Michael Spernat, Leif Kobbelt: *Phong splatting*, pages 25–32.

Computer & Graphics 2004, Vol. 28, No. 6

Leif Kobbelt, Mario Botsch: *A survey of point-based techniques in computer graphics*, pages 801–814.

Pacific Graphics 2004

Mario Botsch, David Bommes, Christoph Vogel, Leif Kobbelt: *GPU-based tolerance volumes for mesh processing*, pages 237–243.

Symposium on Point-Based Graphics 2005

Mario Botsch, Alexander Hornung, Matthias Zwicker, Leif Kobbelt: *High quality surface splatting on today's GPUs*, pages 17–24.

IMA conference on Mathematics of Surfaces 2005

Mario Botsch, David Bommes, Leif Kobbelt: *Efficient linear system solvers for geometry processing.*

Eurographics 2005

Mario Botsch, Leif Kobbelt: *Real-time shape editing using radial basis functions.*

Tutorials

Eurographics 2000

Leif Kobbelt, Stephan Bischoff, Mario Botsch, Kolja Kähler, Christian Rössl, Robert Schneider, Jens Vorsatz: *Geometric modeling based on polygonal meshes.*

Shape Modeling International 2004

Stephan Bischoff, Mario Botsch, Leif Kobbelt: *Freeform shape representations for efficient geometry processing.*