Geometric Modeling Based on Triangle Meshes

Mario Botsch[1] Mark Pauly[1] Christian Rössl[2] Stephan Bischoff[3] Leif Kobbelt[3]

[1]ETH Zurich
[2]INRIA Sophia Antipolis
[3]RWTH Aachen University of Technology

Contents

1. Introduction

In the last years triangle meshes have become increasingly popular and are nowadays intensively used in many different areas of computer graphics and geometry processing. In classical CAGD irregular triangle meshes developed into a valuable alternative to traditional spline surfaces, since their conceptual simplicity allows for more flexible and highly efficient processing.

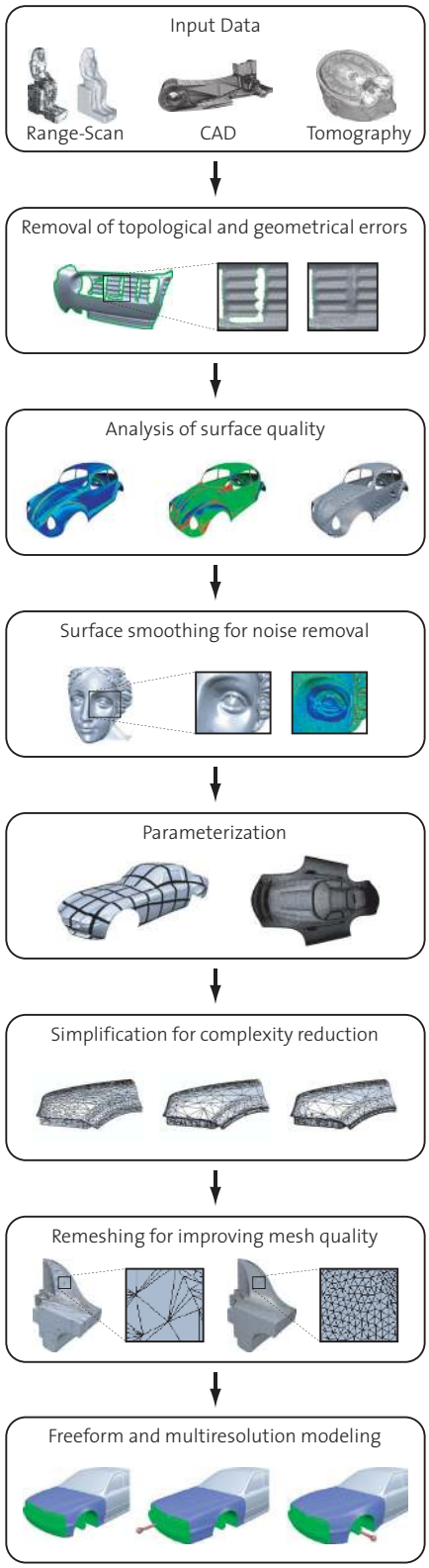
Moreover, the consequent use of triangle meshes as surface representation avoids error-prone conversions, e.g., from CAD surfaces to mesh-based input data of numerical simulations. Besides classical geometric modeling, other major areas frequently employing triangle meshes are computer games and movie production. In this context geometric models are often acquired by 3D scanning techniques and have to undergo post-processing and shape optimization techniques before being actually used in production.

This course discusses the whole geometry processing pipeline based on triangle meshes. We will first introduce general concepts of surface representations and point out the advantageous properties of triangle meshes in Section 2, and present efficient data structures for their implementation in Section 3.

The different sources of input data and types of geometric and topological degeneracies and inconsistencies are described in Section 4, as well as techniques for their removal, resulting in clean two-manifold meshes suitable for further processing. Mesh quality criteria measuring geometric smoothness and element shape together with the corresponding analysis techniques are presented in Section 6.

Mesh smoothing reduces noise in scanned surfaces by generalizing signal processing techniques to irregular triangle meshes (Section 7). Similarly, the underlying concepts from differential geometry are useful for surface parametrization as well (Section 8). Due to the enormous complexity of meshes acquired by 3D scanning, mesh decimation techniques are required for error-controlled simplification (Section 9). The shape of triangles, which is important for the robustness of numerical simulations, can be optimized by general remeshing methods (Section 10).

After optimizing meshes with respect to the different quality criteria, we finally present techniques for intuitive and interactive shape deformation (Section 11). Since solving linear systems is a commonly required component for many of the presented mesh processing algorithms, we will discuss their efficient solution and compare several existing libraries in Section 12.



2. Surface Representations

The efficient processing of different kinds of geometric objects requires — analogously to other fields of computer science — the design of suitable data structures. Since 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* surface representations and *implicit* surface representations.

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 $\mathcal{S} = \mathbf{f}(\Omega) \subset \mathbb{R}^3$. 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., $\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^3 \mid F(\mathbf{x}) = 0\}$. A simple two-dimensional example is the unit circle, which can be described as the range of the explicit function

$$\mathbf{f} : [0, 2\pi] \to \mathbb{R}^2, \quad t \mapsto \begin{pmatrix} \cos(t) \\ \sin(t) \end{pmatrix}$$

as well as by the kernel of the implicit function

$$F : \mathbb{R}^2 \to \mathbb{R}, \quad (x, y) \mapsto \sqrt{x^2 + y^2} - 1 \ .$$

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, one can classify them into the following three categories [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 a surface.

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 the following 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 (Section 2.3).

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 $\mathcal{S}$ (Section 8). When changing the shape of $\mathcal{S}$, 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 are the weak points of explicit surfaces.

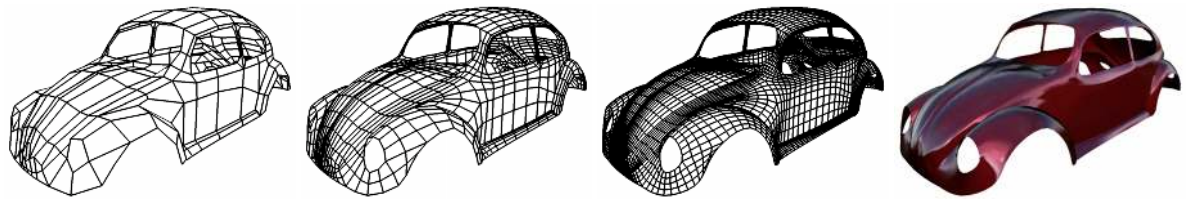


Figure 1: *Subdivision surfaces are generated by an iterative refinement of a coarse control mesh.*

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 surfaces from scratch as well as for later surface deformation tasks. Spline surfaces can conveniently be described by the B-spline basis functions $N_i^n(\cdot)$, for more detail see [Far97, PT97, PBP02].

A tensor product spline surface $\mathbf{f}$ of degree n is a piecewise polynomial surface that is built by connecting several polynomial patches in a smooth C^{n-1} manner:

$$\begin{aligned} \mathbf{f} : [0,1]^2 &\rightarrow \mathbb{R}^3 \\ (u,v) &\mapsto \sum_{i=0}^{m} \sum_{j=0}^{m} \mathbf{c}_{ij} N_i^n(u) N_j^n(v) \quad . \end{aligned}$$

The *control points* $\mathbf{c}_{ij} \in \mathbb{R}^3$ define the so-called *control grid* of the spline surface. Because $N_i^n(u) \geq 0$ and $\sum_i N_i^n \equiv 1$, each surface point $\mathbf{f}(u,v)$ is a convex combination of the control points $\mathbf{c}_{ij}$, i.e., the surface lies within the convex hull of the control grid. Due to the small support of the basis functions, each control point has local influence only. These two properties cause spline surfaces to closely follow the control grid, thereby providing a geometrically intuitive metaphor for modeling surfaces by adjusting its control points.

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 decomposed into a large number of (possibly trimmed) tensor-product patches.

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 significantly complicate surface construction and in particular the later surface modeling tasks.

2.1.2. Subdivision Surfaces

Subdivision surfaces [ZSD*00] can be considered as a generalization of spline surfaces, since they are also controlled by a coarse *control mesh*, but in contrast to spline surfaces allow to represent surfaces of arbitrary topology. Subdivision surfaces are generated by 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. 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., surface meshes whose triangulation is 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, KVLS99a, 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.

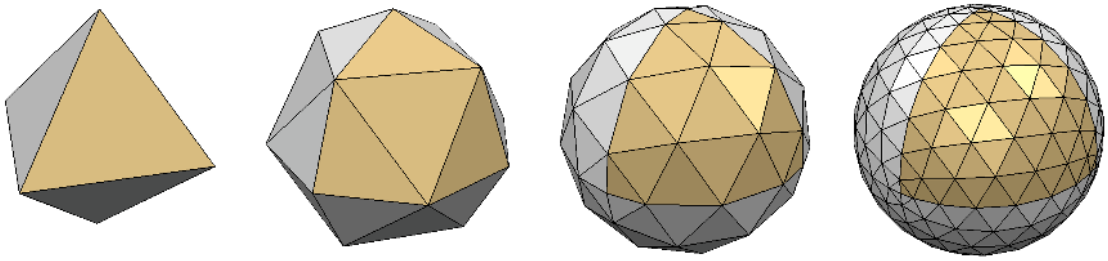


Figure 2: *Each subdivision step halves the edge lengths, increases the number of faces by a factor of 4, and reduces the error by a factor of* $\frac{1}{4}$.

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 $\mathcal{M}$ 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 connecting them

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

where each triangle specifies its three vertices from $\mathcal{V}$. However, as we will see in Section 3, it is sometimes more efficient to represent the connectivity of a triangle mesh in terms of the edges of the respective graph

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

The geometric embedding of a triangle mesh into $\mathbb{R}^3$ 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 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 error is reduced by a factor of $1/4$ by halving the edge lengths. As this refinement splits each triangle into four sub-triangles, it increases the number of triangles from F to $4F$ (cf. Fig. 2). Hence, 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 sufficient 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.

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. 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. 3). Non-manifold meshes are problematic for most algorithms, since around non-manifold configurations there exists no well-defined local geodesic neighborhoods.

Even irregular triangle meshes provide a certain topological structure. The famous Euler formula [Cox89] states an interesting relation between the numbers of vertices V, edges E and faces/triangles F in a closed and connected mesh:

$$V - E + F = 2(1 - g) \ , \tag{1}$$

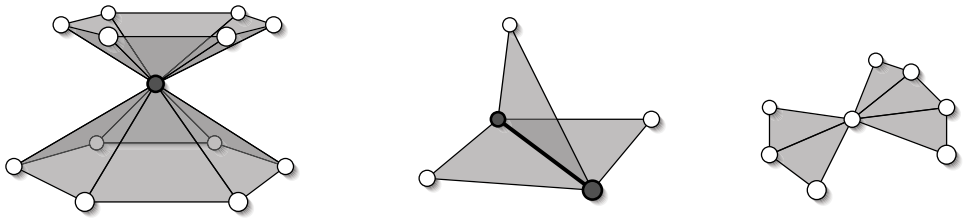


Figure 3: *Two surface sheets meet at a non-manifold vertex (*left*). A non-manifold edge has more than two incident faces (*center*). The right configuration, although being non-manifold in the strict sense, can be handled by most data structures.*

where g is the genus of the surface and intuitively represents the number of handles of an object (cf. Fig. 4). Since for typical meshes the genus is small compared to the numbers of elements, the right-hand side of Eq. (1) can be assumed to be almost zero. Given this and the fact that each triangle is bounded by three edges and that each (interior) edge is incident to two triangles, one can derive the following mesh statistics:

- The number of triangles is twice the number of vertices: $F \approx 2V$.
- The number of edges is three times the number of vertices: $E \approx 3V$.
- The average vertex valence (number of incident edges) is 6.

These relations will become important when considering data structures or file formats for triangle meshes in Section 3.

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. Since the development of efficient algorithms for triangle meshes requires suitable data structures, we will discuss this topic in detail in Section 3.

2.2. Implicit Surface Representations

The basic concept of *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.

There are different representations for implicit functions, like continuous algebraic surfaces, 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 object, such that the zero-level iso-surface $\mathcal{S}$ separates the inside from the outside.

As a consequence, geometric inside/outside queries simplify to function evaluations of F and checking the sign of the resulting value. This makes implicit representations well suited for constructive solid geometry (CSG), where complex objects are constructed by boolean operations of simpler ones (cf. Fig. 5). The different boolean operations can easily be computed by

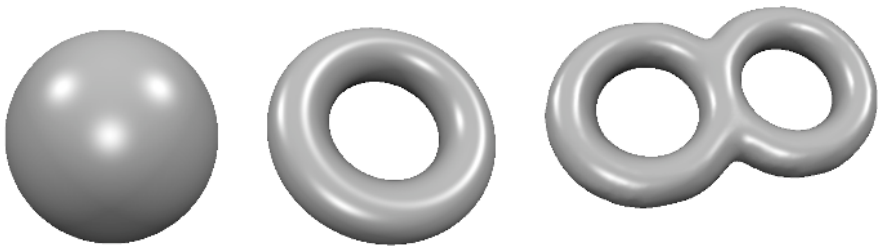


Figure 4: *From left to right: sphere of genus 0, torus of genus 1, double-torus of genus 2.*

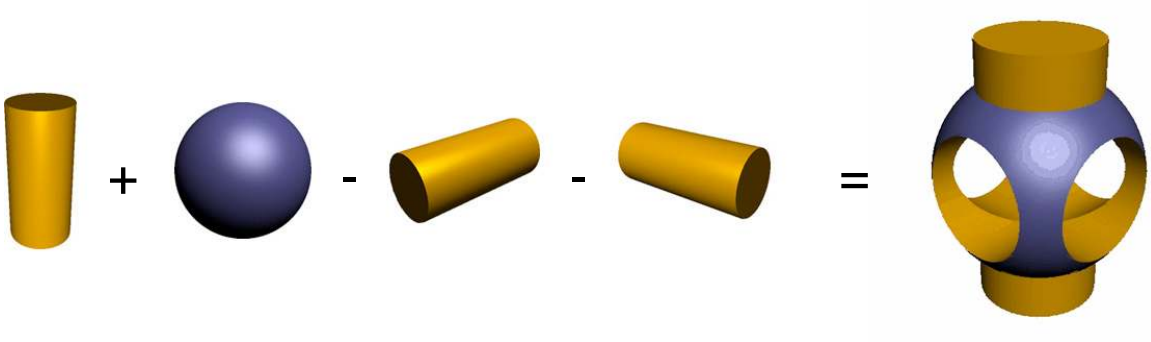


Figure 5: *A complex object constructed by boolean operations.*

simple min and max combinations of the objects' implicit functions. Hence, implicit surfaces can easily change their topology. Moreover, since an implicit surface is a level-set of a potential function, geometric self-intersections cannot occur, which will later be exploited for mesh repair (Section 4).

The implicit function F for a given surface $\mathcal{S}$ is not uniquely determined, but the most common and most natural 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, which can be used to compute and control the global error for mesh processing algorithms [WK03, BBVK04].

On the other hand, enumerating points on an implicit surface, finding geodesic neighborhoods, and even just rendering the surface is quite difficult. 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. Furthermore, boundaries cannot be represented.

2.2.1. Regular Grids

In order to efficiently process implicit representations, the continuous scalar field F is typically discretized in some bounding box around the object using a sufficiently dense grid with nodes $\mathbf{g}_{ijk} \in \mathbb{R}^3$. The most basic representation therefore is a uniform scalar grid of sampled values $F_{ijk} := F(\mathbf{g}_{ijk})$, and function values within voxels are derived by tri-linear interpolation, thus providing quadratic approximation order. However, the memory consumption of this naive data structure grows cubically if the precision is increased by reducing the edge length of grid voxels.

2.2.2. Adaptive Data Structures

For better memory efficiecy 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 store 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 (cf. Fig. 6, left).

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] (cf. Fig. 6, center). 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 (cf. Fig. 6, right).

2.3. Conversion Methods

In order to exploit the specific advantages of explicit and implicit surface representations efficient conversion methods between the different representations are necessary. However, notice that both kinds of representations are usually finite samplings

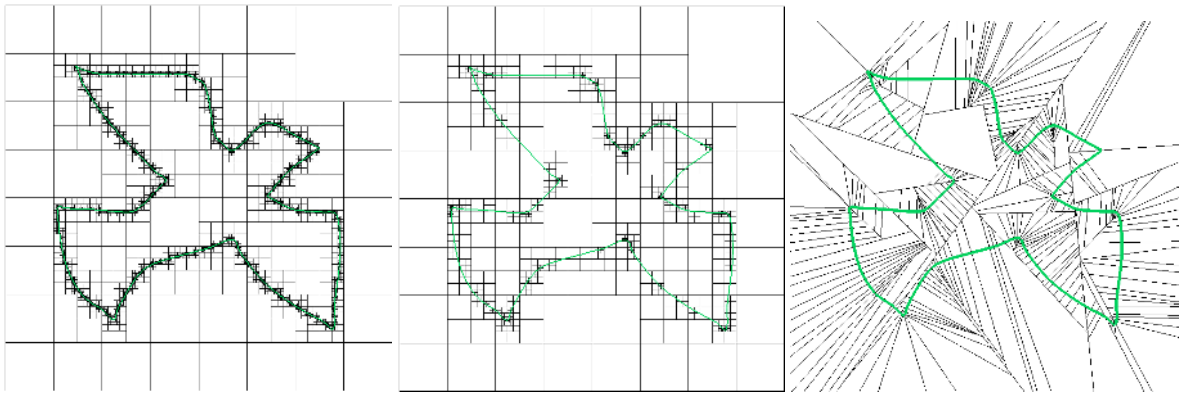


Figure 6: *Different adaptive approximations of a signed distance field with the same accuracy: 3-color quadtree (left, 12040 cells), ADF [FPRJ00] (center, 895 cells), and BSP tree [WK03] (right, 254 cells).*

(triangle meshes in the explicit case, uniform/adaptive grids in the implicit case) and that each conversion corresponds to a re-sampling step. Hence, special care has to be taken in order to minimize 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 or approximation 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. 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 we focus on triangle meshes as explicit representation, 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.

Computing the exact distance of a grid node to a given mesh amounts to computing the distance to the closest triangle, which can be found efficiently by spatial data structures. Notice that in order to compute a *signed* distance field, one additionally has to determine 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].

Computing the distances on a whole grid can be accelerated by *fast marching* methods [Set96]. In a first step, the exact signed distance values are computed for all grid nodes in the immediate vicinity of the triangle mesh. After this initialization, the fast marching method propagates distances to the unknown grid nodes in a breadth-first manner.

2.3.2. Implicit to Explicit

The conversion from an implicit or volumetric representation to an explicit triangle mesh, the so-called isosurface extraction, occurs for instance in CSG modeling (cf. Fig. 5) and in medical applications, e.g., to extract the skull surface from a CT head scan. The de-facto standard algorithm for isosurface extraction is *Marching Cubes* [LC87]. This grid-based method samples the implicit function on a regular grid and processes each cell of the discrete distance field separately, thereby allowing for trivial parallelization. For each cell that 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 $\mathcal{S}$.

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:

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

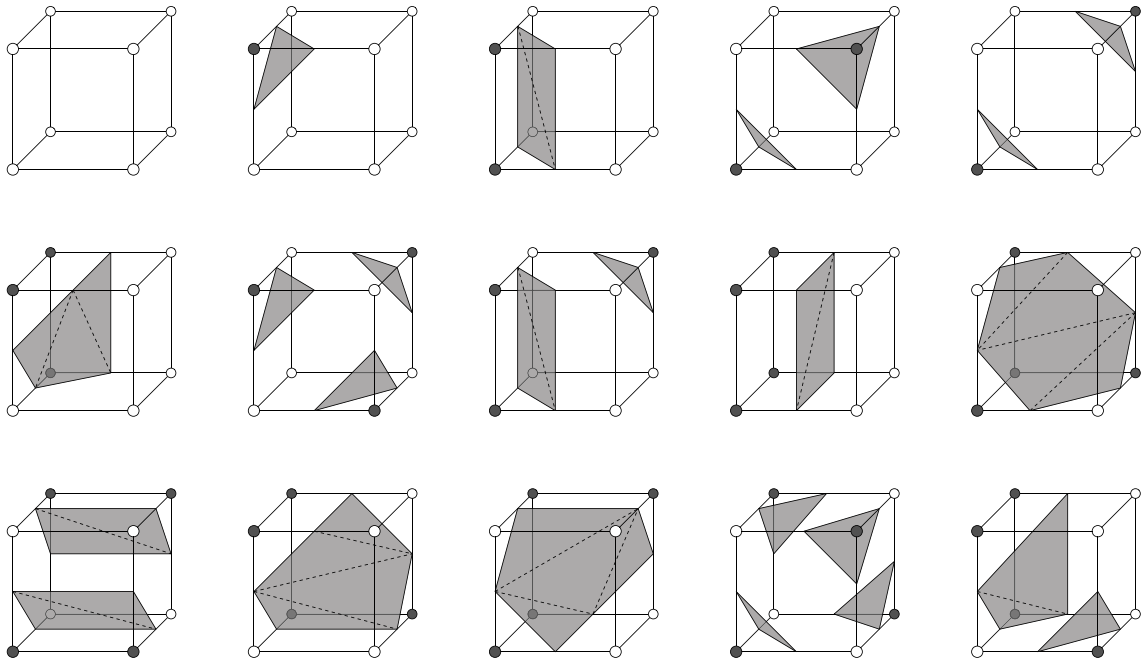


Figure 7: *The 15 base configurations of the Marching Cubes triangulation table. The other cases can be found by rotation or symmetry.*

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. 7). Since the possible combinatorial configurations are determined by the signs at a cell's corners, their number is $2^8 = 256$.

Notice that a few cell configuration are ambiguous, which might lead to cracks in the extracted surface. A properly modified look-up table yields a simple and efficient solution, however, at the price of sacrificing the symmetry w.r.t. sign inversion of F [MSS94]. The resulting isosurfaces then are watertight 2-manifolds, which is exploited by many mesh repair techniques (Section 4).

Notice that Marching Cubes computes intersection points on the edges of a regular grid only, which causes sharp edges or corners to be "chopped of". A faithful reconstruction of sharp features would instead require additional sample points within the cells containing them. The extended Marching Cubes [KBSS01] therefore examines the distance function's gradient ∇F to detect those cells and to find additional sample points by intersecting the tangent planes at the edge intersection points. This principle is depicted in Fig. 8, and a 3D example of the well known fandisk dataset is shown in Fig. 9. An example implementation of the extended Marching Cubes based on the OpenMesh data structure [BSM05] can be downloaded from [Bot05a].

The high complexity of the extracted isosurfaces remains a major problem for Marching Cubes like approaches. Instead of decimating the resulting meshes (Section 9), Ju et al. [JLSW02] proposed the *dual contouring* approach, which allows to directly extract adaptive meshes from an octree. Notice however that their approach yields non-manifold meshes for cell configurations containing multiple surface sheets. A further promising approach is the cubical marching squares algorithm [HWC*05], which also provides adaptive and feature-sensitive isosurface extractions.

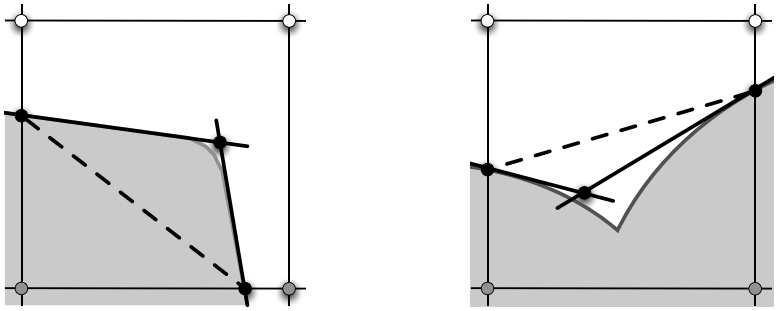


Figure 8: *By using point and normal information on both sides of the sharp feature one can find a good estimate for the feature point at the intersection of the tangent elements. The dashed line is the result the standard Marching Cubes algorithm would produce.*

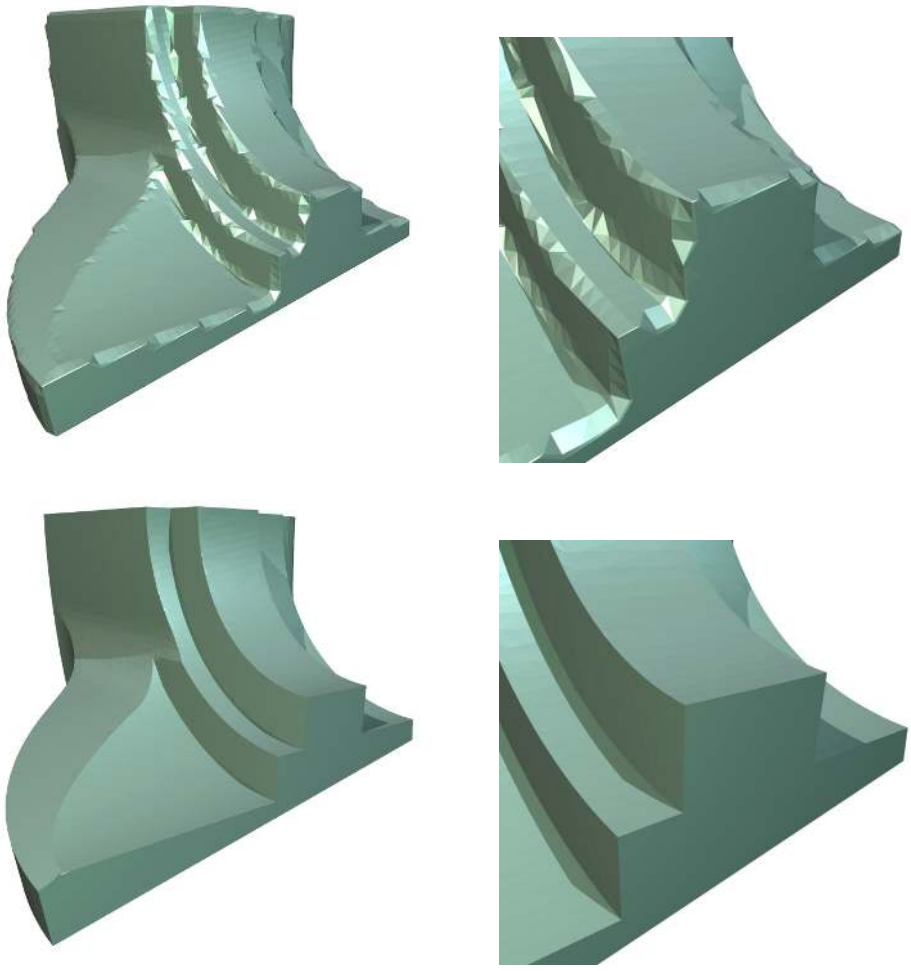


Figure 9: *Two reconstructions of the "fandisk" dataset from a* $65 \times 65 \times 65$ *sampling of its signed distance field. The standard Marching Cubes algorithm leads to severe alias artifacts near sharp features (*top*), whereas the feature-sensitive iso-surface extraction faithfully reconstructs them (*bottom*).*

3. Mesh Data Structures

The efficiency of the geometric modeling algorithms presented in this tutorial crucially depends on the underlying mesh data structures. A variety of data structures has been described in the literature, and a number of different implementations are available. We refer to [Ket98] for an excellent overview and comparison of different mesh data structures and to [FH03, FH05b] for references on data structures for representing non-manifold meshes. In general, when choosing a data structure one has to take into account topological as well as algorithmic considerations:

Topological requirements. Which kinds of meshes need to be represented by the data structure? Do we need boundaries or can we assume closed meshes? Do we need to represent complex edges and singular vertices (see Section 4) or can we rely on a manifold mesh? Can we restrict ourselves to pure triangle meshes or do we need to represent arbitrary polygonal meshes? Are the meshes regular, semi-regular or irregular? Do we want to build up a hierarchy of differently refined meshes or do we need only a flat data structure?

Algorithmic requirements. Which kinds of algorithms will be operating on the data structure? Do we simply want to render the mesh? Do we need to modify only the geometry of the mesh, or do we also have to modify the connectivity/topology? Do we need to associate additional data with the vertices, edges or faces of the mesh? Do we need to have constant-time access to the local neighborhoods of vertices, edges and faces? Can we assume the mesh to be globally orientable?

The simplest representation for triangle meshes would just store a set of *individual* triangles. Some data exchange formats use this representation as a common denominator (e.g., STL format). However, it is immediately clear that this is not sufficient for most requirements: connectivity information cannot be accessed explicitly, and vertices and associated data are replicated. The latter can be fixed by a *shared vertex* data structure, which stores a table of vertices and encodes triangles as triples of indices into this table. In fact this representation is used in many file formats because it is simple and efficient in storage. For comparison we consider a triangle mesh with n vertices and m triangles. For every vertex, only its position is stored as three `float` values. From Euler's formula (Section 2) we know that $m \approx 2n$. Then storing individual triangles results in $3 \cdot 3 \cdot 4 = 36$ bytes per triangle or $36m \approx 72n$ bytes in total. In contrast, a *shared vertex* representation requires $3 \cdot 4 = 12n$ bytes for the vertex positions and $3 \cdot 4m \approx 24n$ bytes for the index table (assuming 4 bytes per integer value), which encodes connectivity.

The shared vertex data structure can be sufficient for algorithms that do not require efficient neighborhood information, like mesh rendering for instance. However, without additional connectivity information this representation is still not efficient for most algorithms. For *face based* data structures each vertex therefore stores a reference to one of its incident triangles, and for each triangle stores references to its three vertices and its three neighboring triangles. This results in a total of $64n$ bytes. This data structure enables more efficient enumeration of neighbors, however, there is no explicit notion of edges, and local searches are necessary to match vertices in neighboring triangles.

Before we go on, we want to identify some minimal set of operations that are frequently used by most algorithms.

- Access of individual vertices, edges, faces. This includes enumeration of *all* elements (in no particular order).
- Oriented traversal of edges of a face, which refers to finding the *next* edge in a face. (This defines also *degree* of the face and the inverse operation for the *previous* halfedge. With additional access to vertices, e.g., rendering of faces is enabled.)
- Access of the faces attached to an edge. Depending on orientation this is either the left or right face in the manifold case. This enables access to neighboring faces and hence traversal of faces (and boundaries as special case).
- Given an edge, access its two vertices.
- Given a vertex, at least one incident face or edge must be accessible. Then (for manifold meshes) all other elements in the one-ring neighborhood of a vertex can be enumerated, i.e., incident faces, edges, or neighboring vertices.

These operations enable local and global traversal of the mesh. They relate vertices, edges and faces by connectivity information (and orientation). We remark that all these operations are possible even for a shared vertex representation, however, this requires expensive searches.

Several data structures have been developed which enable fast traversal of meshes. Well-known are *winged-edge* [Bau72], *quad-edge* [GS85], and *half-edge* [Man88] data structures in different flavors (see, e.g., [O'R94]).

The winged-edge data structure stores one edge reference per vertex, references to two vertices, four adjacent edges, and two adjacent faces per edge, and a reference to one edge per face. This edge-based data structure requires $120n$ bytes in total and enables efficient access and traversal. However, it still lacks orientation of edges, and therefore requires distinction of cases (forward/backward) when traversing local neighborhoods.

For geometry processing, we recommend data structures based on oriented *halfedges* , in particular halfedge data structure (Section 3.1) and directed edges structure [CKS98] (Section 3.2) as a special case for triangle meshes. Both data structures allow for efficient enumeration of neighborhoods of vertices and faces. This operation is frequently used in many algorithms, e.g., in mesh smoothing and mesh decimation. The halfedge data structure is able to represent arbitrary polygonal meshes that are subsets of a 2-manifold. For triangle meshes it requires $120n$ bytes, this is the same as for winged-edges. The directed edges data structure is more memory efficient ($64n$ bytes), but it can only represent 2-manifold triangle meshes.

3.1. Halfedge Data Structure

One of the most convenient and flexible data structures in geometry processing is the halfedge data structure [Man88, Ket98]. This structure is able to represent arbitrary polygonal meshes that are subsets of orientable 2-manifolds. In this data structure each edge is split into two opposing halfedges such that all halfedges are oriented consistently in counter-clockwise order around each face and along the boundary, see Fig. 10. For each halfedge we store a reference to

- the vertex it points to
- its adjacent face (a zero pointer, if it is a boundary halfedge)
- the next halfedge of the face or boundary (in counter-clockwise direction)
- its inverse (or opposite) halfedge
- the previous half-edge in the face (*optional* for better performance)

Additionally we store references for each face to one of its adjacent halfedges and for each vertex to one of its outgoing halfedges. Thus, a basic halfedge structure can be realized using the following classes:

```
struct Halfedge
{
  HalfedgeRef   next_halfedge;
  HalfedgeRef   opposite_halfedge;
  FaceRef       face;
  VertexRef     to_vertex;
};
```

```
struct Face
{
  HalfedgeRef   halfedge;
};
```

```
struct Vertex
{
  Point         position;
  HalfedgeRef   outgoing_halfedge;
};
```

This simple structure already enables us to enumerate for each element (i.e. vertex, edge, halfedge or face) its adjacent elements. As an example, the following procedure enumerates all vertices that are adjacent to a given center vertex (the so-called 1-ring)

```
enumerate_1_ring(Vertex * center)
{
  HalfedgeRef h = outgoing_halfedge(center);
  HalfedgeRef hstop = h;
  do {
    VertexRef v = to_vertex(h);
    // do something with v
    h = next_halfedge(opposite_halfedge(h));
  } while ( h != hstop );
}
```

The implementation of the references (e.g., HalfedgeRef) can be realized in different ways, for instance using pointers or indices. In practice, index representations (see, e.g., Section 3.2) are more flexible even though memory access is indirect: using indices into data arrays enables efficient memory relocation (and simpler and more compact memory management) and *all* attributes of a vertex (edge, face) are identified by the same index. As a side effect, use of indices is platform compatible. More important in this context is the following observation: halfedges always come in pairs. Thus when we actually implement a halfedge data structure we group inverse halfedges pairwise in an array. This trick has two advantages: first, the opposite halfedge is given implicitly by an addition modulo two so there is no need to explicitly store it. Second, we obtain an explicit representation for "full" edges, which is important when we want to associate data with edges rather than halfedges. (Note that this is generally also possible with a pointer implementation.)

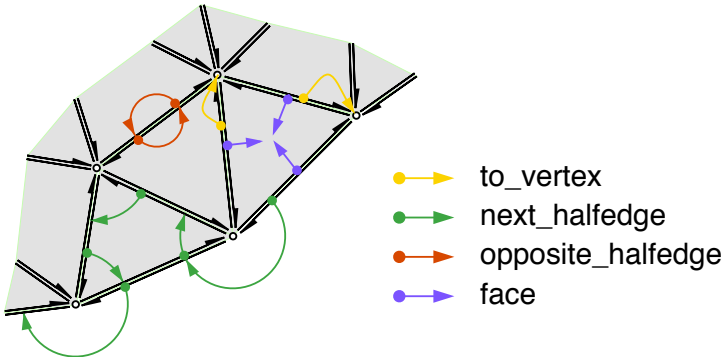


Figure 10: *This figure shows the references stored with each halfedge. Note that the* next_halfedge *references enable traversing the boundary loop.*

3.2. Directed Edges

The directed edges data structure [CKS98] is a memory efficient variant of the halfedge data structure designed for triangles meshes. It has the following restrictions:

- Only triangle meshes can be represented.
- There is no explicit representation of edges.

The main benefit of directed edges is memory efficiency while they can represent all triangle meshes which can be represented by the general halfedge data structure. In addition some atomic operations are more efficient than for general halfedges. However, traversing boundary loops is more expensive as there is no atomic operation to enumerate the next boundary edge.

The directed edges data structure is based on indices as references to each element (vertex, face, halfedge). The indexing is not arbitrary but follows certain rules that *implicitly* encode some of the connectivity information of the triangle mesh. Instead of pairing opposite halfedges (see above), this data structure groups the three halfedges belonging to a common triangle. To be more precise, let f be the index of a face, then the indices of its three halfedges are given as

$$\mathsf{halfedge}(f,i) = 3f+i, \quad i = 0,1,2$$

Now let h be the index of a halfedge. Then the index of its adjacent face is simply given by

$$\mathrm{face}(h) = h/3$$

Not surprisingly, we can also compute the index of h's next halfedge as $(h+1) \mod 3$. The remaining parts of the connectivity have to be stored explicitly in arrays. Thus for each vertex we store the index of an outgoing halfedge. For each halfedge, we store the index of its opposite halfedge and the index of the vertex, the halfedge points to.

Notes

- The directed edge data structure handles boundaries by special (e.g., negative) indices indicating that the inverse edge is invalid. This leads to a non-uniform treatment of the connectivity encoding and some special cases.
- We have described the directed edges data structure for pure triangle meshes. An adaption to pure quad meshes is straightforward. However, it is not possible to mix triangles and quads, which severely limits this extension to regular settings.

3.3. Mesh Libraries: CGAL and OpenMesh

Although the description of a halfedge data structure is straightforward, its implementation is not. Programming a basic mesh data structure might thus be a good exercise for an undergraduate course in geometric modeling, but designing and implementing a full-fledged mesh library that is memory- and time-efficient, robust and easy to use and that is possibly equipped with a number of standard operations and algorithms is an advanced and time consuming task. Among others the following issues have to be taken into account:

- *Access:* How can we conveniently access vertices, edges and faces? How can we conveniently enumerate neighborhoods or navigate along mesh boundaries?
- *Modification:* How can a mesh be modified by the user? How can vertices and faces be added or deleted? How can we guarantee that after a modification the data structure is still consistent?
- *Composed operations:* How can high level operations like halfedge-collapses, face-splits etc. be implemented efficiently?
- *Parameterization:* How can arbitrary additional data efficiently be with the vertices, edges and faces of the mesh? What kind of memory management is efficient?
- *Input and output:* How to read and write data from different file formats? How to build up a halfedge-structure from an indexed face set?

Taking all these issues into account and coping with the often subtle problems when modifying the data structure, we strongly recommend to use one of full featured, publicly available mesh libraries. We refer the interested programmer to the following C++ libraries.

CGAL, the Computational Geometry Algorithms Library, is a generic C++ library for geometric computing. It provides basic geometric primitives and operations, as well as a collection of standard data structures and geometric algorithms, including 3D polyhedral surfaces with a halfedge data structure and a rich set of 2D and 3D triangulations. CGAL is specifically designed to provide reliable solutions to efficiency and robustness issues which are of crucial importance in geometric algorithms. Robustness and scalability of the algorithms are achieved by isolating a minimal number of predicates and constructors, and by the use of templated kernels. The CGAL library is available at `http://www.cgal.org`.

OpenMesh provides efficient halfedge data structures for polygonal meshes, their input/output and several standard geometry processing algorithms. OpenMesh is available at `http://www.openmesh.org`.

Comparing objectives and functionalities of these two libraries, CGAL is much more ambitious. Its rich foundation of algorithms is strongly biased by computational geometry with focus on robust and exact algorithms. CGAL has a wide user base and a number of research institutions actively contribute to its development. A major difference in data structures is the support for tetrahedral meshes. In contrast, OpenMesh is highly specialized on efficient processing of surface meshes based solely on halfedge data structures. It takes over some concepts of CGAL which provided one of the first publicly available halfedge data structures. It is much more focused on requirements of modeling with polygonal meshes and provides a set of standard geometry processing algorithms, like mesh smoothing, decimation, etc. We note that both libraries have different licensing policies.

As some authors of this tutorial were actively involved in the design and implementation of OpenMesh, we will describe this library in more detail here. Note that the same functionality is available in CGAL, however, the code reads differently.

- *Access:* Vertices, edges, halfedges and faces are all explicitly represented in OpenMesh and can easily be accessed through iterators or through handles (which replace indices as references). OpenMesh also provides so-called circulators that allow to enumerate the neighborhoods of each element. The following example shows how to compute the barycenter of the 1-ring of each vertex in a mesh:

```
TriangleMesh mymesh;
(...) // Read a mesh
// A VertexIter is an STL-compliant iterator to enumerate all vertices of a mesh
for ( VertexIter vi = mymesh.vertices_begin(); vi != mymesh.vertices_end(); ++vi )
{
    int cnt = 0;
    Point cog(0,0,0);
    // A VertexVertexIter is a circulator that enumerates the 1-ring of a vertex
    for ( VertexVertexIter vvi = mymesh.vv_iter(vi); vvi; ++vvi )
    {
        cnt += 1;
        cog += mymesh.point(vvi);
    }
    cog /= cnt;
    // Now cog equals the center of gravity of vi's neighbors
}
```

- *Modification:* OpenMesh provides functions to add and remove vertices and faces to and from a mesh. These operations are guaranteed to preserve a consistent state of the mesh. The following example shows how to add a triangle to a mesh:

```
TriangleMesh mymesh;

// Add three vertices to the mesh
VertexHandle v0 = mymesh.add_vertex( Point( 0, 0, 0 ) );
VertexHandle v1 = mymesh.add_vertex( Point( 0, 1, 0 ) );
VertexHandle v2 = mymesh.add_vertex( Point( 3, 0, 2 ) );

// Connect the vertices by a triangle
FaceHandle f = mymesh.add_face( v0, v1, v2 );

// Remove the face
mymesh.delete_face( f );
```

- *Composed operations:* OpenMesh provides a number of high-level operations, among them halfedge-collapse, vertex-split, face-split, edge-split and edge-flip. It also provides functions that test whether a certain operation is legal or not. The following snippet of code tries to collapse all edges that are shorter than a given threshold:

```
TriangleMesh mymesh;

(...)

for ( HalfedgeIter hi = mymesh.halfedges_begin(); hi != mymesh.halfedges_end(); ++hi )
    if ( ! mymesh.status( hi ).is_deleted() )
    {
        Point a = mymesh.point( mymesh.from_vertex_handle( hi ) );
        Point b = mymesh.point( mymesh.to_vertex_handle( hi ) );
        if ( (b-a).norm() < epsilon && mymesh.is_collapse_ok( hi ) )
            mymesh.collapse( hi );
    }
mymesh.garbage_collection();
```

- *Parameterization:* Arbitrary additional data can be associated with the vertices, edges, halfedges or faces of a mesh via OpenMesh's property mechanism. This mechanism allows to assign and remove data from the mesh at runtime. Thus it is for example possible to temporarily assign to each edge a weight:

```
TriangleMesh mymesh;

(...)

// Add a property (in this case a float) to each edge of mymesh
EdgePropertyHandle< float > weight;
mymesh.add_property(weight);

// Assign values to the properties
for ( EdgeIter ei = mymesh.edges_begin(); ei != mymesh.edges_end(); ++ei )
    mymesh.propery( weight, ei ) = some_value;
(...)

// Do something with the properties
for ( EdgeIter ei = mymesh.edges_begin(); ei != mymesh.edges_end(); ++ei )
    do_something_with( mymesh.propery( weight, ei ) );

(...)

// If the weights are not needed anymore, remove them to free some memory
mymesh.remove_property(weight);
```

- *Input and output:* OpenMesh reads and writes `stl` (ASCII and Binary), `off` and `obj` files. Handlers for other file types can easily be added by the user.

```
TriangleMesh mymesh;

read_mesh( mymesh, "a_filename.off" );
(...)
write_mesh( mymesh, "another_filename.stl" );
```

- *Standard algorithms:* OpenMesh provides a set of standard algorithms that can easily be customized to different needs. Among these algorithms are: smoothing (Section 7), decimation (Section 9) and subdivision (see also Section 10).

3.4. Summary

Efficient data structures are crucial for geometry processing based on polygonal meshes. We recommend halfedge data structures (or directed edges as a special case for triangle meshes), for which full-featured and publicly available implementations already exist, e.g. CGAL or OpenMesh.

4. Model Repair

In short, model repair is the task of removing artifacts from a geometric model to produce an output model that is suitable for further processing by downstream applications that have certain quality requirements on their input. Although this definition is most often too general, it nonetheless captures the essence of model repair: the definition of what we mean by a "model", of what exactly constitutes an "artifact" and what is meant by "suitable for further processing" is highly dependent on the problem at hand and there generally is no single algorithm which is be applicable in all situations.

Model repair is a necessity in a wide range of applications. As an example, consider the design cycle in automotive CAD, CAE and CAM: Car models are typically manually designed in CAD systems that use trimmed NURBS surfaces as the underlying data structure for representing geometry. However, downstream applications like numerical fluid simulations cannot handle NURBS patches but need a watertight, manifold triangle mesh as input. Thus there is a need for an intermediate stage that converts the NURBS model into a triangle mesh. Unfortunately, this conversion process often produces artifacts that cannot be handled by downstream applications. Thus, the converted model has to be repaired — often in a manual and tedious post-process.

The goal of this tutorial is to give a practical view on the typical types of artifacts that occur in geometric models and to introduce the most common algorithms that address these artifacts. After giving a short overview on the common types of artifacts in Section 4.1, we start out in Section 4.2 by classifying repair algorithms on whether they *explicitly* identify and resolve artifacts or on whether they rely on an intermediate *volumetric* representation that automatically enforces certain consistency constraints. This classification already gives a hint on the strengths and weaknesses of a particular algorithm and on the quality that can be expected from its output. In Section 4.3 we then give an overview on the different types of input models that are encountered in practice. We describe the specific artifacts and problems of each model and explain their origin. We also give references to algorithms that are designed to resolve these artifacts. Finally, we present some of the common model repair algorithms in more detail in Section 4.5. We give a short description on how each algorithm works and to which models it is applicable. We hope that this provides a deeper understanding of the often subtle problems that occur in model repair and of ways to address these problems. Some of these algorithms are relatively straightforward, while others are more involved such that we can only show their basic mechanisms.

4.1. Artifact Chart

The chart in Fig. 11 shows the most common types of artifacts that occur in typical input models. Note that this chart is by no means complete and in particular in CAD models one encounters further artifacts like self-intersecting curves, points that do not lie on their defining planes and so on. While some of these artifacts, e.g., complex edges, have a precise meaning, others, like the distinction between small scale and large scale overlaps, are described intuitively rather than by strict definitions.

4.2. Types of Repair Algorithms

Most model repair algorithms can roughly be classified as being either *surface oriented* or *volumetric*. Understanding these concepts helps to evaluate the strengths and weaknesses of a given algorithm and the quality that can be expected of its output.

Surface oriented algorithms operate directly on the input data and try to explicitly identify and resolve artifacts on the surface. For example, gaps could be removed by snapping boundary elements (vertices and edges) onto each other or by stitching triangle strips in between the gap. Holes can be closed by a triangulation that minimizes a certain error term. Intersections could be located and resolved by explicitly splitting edges and triangles.

Surface oriented repair algorithms only minimally perturb the input model and are able to preserve the model structure in areas that are away from artifacts. In particular, structure that is encoded in the connectivity of the input (e.g. curvature lines) or material properties that are associated with triangles or vertices are usually well preserved. Furthermore, these algorithms introduce only a limited number of additional triangles.

To guarantee a valid output, surface oriented repair algorithms usually require that the input model already satisfies certain quality requirements (error tolerances). These requirements cannot be guaranteed or even be checked automatically, so these algorithms are rarely fully automatic but need user interaction and manual post-processing. Furthermore, due to numerical inaccuracies, certain types of artifacts (like intersections or large overlaps) cannot be resolved robustly. Other artifacts, like gaps between two closed connected components of the input model that are geometrically close to each other, cannot even be identified.

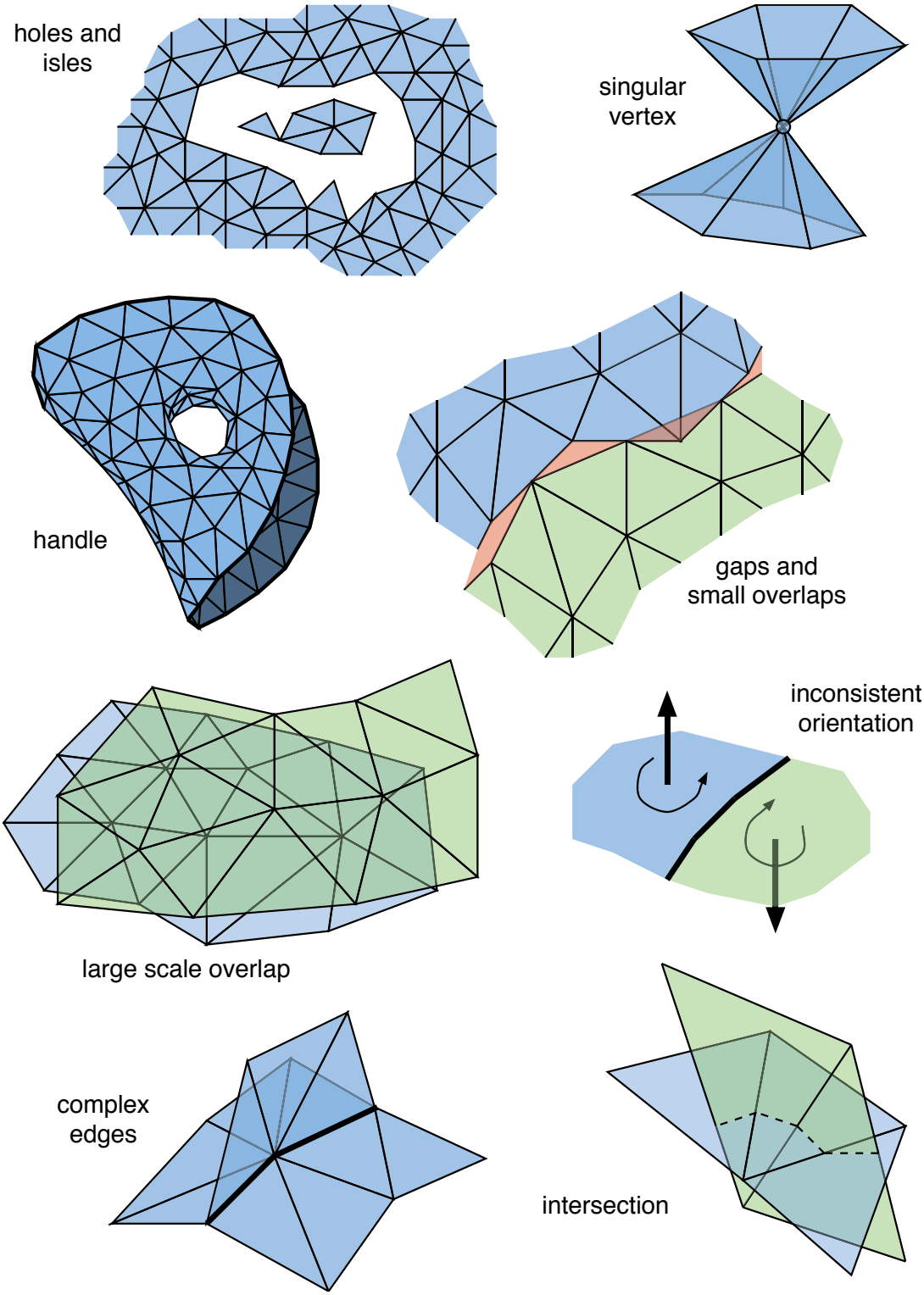


Figure 11: *Artifact chart*

Volumetric algorithms convert the input model into an intermediate volumetric representation from which the output model is then extracted. Here, a volumetric representation is any kind of partitioning of space into cells such that each cell can be classified as either being *inside* or *outside*. Examples of volumetric representations that have been used in model repair include regular Cartesian grids, adaptive octrees, *kd*-trees, BSP-trees and Delaunay triangulations, see also Section 2. The interface between inside and outside cells then defines the topology and the geometry of the reconstructed model. Due to their very nature, volumetric representations do not allow for artifacts like intersections, holes, gaps or overlaps or inconsistent normal orientations. Depending on the type of the extraction algorithm, one can often also guarantee the absence of complex edges and singular vertices. Handles, however, might still be present in the reconstruction.

Volumetric algorithms are typically fully automatic and produce watertight models (Section 2.3.2). Depending on the type of volume, they can often be implemented very robustly. In particular, the discrete neighborhood relation of cells allows to reliably extract a consistent topology of the restored model. Furthermore, well-known morphological operators can be used to robustly remove handles from the volume.

On the downside, the conversion to and from a volume leads to a resampling of the model. It often introduces aliasing artifacts, loss of model features and destroys any structure that might have been present in the connectivity of the input model. The number of triangles in the output of a volumetric algorithm is usually much higher than that of the input model and thus has to be decimated in a post-processing step. Also the quality of the output triangles often degrades and has to be improved afterwards (see also Fig. 9). Finally, volumetric representations are quite memory intensive so it is hard to run them at high resolutions.

4.3. Types of Input

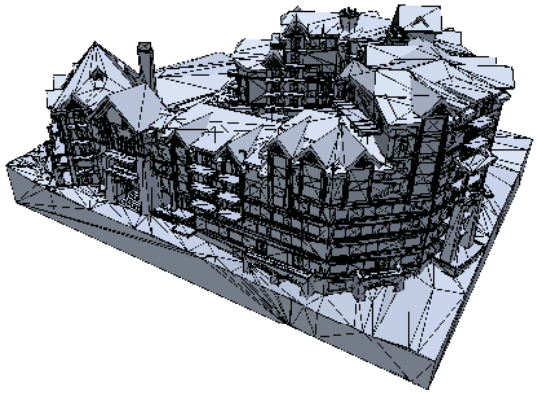
In this section we list the most common types of input models that occur in practice. For each type we describe its typical artifacts (see also Section 4.1) and give references to algorithms that can be used to remove them.

Registered Range Scans are a set of patches (usually triangle meshes) that represent overlapping parts of the surface $\mathcal{S}$ of a scanned object. While large overlaps are a distinct advantage in registering the scans, they pose severe problems when these patches are to be fused into a single consistent triangle mesh. The main geometric problem in this setup are the potentially very large overlaps of the scans such that a point $\mathbf{x}$ on $\mathcal{S}$ is often described by multiple patches that do not necessarily agree on $\mathbf{x}$'s position. Furthermore, each patch has its own connectivity that is usually not compatible to the connectivity of the other patches. This is in particular a problem for surface oriented repair algorithms.

There are only a few surface oriented algorithms for fusing range images, e.g., Turk et al.'s mesh zippering algorithm [TL94]. The most well-known volumetric method is due to Curless and Levoy [CL96].

Fused Range Scans Fused range images are manifold meshes with boundaries, i.e., holes and isles. These artifacts are either due to obstructions in the line of sight of the scanner or result from bad surface properties of the scanned model such as transparency or glossiness. The goal is to identify and fill these holes. In the simplest case, the filling is a patch that minimizes some bending energy and joins smoothly to the boundary of the hole. Advanced algorithms synthesize new geometric detail that resembles the detail that is present in a local neighborhood of the hole or transplant geometry from other parts of the model in order to increase the realism of the reconstruction. The main obstacles in hole filling are the incorporation of isles into the reconstruction and the avoidance of self-intersections.

Kliencsek proposes an algorithm based on dynamic programming for finding minimum weight triangulations of planar polygons [Kli80]. This algorithm is a key ingredient in a number of other model repair algorithms. Liepa proposes a surface oriented method to smoothly fill holes such that the vertex densities around the hole are interpolated [Lie03]. Podolak et al. cast hole filling as a graph-cut problem and report an algorithm that is guaranteed to produce non-intersecting patches [PR05]. Davis et al. propose a volumetric method that diffuses a signed distance function into empty regions of the volume [DMGL02]. Pauly et al. use a database of geometric priors from which they select shapes to fill in regions of missing data [PMG*05].

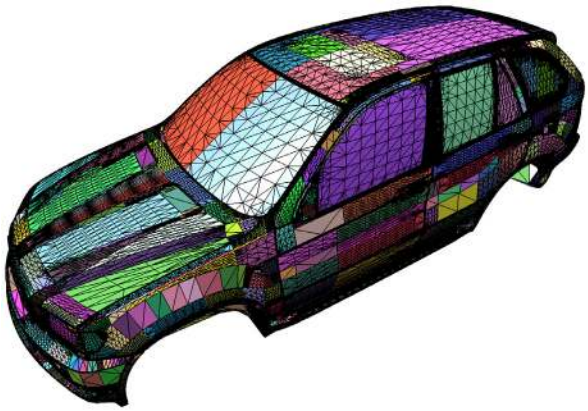


Triangle Soups are mere sets of triangles with no or only little connectivity information. They most often arise in CAD models that are manually created in a boundary representation where users typically assemble predefined elements (taken from a library) without bothering about consistency constraints. Due to the manual layout, these models typically are made of only a few thousands of triangles, but they may contain all kinds of artifacts. Thus triangle soups are well suited for visualization, but cannot be used in most downstream application.

Intersecting triangles are the most common type of artifact in triangle soups, as the detection and in particular the resolution of intersecting geometry would be much too time-consuming and numerically unstable. Complex edges and singular vertices are often intentionally created in order to avoid the duplication of vertices and the subsequent need to keep these duplicate vertices consistent. Other artifacts include inconsistent normal orientations, small gaps and excess interior geometry.

Surface oriented methods that are able to automatically repair triangle soups are not known. However, there are a number of volumetric methods that can be applied to triangle soups: Murali et al. produce a BSP tree from the triangle soup and automatically compute for each leaf a solidity [MF97]. Nooruddin et al. use ray-casting and filtering to convert the triangle soup into a volumetric representation from which they then extract a consistent, watertight model [NT03]. Shen et al. create an implicit representation by generalizing the moving least squares approach from point sets to triangle soups [SOS04]. Bischoff and Kobbelt scan convert the soup into a binary grid, use morphological operators to determine inside/outside information and then invoke a feature-sensitive extraction algorithm [BPK05]. Gress and Klein use a *kd*-tree to improve the geometric fidelity of the volumetric reconstruction [GK03].

Tringulated NURBS Patches typically are a set of triangle patches that contain gaps and small overlaps. These artifacts arise when triangulating two or more trimmed NURBS patches that join at a common boundary curve. Usually, each patch is triangulated separately, thus the common boundary is sampled differently from each side. Other artifacts present in such models include intersecting patches and inconsistent normal orientations. Triangulated NURBS patches are usually repaired using surface oriented methods. These methods first try to establish a consistent orientation of the input patches. Then they identify corresponding parts of the boundary and snap these parts onto each other. Thus any structure that might be present in the triangulation (like iso-lines, curvature lines, etc.) is preserved.



Barequet and Sharir use a geometric hashing technique to identify and bridge boundary parts that have a similar shape [BS95]. Barequet and Kumar describe an algorithm that identifies geometrically close edges and snaps them onto each other [BK97]. Borodin and Klein generalize the vertex-contraction operator to a vertex-edge contraction operator and thus are able to progressively close gaps [BNK02]. Bischoff and Kobbelt use a volumetric repair method locally around the artifacts and stitch the resulting patches into the remaining mesh [BK05a]. Borodin et al. propose an algorithm to consistently orient the normals which takes visibility information into account [BZK04].

Contoured Meshes are meshes that have been extracted from a volumetric dataset by Marching Cubes, Dual Contouring or other extraction algorithms. Provided that the correct triangulation look-up tables are used, contoured meshes are always guaranteed to be watertight and manifold (Section 2.3.2). However, these meshes often contain topological artifacts, such as small handles.

Volumetric data arises most often in medical imaging (CT, MRI,...), as an intermediate representation when fusing registered range scans or in constructive solid geometry (CSG). In a volumetric dataset, each voxel is classified as being either inside or outside the object. Unfortunately, due to the finite resolution of the underlying grid, voxels are often classified wrongly (so-called partial volume effect). This leads to topological artifacts in the reconstruction, like handles, holes, or disconnected components, that are not consistent with the model that should be represented by the volume. A famous example are MRI datasets of the brain: It is well known that the surface of the brain is homeomorphic to a sphere, but all too often a model of higher genus is extracted.

While disconnected components and small holes can easily be detected and removed from the main part of the model, handles are more problematic. Due to the simple connectivity of the underlying Cartesian grid, it is usually easiest to remove them from the volume dataset before applying the contouring algorithm or to identify and resolve them during reconstruction [WHDS04]. Guskov and Wood presented one of the few surface oriented algorithms to remove handles from an input mesh [GW01].

Badly Meshed Manifolds contain degenerate elements like triangles with zero area, caps, needles and triangle flips. These meshes result from the tessellation of CAD models or are the output of marching cubes like algorithms, in particular if they are enhanced by feature-preserving techniques. Although badly meshed manifolds are in fact manifold and even often watertight, the degenerate shape of the elements prevents further processing, e.g., in finite element meshers, and leads to instabilities in numerical simulations. The repair of such meshes is called *remeshing,* and we discuss this issue in depth in Section 10.

4.4. Surface Oriented Algorithms

In this section we describe some of the most common surface oriented repair algorithms. These algorithms work directly on the input surface and try to remove artifacts by explicitly modifying the geometry and the connectivity of the input.

4.4.1. Consistent Normal Orientation

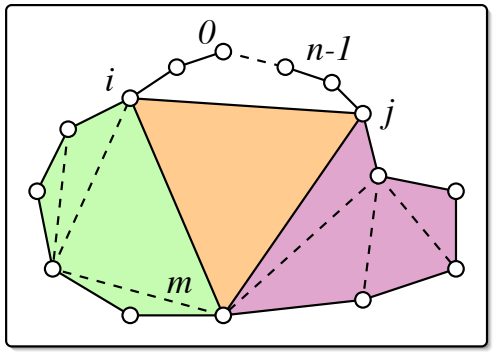
Consistently orientating the normals of an input model is part of most surface oriented repair algorithms and can even improve the performance of volumetric algorithms. Usually the orientation of the normals is propagated along a minimum spanning tree between neighboring patches either in a preprocessing step or implicitly during traversal of the input. Borodin et al. describe a more sophisticated algorithm that additionally takes visibility information into account [BZK04].

The input is a set of arbitrarily oriented polygons. In a preprocessing phase the polygons are assembled into larger, manifold patches (possibly with boundary) as described in Section 4.4.3. The algorithm then builds up a connectivity graph of neighboring patches where the label of each edge encodes the *normal coherence* of the two patches. Furthermore, for each side of each patch a *visibility coefficient* is computed that describes how much of the patch is visible when viewed from the outside. Finally, a globally consistent orientation is computed by a greedy optimization algorithm: If the coherence of two patches is high, normal consistency is favoured over front-face visibility and vice versa.

4.4.2. Surface Based Hole Filling

In this section we describe an algorithm for computing a fair triangulation of a hole. The algorithm was proposed by Liepa [Lie03] and builds on work of Klincsek [Kli80] and Barequet and Sharir [BS95]. It is a basic building block of many other repair algorithms.

The goal is to produce a triangulation of a polygon $\mathbf{p}_0,\ldots,\mathbf{p}_{n-1}$ that minimizes some given weight function. In the context of mesh repair, this weight function typically measures the fairness of the triangulation, e.g., its area or the variation of the triangle normals (see also Section 6).



Let $\phi(i,j,k)$ be a weight function that is defined on the set of all triangles $(\mathbf{p}_i,\mathbf{p}_j,\mathbf{p}_k)$ that could possibly appear during construction of the triangulation and let $w_{i,j}$ be the minimum total weight that can be achieved in triangulating the polygon $\mathbf{p}_i,\ldots,\mathbf{p}_j$, $0 \le i < j < n$. Then $w_{i,j}$ can be computed recursively as

$$w_{i,j} = \min_{i<m<j} w_{i,m} + w_{m,j} + \phi(i,m,j) \quad .$$

The triangulation that minimizes $w_{0,n-1}$ is computed by a dynamic programming algorithm that caches the intermediate values $w_{i,j}$.

Liepa suggests a weight function ϕ that is designed to take into account the dihedral angles between neighboring triangles as well as triangle area. It produces tuples

$$\phi(i,j,k) = (\alpha, A) \quad ,$$

where α is the maximum of the dihedral angles to the neighbors of $(\mathbf{p}_i,\mathbf{p}_j,\mathbf{p}_k)$ and A is its area. Note that this weight function in particular penalizes fold-overs. When comparing different values of ω, a low normal variation is favored over a low area:

$$(\alpha_1,A_1) < (\alpha_2,A_2) \quad :\Leftrightarrow \quad (\alpha_1 < \alpha_2) \vee (\alpha_1 = \alpha_2 \wedge A_1 < A_2)$$

Note that when evaluating ω one has to take into account that the neighboring triangles can either belong to the mesh that surrounds the hole or to the patch that is currently being created. A triangulation of a hole that is produced using this weight function is shown in Fig. 12.

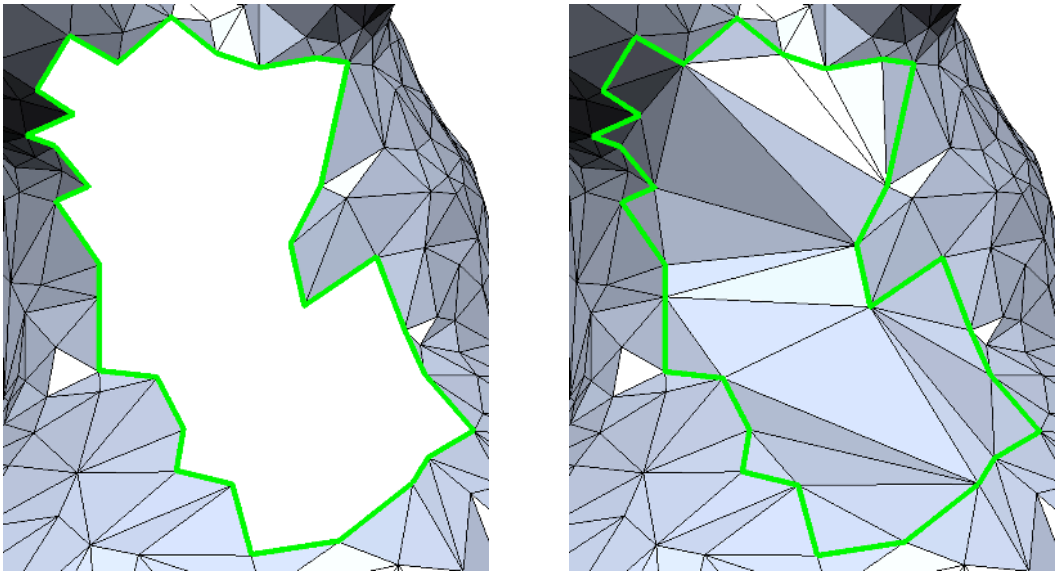


Figure 12: *A hole triangulation that minimizes normal variation and total area.*

To produce a fair hole filling, Liepa suggests to produce a tangent continuous fill-in of minimal thin plate energy: First the holes are identified and filled by a coarse triangulation as described above. These patches are then refined such that their vertex densities and edge lengths match that of the area surrounding the holes, see Section 10. Finally, the patch is smoothed such as to blend with the geometry of the surrounding mesh, see Section 7.

Discussion The algorithm reliably closes holes in models with smooth patches. The density of the vertices matches that of the surrounding surface, see Fig. 13. The complexity of building the initial triangulation is $O(n^3)$, which is sufficient for most holes that occur in practice. However, the algorithm does not check or avoid self intersections and does not detect or incorporate isles into the filling.

The described hole filling is a simple and basic tool. Lévy [Lév03] developed a more advanced method to fill holes, e.g., in incomplete range scans, by extrapolation of the given geometry. This method operates in 3D as well as in a 2D parameter domain and relies on parameterization (Section 8) and smoothing (Section 7).

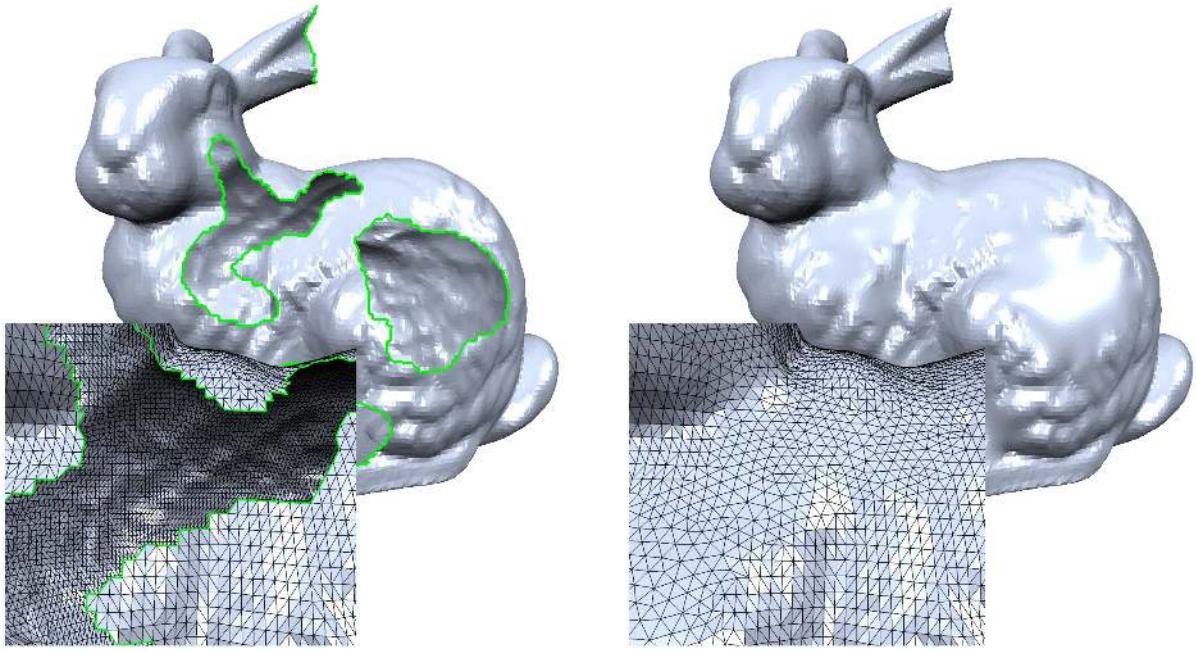


Figure 13: *Liepa's hole filling algorithm. Note that the point density of the fill-in matches that of the surrounding area.*

4.4.3. Conversion to Manifolds

Gueziec et al. propose a method to remove complex edges and singular vertices from non-manifold input models [GTLH01]. The output is guaranteed to be a manifold triangle mesh, possibly with boundaries. As the algorithm operates solely on the connectivity of the input model, it does not suffer from numerical robustness issues. In a preprocessing phase all complex edges and singular vertices are identified. The input is then cut along these complex edges into manifold patches (usually with boundaries). Finally, pairs of matching edges (i.e., edges that have the same endpoints) are identified and – if possible – merged.

In the preprocessing phase the input is split into separate faces and all complex edges are identified by counting the number of adjacent faces: edges with one, two, or more than two adjacent faces are boundary, regular interior or complex respectively. Then the input model is separated into manifold patches along the complex edges by stitching the two adjacent faces of each *interior regular* edge. This method implicitly handles stand-alone and singular vertices.

Gueziec et al. propose two different strategies for stitching further edges: *pinching* and *snapping*. The pinching strategy only stitches along edges that belong to the same connected component. Thus small erroneous connected components are separated from the main part of the model and can be easily detected and removed in a post-processing step. The algorithm iterates once over all boundary vertices. Let v be a boundary vertex, v_p its predecessor and v_n its successor along the boundary. If $v_p = v_n$ the two edges (v_p, v) and (v, v_n) are merged.

In contrast to pinching, the snapping strategy reduces the number of connected components of the model. The basic idea is to locate candidate pairs of boundary edges and to stitch them if a certain stitchability criterion is met. This criterion asserts that after stitching, the model does not contain new complex edges or singular vertices. The snapping strategy can be extended to also allow the stitching of edges that are geometrically close to each other.

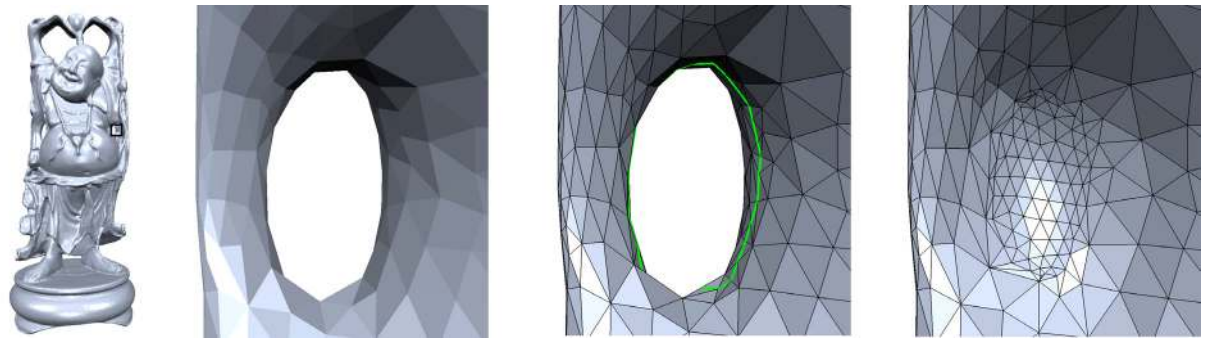


Figure 14: *Left and middle left: The Happy Buddha model contains more than 100 handles. Middle right: A non-separating closed cycle along a handle. Right: The handle was removed by cutting along the non-separating cycle and closing the holes with triangle patches.*

Discussion The scope of this algorithm is limited to the removal of complex edges and singular vertices. This, however, is done efficiently and robustly.

4.4.4. Gap Closing

A number of surface oriented algorithms have been proposed to close the gaps and small overlaps that are typical for triangulated NURBS models.

Barequet and Sharir proposed one of the first algorithms to fill gaps and remove small overlaps [BS95]. The algorithm identifies matching parts of the boundaries by a geometric hashing technique and fills the gaps by patching them with triangle strips or by the technique presented in Section 4.4.2.

Barequet and Kumar propse an algorithm to repair CAD models that identifies and merges pairs of boundary edges [BK97]. For each pair of boundary edges the area between the two edges normalized by the edge lengths is computed. This score measures the geometric error that would be introduced by merging the two edges. Pairs of boundary edges are then iteratively merged in order of increasing score.

Borodin et al. [BNK02] propose an algorithm that snaps boundary vertices to nearby boundary edges. The algorithm is based on a standard mesh-decimation technique, but replaces the vertex-vertex contraction operator by a vertex-edge contraction operator, that operates on boundary vertices v and boundary edges e: Let c be the closest point to v on e. If c is an interior point of e, c is inserted into e by splitting the adjacent triangle in two. Finally, v and c are merged. The cost of a vertex-edge collapse is defined as the distance of v to c. The algorithm maintains a priority queue of vertex/edge pairs and snaps them in order of increasing distance.

Discussion The semantics of these surface oriented algorithms is well defined and they are typically easy to implement. If the input data is well-behaved and the user parameters are chosen in accordance with the error that was accepted during triangulation, they also produce satisfying results. However, there are no guarantees on the quality of the output. Due to the simple heuristics, many artifacts remain unresolved. Therefore, these algorithms are usually run in an interactive loop that allows designers to override the decisions made by the algorithms or to steer the algorithms in a certain direction.

4.4.5. Topology Simplification

Guskov and Wood proposed an algorithm that detects and resolves all handles up to a given size ε in a manifold triangle mesh [GW01]. Handles are removed by cutting the input along a non-separating closed path and sealing the two resulting holes by triangle patches, see Fig. 14.

Given a seed triangle s, the algorithm conquers a geodesic region $\mathcal{R}_\varepsilon(s)$ around s in the order that is given by Dijkstra's algorithm on the dual graph of the input mesh $\mathcal{M}$. Note that Dijkstra's algorithm not only computes the length of a shortest path from each triangle t to the seed s, but it also produces a parent $p(t)$ such that $t, p(t), p^2(t), \ldots, s$ actually *is* a shortest path from t to s.

The boundary of $\mathcal{R}_\varepsilon(s)$ consists of one or more boundary loops. Whenever a boundary loop touches itself along an edge, it is split into two new loops and the algorithm proceeds. However, when two different loops touch along a common edge, a handle is detected. Let t_1 and t_2 be the two triangles that are adjacent to the common edge and $p^{n_1}(t_1) = p^{n_2}(t_2)$ a common ancestor of t_1 and t_2. The closed path

$$p^{n_1}(t_1), \ldots, p(t_1), t_1, t_2, p(t_2), \ldots, p^{n_2}(t_2)$$

is then a cycle of adjacent triangles that stretches around the handle. The input model is cut along this triangle strip and the two boundary loops that are created by this cut are then sealed, e.g., by the method presented in Section 4.4.2.

To detect all handles of $\mathcal{M}$, one has to perform the region growing for all triangles $s \in \mathcal{M}$. Guskov and Wood describe a method to considerably reduce the necessary number of seed triangles and thus are able to significantly speed up the algorithm.

Discussion The proposed method reliably detects small handles up to a user-prescribed size and removes them. However, the algorithm is slow, it does not detect long, thin handles and it cannot guarantee that no self-intersections are created when a handle is removed.

4.5. Volumetric Repair Algorithms

This section presents recent repair algorithms that use an intermediate volumetric representation to implicitly remove the artifacts of a model. This volumetric representation might be as simple as a regular Cartesian grid or as complex as a binary space partition.

4.5.1. Volumetric Repair on Regular Grids

Nooruddin and Turk proposed one of the first volumetric techniques to repair arbitrary models that contain gaps, overlaps and intersections [NT03]. Additionally they employed morphological operators to resolve topological artifacts like holes and handles.

First, the model is converted into a Cartesian voxel grid: A set of projection directions $\{\mathbf{d}_i\}$ is produced, e.g., by subdividing an octahedron or icosahedron. Then the model is projected along these directions onto an orthogonal planar grid. For each grid point $\mathbf{x}$, the algorithm records the first and last intersection point of the ray $\mathbf{x} + \lambda\mathbf{d}_i$ and the input model. A voxel is classified by such a ray to be *inside*, if it lies between these two extreme depth samples, otherwise it is classified as *outside*. The final classification of each voxel is derived from the majority vote of all the rays passing through that voxel. A Marching Cubes algorithm is then used to extract the surface between inside and outside voxels.

In an optional second step, thin handles and holes are removed from the volume by applying *morphological operators* that are also known from image processing [HSZ87]. The *dilation* operator d_ε computes the distance from each outside voxel to the inside component. All voxels that are within a distance of ε to the inside are also set to *inside*. Thus the dilation operator closes small handles and bridges small gaps. The *erosion* operator e_ε works exactly the other way round and removes thin bridges and handles. Usually, dilation and erosion are used in conjunction, $e_\varepsilon \circ d_\varepsilon$ to avoid expansion or shrinkage of the model.

Discussion The classification of inside and outside voxels is rather heuristic and often not reliable. Furthermore, the algorithm is not feature-sensitive.

4.5.2. Volumetric Repair on Adaptive Grids

Bischoff et al. [BPK05] propose an improved volumetric technique to repair arbitrary triangle soups. The user provides an error tolerance ε and a maximum diameter ρ up to which gaps should be closed. The algorithm first creates an adaptive octree representation of the input model where each cell stores the triangles intersecting with it. From these triangles a feature-sensitive sample point can be computed for each cell. Then a sequence of morphological operations is applied to the octree to determine the topology of the model. Finally, the connectivity and geometry of the reconstruction are derived from the octree structure and samples, respectively.

Let us assume that the triangle soup is scaled to fit into the root cell of the octree. We set the maximum depth of the octree cells such that the diameter of the finest level cells is smaller than ε. Each cell stores references to the triangles that intersect it and initially all triangles are associated with the root cell. Then cells that are not yet on maximum depth are recursively split if they either contain a boundary edge or if the triangles within the cell deviate too much from a common supporting plane. Whenever a cell is split, its triangles are distributed to its children. The result is a memory-efficient octree with large cells in planar or empty regions and fine cells along the features and boundaries of the input model.

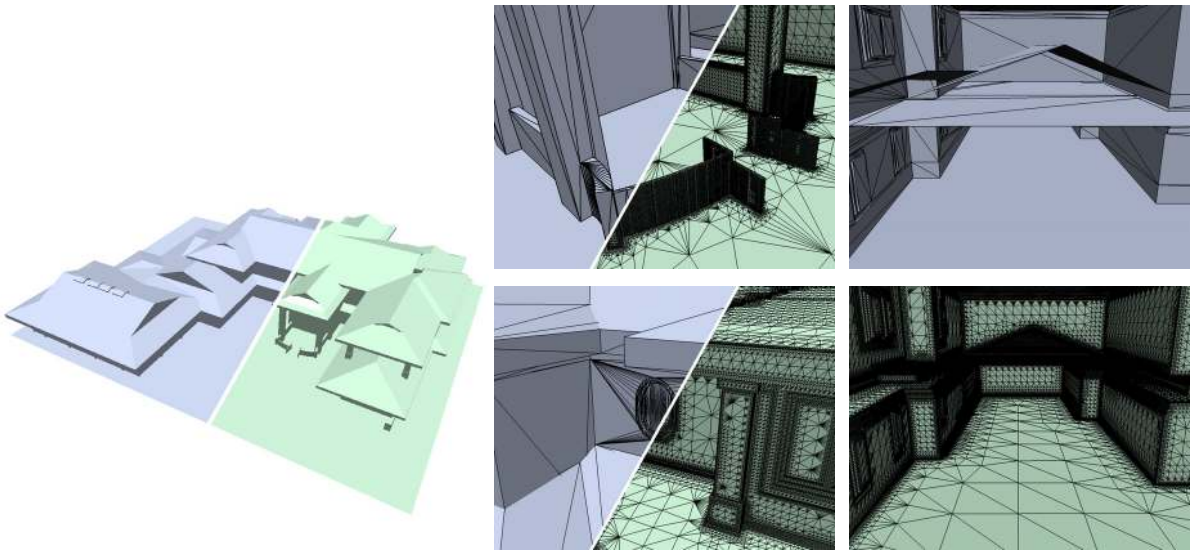
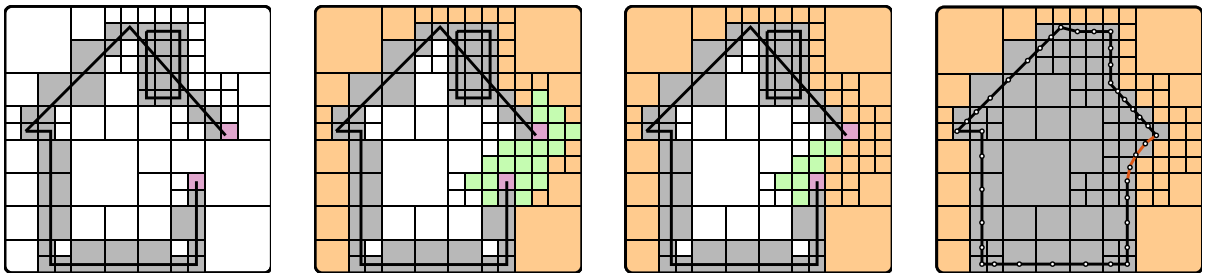


Figure 15: *Reconstruction (green) of a triangle soup (blue). Left: Visually there is no difference between the triangle soup and the reconstruction. Middle: The reconstruction is a watertight mesh that is refined near the model features. Right: The volumetric approach allows to reliably detect and remove excess interior geometry from the input.*

In the second phase, each leaf cell of the octree is classified as being either *inside* or *outside*. First, all cells that contain a boundary of the model are dilated by $n := \rho/\varepsilon$ layers of voxels such that all gaps of diameter $\leq \rho$ are closed. A flood fill algorithm then propagates the outside label from the boundary of the octree into its interior. Finally, the outside component is dilated again by n layers to avoid an expansion of the model.

A Dual Contouring algorithm then reconstructs the interface between the outside and the inside cells by connecting sample points. These sample points are the minimizers of the squared distances to their supporting triangle planes, thus features like edges and corners are well preserved (see also Section 9 on quadric error metrics). If no such planes are available (e.g., because the cell was one of the dilated boundary cells), the corresponding sample point is smoothed in a post-processing step (Section 7).



Adaptive octree, boundary cells are marked red

Dilated boundary (green) and outside component (orange)

Outside component dilated back into the boundary cells.

Final reconstruction

Discussion As this algorithm is based on a volumetric representation, it produces guaranteed manifold output (Fig. 15). Features are also well preserved. However, despite the adaptive octree, the resolution of the reconstruction is limited.

4.5.3. Volumetric Repair with BSP Trees

A unique method for converting triangle soups to manifold surfaces was presented by Murali and Funkhouser [MF97]. The polygon soup is first converted into a BSP tree, the supporting planes of the input polygons serve as splitting planes for the space partition. The leaves of the tree thus correspond to closed convex spatial regions C_i. For each C_i a *solidity coefficient* $s_i \in [-1,1]$ is computed. Negative solidity coefficients designate empty regions, while positive coefficients designate solid regions.

All unbounded cells naturally lie outside the object and thus are assigned a solidity value of -1. Let C_i be a bounded cell and let $\mathcal{N}(i)$ be the indices of all its face neighbors. Thus for each $j \in \mathcal{N}(i)$ the intersection $P_{ij} = C_i \cap C_j$ is a planar polygon that might be partially covered by the input geometry. For each $j \in \mathcal{N}(i)$ let t_{ij} be the transparent area, o_{ij} the opaque area and a_{ij} the total area of P_{ij}. The solidity s_i is then related to the solidities s_j of its face neighbors by

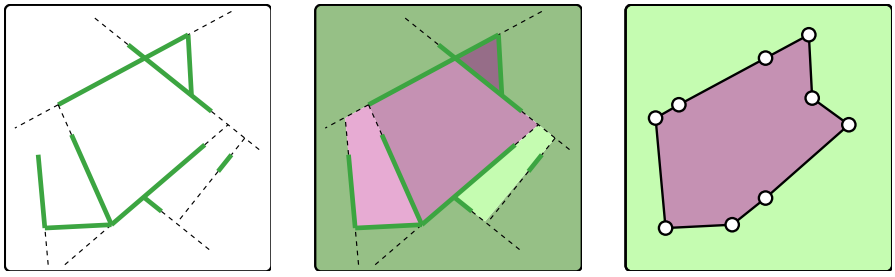
$$s_i = \frac{1}{A_i} \sum_{j \in \mathcal{N}(i)} (t_{ij} - o_{ij}) s_j \quad , \tag{2}$$

where $A_i = \sum a_{ij}$ is the total area of the boundary of C_i. Note the two extreme cases: If P_{ij} is fully transparent, $t_{ij} - o_{ij} = a_{ij} > 0$ the correlation of s_i and s_j is positive, indicating that both cells should be solid or both cells should be empty. If, on the other hand, P_{ij} is fully opaque, $t_{ij} - o_{ij} = -a_{ij} < 0$, the negative correlation indicates that one cell should be solid and the other empty. Collecting all equations Eq. (2) leads to a sparse linear system

$$\mathbf{M}[s_1, \ldots, s_n]^T = \mathbf{b} \quad ,$$

which can be solved efficiently using an iterative solver (Section 12). It can be shown that $\mathbf{M}$ is always invertible and that the solidity coefficients of the solution in fact lie in the range $[-1,1]$.

Finally, the surface of the solid cells is extracted by enumerating all neighboring pairs of leaf cells (C_i, C_j). If one of them is empty and the other is solid, the corresponding (triangulated) boundary polygon P_{ij} is added to the reconstruction.



BSP *Solidity coefficients* *Reconstruction*

Discussion This method does not need (but also cannot incorporate) any user parameters to automatically produce watertight models. The output might contain complex edges and singular vertices, but these can be removed using the algorithm presented in Section 4.4.3. Unfortunately, a robust and efficient computation of the combinatoric structure of the BSP is hard to accomplish.

4.5.4. Volumetric Repair on the Dual Grid

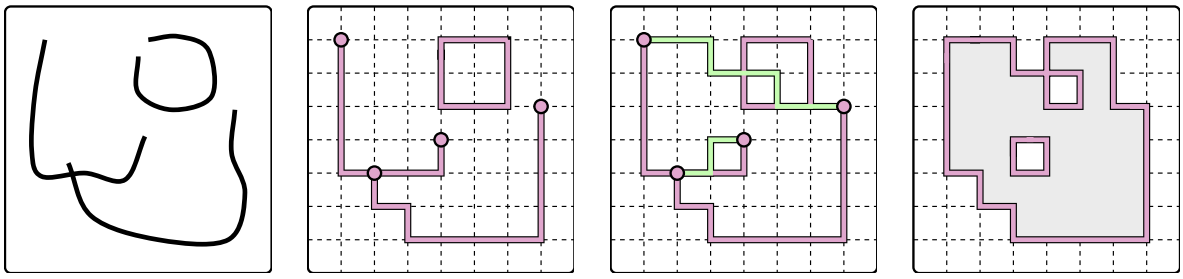
Ju proposes an interesting volumetric algorithm to repair arbitrary triangle soups [Ju04]. While the boundary loops are explicitly traced and filled, the overall scheme is volumetric.

The algorithm first approximates the input model by a subset $\mathcal{F}$ of the faces of a Cartesian grid. For memory efficiency, these faces are stored in an adaptive octree. Additionally, a sample point (and possibly a normal) from the input model are associated with each face, to allow for a more accurate reconstruction. The boundary $\partial\mathcal{F}$ of $\mathcal{F}$ is defined to be the subset of the grid edges that are incident to an odd number of faces in $\mathcal{F}$. Note that if $\mathcal{G}$ is another face set, such that $\partial\mathcal{G} = \partial\mathcal{F}$, then $\partial(\mathcal{F} \ominus \mathcal{G}) = \emptyset$. Here, the *symmetric difference* (xor) of two sets $\mathcal{A}$ and $\mathcal{B}$ is defined as $\mathcal{A} \ominus \mathcal{B} = (\mathcal{A} \cup \mathcal{B}) \setminus (\mathcal{A} \cap \mathcal{B})$. Also, if $\partial\mathcal{F} = \emptyset$ then the grid voxels can be two-colored by *inside* and *outside* labels such that two adjacent voxels have the same label, while two voxels that are separated by a face of $\mathcal{F}$ have different labels.

For each boundary loop B_i of $\mathcal{F}$, the algorithm constructs a minimal face set $\mathcal{G}_i$ such that $\partial\mathcal{G}_i = B_i$. Then $\mathcal{F}$ is replaced by

$$\mathcal{F}' = \mathcal{F} \ominus \mathcal{G}_1 \ominus \cdots \ominus \mathcal{G}_n,$$

thus $\partial\mathcal{F}' = \emptyset$. As voxels at the corners of the bounding box are known to be *outside*, they are used as seeds for propagating the *inside/outside* information over the grid. The interface between *inside* and *outside* voxels is then extracted using either a Marching Cubes or a Dual Contouring algorithm.



Input geometry *Face set* *Patches* *Reconstruction*

Discussion Ju's algorithm uses a volumetric representation and thus produces guaranteed manifold output. The algorithm is memory-less, i.e., insensitive to the size of the input and thus can process arbitrarily large meshes out-of-core. On the other hand, the algorithm has problems handling thin structures. In particular, if the discrete approximation that is used in the hole filling step overlaps with the input geometry, this part of the mesh may disappear or be shattered into many pieces. Due to the volumetric representation the whole input model is resampled and the output might become arbitrarily large for fine resolutions.

4.5.5. Extending MLS to Triangle Soups

Shen et al. propose a volumetric repair algorithm that operates on arbitrary triangle soups [SOS04]. It is a generalization of the moving least squares approach that can for instance be used for reconstructing geometry from point clouds. Instead of approximating positional information only, they also incorporate normal constraints into the reconstruction and thus avoid an oscillating solution. The details of this algorithm are involved and we restrict ourselves to the basic ideas.

Let $t_1,\ldots,t_N$ be a set of triangles and let $\mathbf{n}_1,\ldots,\mathbf{n}_N$ be their normals. The goal is to generate a function $s:\mathbb{R}^3\rightarrow\mathbb{R}$ whose zero level-set matches $t_1,\ldots,t_N$ as close as possible. An arbitrary contouring algorithm can then be used to extract a reconstruction of $t_1,\ldots,t_N$ from s. For a single triangle t_k, the corresponding function s_k is of course linear

$$s_k(\mathbf{x}) = \mathbf{n}_k^T(\mathbf{x}-\mathbf{q}_k)$$

where $\mathbf{q}_k$ is an arbitrary point on t_k.

The function s is expressed as a linear combination of a set $\mathbf{b}(\mathbf{x}) = [b_1(\mathbf{x}),\ldots,b_M(\mathbf{x})]^T$ of basis functions, thus

$$s(\mathbf{x}) = \mathbf{b}(\mathbf{x})^T\mathbf{c} \tag{3}$$

for some vector $\mathbf{c}$. So-called *radial basis functions* are a common choice for $\mathbf{b}(\mathbf{x})$, but they lead to a large linear system that is hard to solve efficiently. Instead, Chen et al. follow an approach that is known as *Moving Least Squares (MLS),* see [AK04] and references therein.

The idea of MLS is to only use a very limited set of basis functions, typically $\mathbf{b}(x,y,z) = [1,x,y,z]^T$. To compensate the limited degrees of freedom in choosing a small number of basis functions, Eq. (3) is made dependent on the point $\mathbf{x}_0$ at which one plans to evaluate and triangles that are close to $\mathbf{x}_0$ are given a greater weight than those that are far away. Thus, for a fixed point $\mathbf{x}_0$ one seeks to minimize

$$\sum_k \int_{t_k} w_{\mathbf{x}_0}(\mathbf{x})^2 \left(\mathbf{b}(\mathbf{x})^T\mathbf{c}_{\mathbf{x}_0} - s_k(\mathbf{x})\right)^2 d\mathbf{x} \tag{4}$$

with respect to $\mathbf{c}_{\mathbf{x}_0}$ where the weight function $w_{\mathbf{x}_0}(\mathbf{x})$ is chosen as

$$w_{\mathbf{x}_0}(\mathbf{x}) = \frac{1}{||\mathbf{x}-\mathbf{x}_0||^2+\varepsilon^2}$$

Setting the derivative of Eq. (4) w.r.t. $\mathbf{c}_{\mathbf{x}_0}$ to zero leads to a 4×4 linear system

$$\sum_k \mathbf{A}_k\mathbf{c}_{\mathbf{x}_0} = \sum_k \mathbf{a}_k$$

where

$$\mathbf{A}_k = \int_{t_k} w_{\mathbf{x}_0}(\mathbf{x})^2 \mathbf{b}(\mathbf{x})\mathbf{b}(\mathbf{x})^T d\mathbf{x} \quad \text{and} \quad \mathbf{a}_k = \int_{t_k} w_{\mathbf{x}_0}(\mathbf{x})^2 \mathbf{b}(\mathbf{x})^T s_k(\mathbf{x}) d\mathbf{x}$$

Thus, the function s is given as $s(\mathbf{x}_0) = \mathbf{b}(\mathbf{x}_0)^T \mathbf{c}_{\mathbf{x}_0}$.

The integrands that appear in $\mathbf{A}_k$ and $\mathbf{a}_k$ are rational polynomials and Chen et al. devise a suitable numerical integration scheme to evaluate them. They also propose a method to speed up the evaluation.

Discussion This algorithm produces watertight models and automatically bridges gaps in an intuitive way. The method can be modified to produce hulls of a different geometric complexities that enclose the input model. These hulls can then be used, e.g., for fast collision detection tests. Unfortunately, the algorithm does not cope well with models that contain interior excess geometry.

5. Discrete Curvatures

This section introduces differential properties of 2-manifold surfaces and discusses the corresponding approximations on arbitrary triangle meshes. These discrete differential operators play a central role in many mesh processing applications such as surface smoothing (Section 7), parameterization (Section 8), or mesh deformation (Section 11).

5.1. Differential Geometry

We provide a brief review of important concepts from differential geometry that form the basis of the definition of the discrete operators on triangle meshes. For an in-depth discussion we refer to standard textbooks such as [dC76].

Let a continuous surface $\mathcal{S} \subset \mathbb{R}^3$ be given in parametric form as

$$\mathbf{x}(u,v) = \begin{pmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{pmatrix}, \ (u,v) \in \mathbb{R}^2,$$

where x, y, z are (sufficiently often) differentiable functions in u and v. The partial derivatives $\mathbf{x}_u$ and $\mathbf{x}_v$ span the tangent plane to $\mathcal{S}$ at $\mathbf{x}$. Assuming a regular parameterization, i.e., $\mathbf{x}_u \times \mathbf{x}_v \neq \mathbf{0}$, the normal vector is given as $\mathbf{n} = (\mathbf{x}_u \times \mathbf{x}_v)/\|\mathbf{x}_u \times \mathbf{x}_v\|$.

The *first fundamental form* of $\mathbf{x}$ is given by the matrix

$$\mathbf{I} = \begin{bmatrix} E & F \\ F & G \end{bmatrix} := \begin{bmatrix} \mathbf{x}_u^T \mathbf{x}_u & \mathbf{x}_u^T \mathbf{x}_v \\ \mathbf{x}_u^T \mathbf{x}_v & \mathbf{x}_v^T \mathbf{x}_v \end{bmatrix}, \tag{5}$$

which defines an inner product on the tangent space of $\mathcal{S}$. The corresponding arc element ds is given as

$$ds^2 = Edu^2 + 2Fdudv + Gdv^2.$$

The area element can be derived as

$$dA = \sqrt{EG - F^2}dudv.$$

The *second fundamental form* is defined as

$$\mathbf{II} = \begin{bmatrix} e & f \\ f & g \end{bmatrix} := \begin{bmatrix} \mathbf{x}_{uu}^T \mathbf{n} & \mathbf{x}_{uv}^T \mathbf{n} \\ \mathbf{x}_{uv}^T \mathbf{n} & \mathbf{x}_{vv}^T \mathbf{n} \end{bmatrix}. \tag{6}$$

Alternatively, $\mathbf{II}$ can be expressed using the identities $\mathbf{x}_{uu}^T \mathbf{n} = -\mathbf{x}_u^T \mathbf{n}_u$, $\mathbf{x}_{uv}^T \mathbf{n} = \mathbf{x}_{vu}^T \mathbf{n} = -\frac{1}{2}(\mathbf{x}_u^T \mathbf{n}_v + \mathbf{x}_v^T \mathbf{n}_u)$, and $\mathbf{x}_{vv}^T \mathbf{n} = -\mathbf{x}_v \mathbf{n}_v$.

The symmetric bilinear first and second fundamental forms allow to measure length, angles, area, and curvatures on the surface.

Let $\mathbf{t} = a\mathbf{x}_u + b\mathbf{x}_v$ be a unit vector in the tangent plane at $\mathbf{p}$, represented as $\bar{\mathbf{t}} = (a,b)^T$ in the local coordinate system. The *normal curvature* $\kappa_n(\bar{\mathbf{t}})$ is the curvature of the planar curve that results from intersecting $\mathcal{S}$ with the plane through $\mathbf{p}$ spanned by $\mathbf{n}$ and $\mathbf{t}$. The normal curvature in direction $\bar{\mathbf{t}}$ can be expressed in terms of the fundamental forms as

$$\kappa_n(\bar{\mathbf{t}}) = \frac{\bar{\mathbf{t}}^T \mathbf{II}\, \bar{\mathbf{t}}}{\bar{\mathbf{t}}^T \mathbf{I}\, \bar{\mathbf{t}}} = \frac{ea^2 + 2fab + gb^2}{Ea^2 + 2Fab + Gb^2}$$

The minimal normal curvature κ_1 and the maximal normal curvature κ_2 are called *principal curvatures*. The associated tangent vectors $\mathbf{t}_1$ and $\mathbf{t}_2$ are called *principal directions* and are always perpendicular to each other.

The principal curvatures are also obtained as eigenvalues of the *Weingarten curvature matrix* (or second fundamental tensor)

$$\mathbf{W} := \frac{1}{EG - F^2} \begin{bmatrix} eG - fF & fG - gF \\ fE - eF & gE - fF \end{bmatrix}. \tag{7}$$

$\mathbf{W}$ represents the Weingarten map or shape operator, which measures the directional derivative of the normal, i.e. $\mathbf{W}\bar{\mathbf{t}} = \frac{\partial}{\partial \bar{\mathbf{t}}}\mathbf{n}$. This allows the normal curvature to be expressed as

$$\kappa_n(\bar{\mathbf{t}}) = \bar{\mathbf{t}}^T \mathbf{W} \bar{\mathbf{t}}\,.$$

With a local coordinate system defined by the principal directions $\mathbf{t}_1$ and $\mathbf{t}_2$, $\mathbf{W}$ is a diagonal matrix, or in general

$$\mathbf{W} = \begin{bmatrix} \bar{\mathbf{t}}_1 & \bar{\mathbf{t}}_2 \end{bmatrix} \begin{bmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{bmatrix} \begin{bmatrix} \bar{\mathbf{t}}_1 & \bar{\mathbf{t}}_2 \end{bmatrix}^{-1} . \tag{8}$$

Then the normal curvature can also be written as

$$\kappa_n(\bar{\mathbf{t}}) = \kappa_n(\phi) = \kappa_1 \cos^2\phi + \kappa_2 \sin^2\phi, \tag{9}$$

where ϕ is the angle between $\bar{\mathbf{t}}$ and $\bar{\mathbf{t}}_1$ (Euler's theorem).

The *curvature tensor* $\mathbf{T}$ is expressed as a symmetric 3×3 matrix with the eigenvalues κ_1, κ_2, 0 and the corresponding eigenvectors $\mathbf{t}_1$, $\mathbf{t}_2$, $\mathbf{n}$. The tensor $\mathbf{T}$ measures the change of the unit normal with respect to a tangent vector $\mathbf{t}$ independently of the parameterization. It can be constructed as

$$\mathbf{T} = \mathbf{PDP}^{-1} ,$$

with $\mathbf{P} = [\mathbf{t}_1, \mathbf{t}_2, \mathbf{n}]$ and $\mathbf{D} = \mathrm{diag}(\kappa_1, \kappa_2, 0)$.

The *Gaussian curvature* K is defined as the product of the principal curvatures, i.e.,

$$K = \kappa_1 \kappa_2 = \det(\mathbf{W}), \tag{10}$$

the *mean curvature* H as the average of the principal curvatures, i.e.,

$$H = \frac{\kappa_1 + \kappa_2}{2} = \frac{1}{2}\mathrm{trace}(\mathbf{W}). \tag{11}$$

The mean curvature can alternatively be expressed as the (continuous) average of the normal curvatures

$$H = \frac{1}{2\pi} \int_0^{2\pi} \kappa_n(\phi) d\phi . \tag{12}$$

In differential geometry, properties that only depend on the first fundamental form are called *intrinsic*. Intuitively, the intrinsic geometry of a surface can be perceived by 2D creatures that live on the surface without knowledge of the third dimension. Examples include length and angles of curves on the surface. Gauss' famous Theorema Egregium states that the Gaussian curvature is invariant under local isometries and as such also intrinsic to the surface [dC76]. Note that the term "intrinsic" is often also used to denote independence of a particular parametrization.

5.1.0.1. Laplace Operator. The following sections will make extensive use of the *Laplace operator* Δ, resp., the *Laplace-Beltrami operator* $\Delta_{\mathcal{S}}$. In general, the Laplace operator is defined as the divergence of the gradient, i.e. $\Delta = \nabla^2 = \nabla \cdot \nabla$. In Euclidean space this second order differential operator can be written as the sum of second partial derivatives

$$\Delta f = \mathrm{div} \nabla f = \sum_i \frac{\partial^2 f}{\partial x_i^2} \tag{13}$$

with Cartesian coordinates x_i. The *Laplace-Beltrami operator* extends this concept to functions defined on surfaces. For a given function f defined on a manifold surface $\mathcal{S}$ the Laplace-Beltrami is defined as

$$\Delta_{\mathcal{S}} f = \mathrm{div}_{\mathcal{S}} \nabla_{\mathcal{S}} f,$$

which requires a suitable definition of the divergence and gradient operators on manifolds (see [dC76] for details). Applied to the coordinate function $\mathbf{x}$ of the surface the Laplace-Beltrami operator evaluates to the mean curvature normal

$$\Delta_{\mathcal{S}} \mathbf{x} = -2H\mathbf{n}.$$

Note that the Laplace-Beltrami operator is an intrinsic property that only depends on the metric tensor of the surface and is thus independent of a specific parameterization.

5.2. Discrete Differential Operators

The differential properties defined in the previous section require a surface to be sufficiently often differentiable, e.g., the definition of the curvature tensor requires the existence of second derivatives. Since polygonal meshes are piecewise linear surfaces, the concepts introduced above cannot be applied directly. The following definitions of discrete differential operators are thus based on the assumption that meshes can be interpreted as piecewise linear approximations of smooth surfaces. The goal is then to compute approximations of the differential properties of this underlying surface directly from the mesh data.

Different approaches have been proposed in recent years and we will provide a brief overview and comparison of the different techniques. For details we refer to the references given throughout the text and to the survey [Pet01].

The general idea of the techniques described below is to compute discrete differential properties as spatial averages over a local neighborhood $\mathcal{N}(\mathbf{x})$ of a point $\mathbf{x}$ on the mesh. Often $\mathbf{x}$ coincides with a mesh vertex v_i, and n-ring neighborhoods $\mathcal{N}_n(v)$ or local geodesic balls are used as the averaging domain. The size of the local neighborhood critically affects the stability and accuracy of the discrete operators. The bigger the neighborhoods the more smoothing is introduced by the averaging operation, which makes the computations more stable in the presence of noise. For clean data sets, small neighborhoods, e.g., one-rings, are typically preferable, as they more accurately capture fine-scale variations of differential properties.

In order to estimate the curvature tensor at a vertex a certain neighborhood of this vertex is considered, typically its one-ring. A common approach is to first discretize the normal curvature along edges. Given is an edge (v_i, v_j), vertex positions $\mathbf{p}_i$, $\mathbf{p}_j$, and the normal $\mathbf{n}_i$, then

$$\kappa_{ij} \;=\; 2\frac{(\mathbf{p}_j-\mathbf{p}_i)\mathbf{n}_i}{\|\mathbf{p}_j-\mathbf{p}_i\|^2} \tag{14}$$

provides an approximation of the normal curvature at $\mathbf{p}_i$ in the tangent direction which results from projecting $\mathbf{p}_i$ and $\mathbf{p}_j$ into the tangent plane defined by $\mathbf{n}_i$. This expression can be interpreted geometrically as fitting the osculating circle interpolating $\mathbf{p}_i$ and $\mathbf{p}_j$ with normal $\mathbf{n}_i$ at $\mathbf{p}_i$ (cf. [MS92]). Alternatively, the equation can be derived from discretizing the curvature of a smooth planar curve (see [Tau95a]). With estimates κ_{ij} of the normal curvature for all edges incident to vertex v_i, Euler's theorem (9) can be applied to relate the κ_{ij} to the unknown principal curvatures (and principal directions). Then approximations to the principal curvatures can be obtained either directly as functions of the eigenvalues of a symmetric matrix ([Tau95a, PKS*01]) or from solving a least-squares problem ([MS92, MDSB03]). Alternatively, [WB01] apply the trapezoid rule to get a discrete approximation of (12), which provides the mean curvature H, the Gaussian curvature K is obtained from a similar integral over κ_n^2, and the principal curvatures are then obtained from equations (10), (11). Exact quadrature formulas for curvature estimation are provided in [LBS05].

A straightforward approach to estimating local surface properties uses a local higher-order reconstruction of the surface, followed by analytical evaluation of the desired properties on the reconstructed surface patch. Local surface patches, typically bivariate polynomials of low degree, are fitted to sample points [CP03, Pet01, WW94] and possibly normals [GI04] within a local neighborhood. Special care is required to ensure good conditioning of the arising local least-squares problems which depend on local parameterization. A (rather expensive) global fitting of an implcit surface is applied in [OB04].

Taubin [Tau95b] proposed the uniform discretization of the Laplace-Beltrami operator

$$\Delta_{uni} f(v) \;:=\; \frac{1}{|\mathcal{N}_1(v)|}\sum_{v_i\in\mathcal{N}_1(v)} \left(f(v_i)-f(v)\right) \quad , \tag{15}$$

where the sum is taken over all one-ring neighbors $v_i \in \mathcal{N}_1(v)$ (cf. Fig. 16). 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. For example, when smoothing a planar (and hence perfectly smooth) triangulation, 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 Laplace-Beltrami of the surface (which should be parallel to the surface normal: $\Delta_{\mathcal{S}}\mathbf{p} = -2H\mathbf{n}$). A better (and the current standard) discretization was proposed in [PP93, DMSB99, MDSB03]:

$$\Delta_{\mathcal{S}} f(v) \;:=\; \frac{2}{A(v)}\sum_{v_i\in\mathcal{N}_1(v)} (\cot\alpha_i+\cot\beta_i)\left(f(v_i)-f(v)\right), \tag{16}$$

where $\alpha_i = \angle(\mathbf{p}(v),\mathbf{p}(v_{i-1}),\mathbf{p}(v_i))$, $\beta_i = \angle(\mathbf{p}(v),\mathbf{p}(v_{i+1}),\mathbf{p}(v_i))$, and $A(v)$ denotes the Voronoi area around the vertex v as shown in Fig. 16 (for an exact definition of the Voronoi region area see [MDSB03]). The same approach yields a discrete estimate for Gaussian curvature as

$$K(v) = \frac{1}{A(v)}\left(2\pi - \sum_{v_i\in\mathcal{N}_1(v)}\theta_i\right), \tag{17}$$

where the angles of the incident triangles at vertex v are denoted by θ_i. This formula is a direct consequence of the Gauss-Bonnet theorem. Given the mean curvature normal as defined in (16) and the approximation of the Gaussian curvature of (17), the principal curvatures can be computed from (10) and (11) as

$$\kappa_{1,2}(v) = H(v) \pm \sqrt{H(v)^2 - K(v)}$$

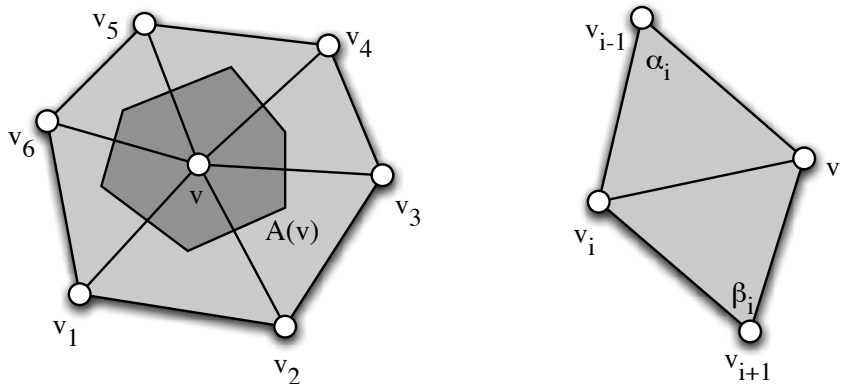


Figure 16: *The Laplace-Beltrami $\Delta_{\mathcal{S}} f(v)$ of a vertex $v \in \mathcal{V}$ is computed by a linear combination of its function value $f(v)$ and those of its one-ring neighbors $f(v_i)$. The corresponding weights are given by the cotangent values of α_i and β_i and the Voronoi area $A(v)$.*

where $H(v) = \frac{1}{2}\|\Delta_{\mathcal{S}}\mathbf{p}(v)\|$.

Eq. (16) is probably the most widely used discretization of the Laplace-Beltrami for triangle meshes and is typically applied for various geometry processing operations, such as surface smoothing (Section 7), parameterization (Section 8), and shape modeling (Section 11). However, there are some disadvantages of the cotangent formula of (16):

- The cotangent weights $\omega_i = \cot\alpha_i + \cot\beta_i$ become negative if $\alpha_i + \beta_i > \pi$. This is well-known and can lead to flipped triangles in certain applications, e.g., when computing a parameterization (see Section 8).
- The definition of the Laplace-Beltrami is not purely intrinsic, i.e., its evaluation can lead to different results even for two isometric surfaces, if their triangulation is different (see [BS05a]).

The first point can possibly be fixed by using different weights. In [ZRS05a] the positive mean value coordinates [Flo03a] are interpreted as an alternative, less accurate discretization of the Laplace-Beltrami operator where integration over the Voronoi area is replaced by integration over circle areas.

Bobenko and Springborn [BS05a] propose an alternative definition that addresses these shortcomings for the case of piecewise flat surfaces, i.e., 2-dimensional manifolds that are equipped with a metric that is flat except at isolated points. The resulting formula is the same as (16), but with respect to an intrinsic Delaunay triangulation of the simplicial surface. For a piecewise flat surface, this triangulation is unique, which makes the evaluation of the discrete Laplace-Beltrami operator independent of the specific tessellation of the mesh. In addition, the Delaunay property guarantees positive weights by construction. Computing the intrinsic Laplace-Beltrami requires to first compute the restricted Delaunay triangulation using an edge flipping algorithm, which is guaranteed to converge. Thus this approach is computationally more involved, in particular for applications that iteratively modify the vertex positions, e.g., curvature flow (Section 7), where the re-tessellation is required after each time step.

Rusinkiewicz proposed a scheme that approximates the curvature tensor using finite differences of vertex normals [Rus04]. As discussed above the curvature tensor measures the change of the normal along the tangent directions. For a given triangle three such directions are given by the triangle edges. The change of normals along each of these edges can be approximated from the difference of the normals of the corresponding vertices. The resulting set of linear constraints on the elements of the curvature tensor can be used in a least-squares optimization to obtain a per-face estimate. The approximation of the curvature tensor for a vertex is then computed using weighted averaging of all per-face estimates of the one-ring based on an appropriate coordinate transformation as discussed in [Rus04]. The paper also shows how this approach can be extended to higher order derivatives. Since the per-face estimates depend on vertex normals that are computed by standard weighted averaging of one-ring face normals, the averaging domain of this method is the two-ring neighborhood. As such, the results produced by this method are somewhat more stable for noisy data. The computation is efficient however, since it can be performed using two passes over the one-rings of the mesh.

In [TRZS04] the piecewise linear surface is considered together with a piecewise linear normal field. Their discrete derivatives define the Weingarten map (7) and hence the tensor of curvature. The precomputed normal field replaces the second order derivatives, which are not available for piecewise linear functions. This idea is motivated by Phong-shading well-known in computer graphics, and similarly the inherent inconsistencies lead to artifacts — the Weingarten matrix is not symmetric anymore

— and hence approximation errors. However, [TRZS04] show convergence to curvatures of smooth surfaces and the errors are small enough to be competitive with other methods. The method yields a piecewise function for the curvature tensor which varies across faces as normals are interpolated. Gaussian and mean curvatures can be written as simple expressions of certain determinants. Evaluation is purely local and efficient, as curvature estimates at vertices are obtained by averaging.

Cohen-Steiner and Morvan [CM03] (see also [ACD*03] and [HP04]) propose a method for estimating the curvature tensor by averaging a line density of tensors defined on each edge of the mesh. This method is derived from the concept of normal cycles, which has been introduced to provide a unified way to define curvature for booth smooth and polygonal surfaces. It includes a proof of convergence under certain sampling conditions based on measure theory. Intuitively, a curvature tensor can be defined for an edge by assigning a minimum curvature along the edge and a maximum curvature across the edge. Averaging over the local neighborhood region $\mathcal{N}(v)$ yields a simple summation formula over the edges intersecting $\mathcal{N}(v)$:

$$\mathbf{C}(v) = \frac{1}{|\mathcal{N}(v)|} \sum_{\mathbf{e} \in \mathcal{N}(v)} \beta(\mathbf{e}) \, \|\mathbf{e} \cap \mathcal{N}(v)\| \, \bar{\mathbf{e}} \bar{\mathbf{e}}^T,$$

where $|\mathcal{N}(v)|$ denotes the surface area of the local neighborhood around v, $\beta(\mathbf{e})$ is the signed dihedral angle between the normals of the two incident faces, $\|\mathbf{e} \cap \mathcal{N}(v)\|$ is the length of the part of the edge $\mathbf{e}$ that is contained in $\mathcal{N}(v)$, and $\bar{\mathbf{e}} = \mathbf{e}/\|\mathbf{e}\|$. The local neighborhood $\mathcal{N}(v)$ is typically chosen to be the one- or two-ring of the vertex v, but can also be computed as a local geodesic disk, i.e., all points on the mesh that are within a certain (geodesic) distance d from v. This can be more appropriate for non-uniformly tessellated surface, where the size of n-ring neighborhoods $\mathcal{N}_n(v)$ can vary significantly over the mesh. As noted in [Rus04], tensor averaging can yield inaccurate results for low-valence vertices and small, e.g., one-ring, neighborhoods.

6. Mesh Quality

This section provides a brief overview of methods used to interactively evaluate the quality of triangle meshes. The techniques discussed here are adapted from smooth free-form surfaces (e.g. NURBS), and are mainly used to visualize surface quality in order to detect surface defects. Different applications may require different quality criteria. We distinguish between *smoothness* and *fairness*. While the former denotes the continuous differentiability (C^k) of a surface, e.g., C^2 for cubic splines, the latter is a more abstract concept required for high-quality surface design. Note that smoothness and fairness are not always used consistently. For example, surface smoothing typically denotes the process of improving the fairness of a surface (see Section 7).

A surface may be smooth in a mathematical sense but still unsatisfactory from an aesthetical point of view. Fairness is an aesthetic measure of "well-shapedness" and therefore more difficult to define in technical terms than smoothness (distribution vs. variation of curvature) [BAFS94]. An important rule is the so called *principle of simplest shape* that is derived from fine arts. A surface is said to be well-shaped, if it is simple in design and free of unessential features. So a *fair* surface meets the mathematically defined goals (e.g. interpolation, continuity), while obeying this design principle. The most common measures for fairness are motivated by physical models like the strain energy of a thin plate

$$\int_{\mathcal{S}} \kappa_1^2 + \kappa_2^2 \, dA,$$

or are defined in terms of differential geometry, like the variation of curvature

$$\int_{\mathcal{S}} \left(\frac{\partial \kappa_1}{\partial \mathbf{t}_1} \right)^2 + \left(\frac{\partial \kappa_2}{\partial \mathbf{t}_2} \right)^2 dA,$$

with principal curvatures κ_i and principal directions $\mathbf{t}_i$ (see Section 5). In general, some surface energy is defined that quantifies surface fairness, and curvature is used to express these terms as it is independent of the special parameterization of a surface. A fair surface is then designed by minimizing these energies (cf. Section 7). Our current goal is not to improve, but to check surface quality, so we need to visualize these energies. Note that there are also different characterizations of fairness, such as aesthetical shape of isophotes/reflection lines [YBP97].

Another important aspect of mesh quality is triangle shape. Some applications require "well shaped" triangles, e.g., simulations using *Finite Element Methods* (FEM). This requires constraints on shape parameters such as angles and area, which will also be discussed in Section 10.

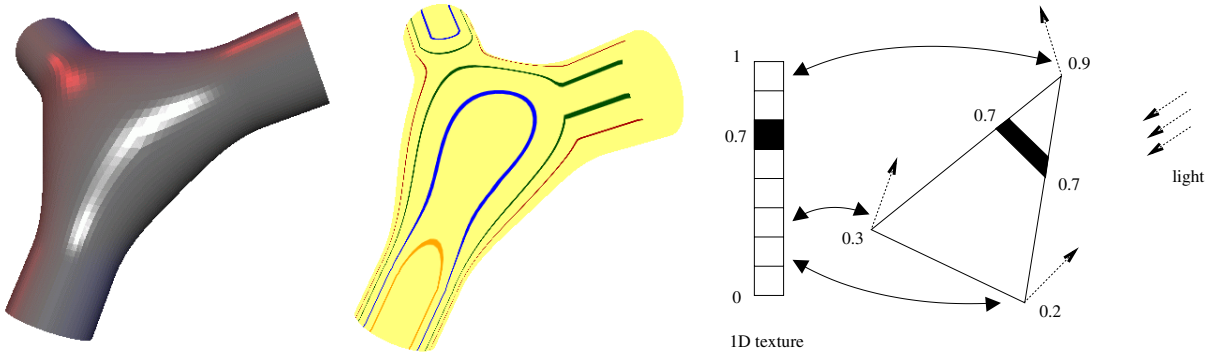


Figure 17: *Isophotes. The center part of the surface was blended between the three tubes using C^1 boundary conditions. The discontinuities of the curvature at the joints are hard to detect from the flat shaded image (left), but clearly visualized by isophotes (middle) since C^1 blends cause C^0 isophotes. The right image sketches the rendering of isophotes with a 1D-texture: The illumination values are calculated for the vertices of a triangle from vertex normals and the light direction. These values are used as texture coordinates. The texel denoting the iso-value is colored black. Iso-lines are interpolated within the triangle.*

6.1. Visualizing smoothness

In order to interactively visualize surface quality, graphics hardware support should be exploited whenever possible. A given surface is tessellated into a set of triangles for rendering (in contrast to more involved rendering techniques like ray-tracing). Since a mesh can be interpreted as an accurate tessellation of, e.g., a set of NURBS patches, the same techniques for quality control can be used that are applied for smooth surfaces [HHS*92].

Specular shading The simplest visualization technique is to use standard lighting and shading (Phong illumination model, flat- or Gouraud shading) as provided by the graphics subsystem. The local illumination of a vertex depends on the position of the light sources, on the surface normal, and on the view point/direction. This approach to surface interrogation is the most straightforward one, but it is difficult to find minor perturbations of a surface (cf. Fig. 17, left).

Isophotes Isophotes are lines of constant illumination on a surface. For a Lambertian surface with purely diffuse reflection, isophotes are independent of the view point. When using a single, infinitely distant point light source, the illumination I_p of a surface point $\mathbf{p}$ is given by

$$I_p = \max\left\{\left\langle \mathbf{n} \,\middle|\, \mathbf{L} \right\rangle, 0\right\},$$

where $\mathbf{n}$ is the surface normal at $\mathbf{p}$ and $\mathbf{L}$ is the direction of light. Both vectors are normalized, so the value of I_p is in the interval $[0,1]$. Now some values $I_{c,j} \in [0,1] = \text{const}$ (e.g., $I_{c,j} = \frac{j}{n},\ j = 0,\ldots,n$) are chosen and the isophotes/iso-curves $I = I_{c,j}$ are rendered.

The resulting image makes it easier to detect irregularities on the surface compared to standard shading. The user can visually trace the lines, rate their smoothness and transfer these observations to the surface: If the surface is C^k continuous then the isophotes are C^{k-1} continuous, since they depend on normals, i.e., on first derivatives (cf. Fig. 17).

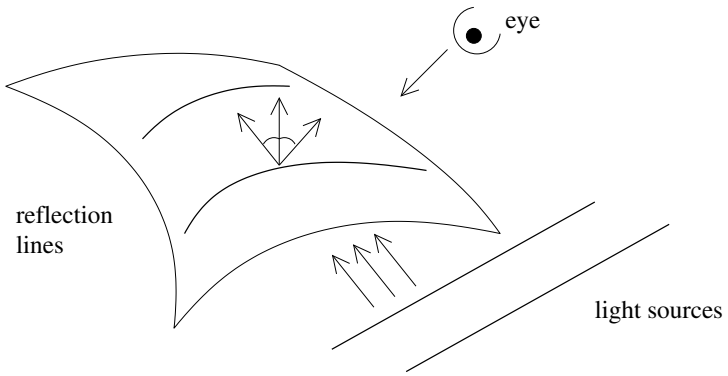


Figure 18: *Reflection lines. The light source consists of parallel lines that are reflected by the surface. The reflection property requires that angles of incidence (light,normal) are equal to angles of emission (viewing direction,normal).*

There are two main approaches to render iso-curves, such as isophotes: The first approach is to explicitly extract the curves or curve segments and then display them as lines. Here, in principle the same algorithms as for extracting iso-surfaces can be applied (Section 2.3.2), reduced to the setting of extracting a curve on a surface.

The second approach takes advantage of the graphics hardware and allows direct rendering of isophotes from illumination values at the vertices of a triangle mesh: A one-dimensional texture is initialized with a default color C. Illumination values I_p are now treated as texture coordinates, and for the isophote values $I_{c,j}$ the corresponding texels are set to a color $C_j \neq C$. With this setup the graphics subsystem will linearly interpolate the 1D texture within the triangles resulting in a rendered image of the isophotes (colors C_j) that are drawn onto the surface (color C) (cf. Fig. 17). The 1D texture approach benefits more from the graphics hardware in contrast to explicitly calculating line segments. A drawback is that the width of the curves varies due to texture interpolation.

Reflection lines In contrast to isophotes, rendering of reflection lines assumes a specular surface. As a consequence reflection lines change when the point of view is modified and when the object is rotated or translated. The light source consists of a set of "light-lines" that are placed in 3-space space. Normally, the light-lines are parallel lines (cf. Fig. 18).

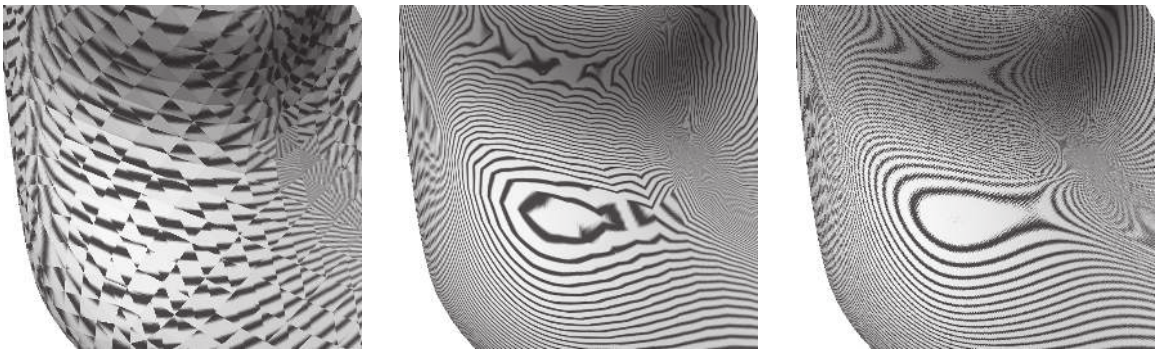


Figure 19: *Reflection lines on C^0, C^1 and C^2 surfaces. One clearly sees that the differentiability of the reflection lines is one order lower, i.e., C^{-1}, C^0 and C^1 respectively.*

Traditionally, reflection lines have been used in the process of designing cars. An arrangement of parallel fluorescent tubes is placed above the car model to survey the surface and its reflection properties.

Under the assumption that the light source is infinitely far away from the object, *environment mapping* can be used to display reflection lines in real-time. A texture for environment mapping is generated once by ray-tracing the light sources over a sphere. The graphics subsystem will then automatically generate appropriate texture coordinates for every vertex depending on its relative position and normal.

Reflection lines are an effective and intuitive tool for surface interrogation. If the surface is C^k continuous then the reflection lines are C^{k-1} continuous. Just like isophotes, they can be efficiently rendered by taking advantage of graphics hardware and they are also sensitive to small surface perturbations. In addition, the concept that a real-world process is simulated makes their application very intuitive even for unexperienced users. Fig. 19 shows reflection lines for C^0, C^1 and C^2 surfaces.

6.2. Visualizing curvature and fairness

If fairness is expressed in terms of curvature, the techniques described in Section 5 can be used for visualization. Gaussian curvature $K = \kappa_1\kappa_2$ indicates the local shape of the surface (elliptic for $K > 0$, hyperbolic for $K < 0$ and parabolic for $K = 0 \wedge H \neq 0$ resp. flat for $K = 0 \wedge H = 0$). A local change of the sign of K may denote a (even very small) perturbation of the surface. Additionally, mean curvature, principal curvatures, and total curvature $\kappa_1^2 + \kappa_2^2$ can be used. These scalar values are typically visualized using color-coding as shown in Fig. 20

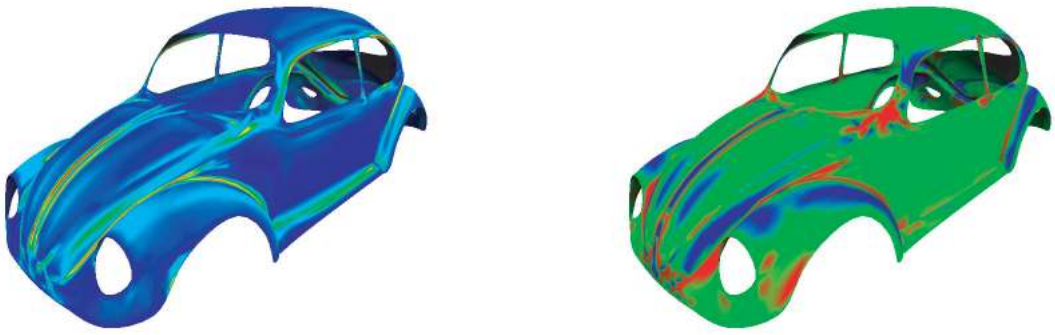


Figure 20: *Color coding curvature values, mean curvature (left) and Gaussian curvature (right).*

Figure 21: *Lines of curvature. Lines of curvature are superimposed on a flat shaded image of a VW Beetle model.*

Iso-curvature lines Iso-curvature lines are lines of constant curvature on a surface. They can be displayed similarly to isophotes, where instead of illumination values, curvature values are used. If the surface is C^k continuous, then the iso-curvature lines are C^{k-2} continuous, so iso-curvature lines are even more sensitive to discontinuities than isophotes or reflection lines.

A problem when rendering iso-curvature lines with 1D-textures may be a wide range of curvature values that may not map appropriately to the $[0,1]$ interval of texture coordinates or the actual texels. One solution is to clamp the curvature values to a suitable interval, the other solution is to explicitly extract the curves and draw them as lines.

Lines of curvature Besides the scalar principal curvatures, the principal directions also carry information on the local surface properties. They define discrete direction fields in the tangent space of the surface. By linearly interpolating principal directions computed at the mesh vertices over triangles using barycentric coordinates, a continuous field can be defined. Lines of curvature can then be traced on this direction field using Euler integration (see Section 10.2.1 for more details). Fig. 21 shows lines of curvature that provide very good and intuitive impression of the surface. Alternatively texture based techniques like *line integral convolution* (LIC) [CL93] can also be used on triangle meshes. However, tracing and constructing a large number of lines of curvature is rather expensive compared to the other techniques.

6.3. The shape of triangles

Some applications need "well-shaped", round triangles in order to prevent them from running into numerical problems, e.g., numerical simulations based on FEM. For this purpose, "round" triangles are needed, e.g., the ratio of the radius of the circumcircle to the shortest edge should be as small as possible [She02] (cf. Fig. 22).

The most common way to inspect the quality of triangles is to view a wireframe or hidden-line rendered image. This may not be an option for very complex meshes, however. A straightforward solution is a color coding criterion based on triangle shapes. This helps to identify even single "badly shaped" triangles (see also Section 10).

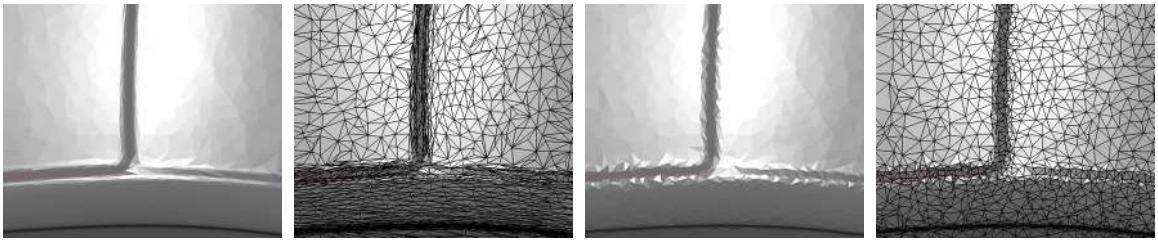


Figure 22: *Triangle mesh optimized for smooth appearance, leading to skinny triangles (left), and for triangle shape, leading to rendering artifacts (right).*

7. Mesh Smoothing

Mesh smoothing is a central tool in geometry processing with many applications such as denoising of acquired data, surface blending and hole-filling, or design of high-quality surfaces. In addition, smoothing techniques constitute foundations for geometric filtering or signal processing used in multi-resolution shape editing and mesh deformation methods as will be discussed in Section 11.

Many different techniques for mesh smoothing have been developed within the last decade. In this section, we will concentrate mainly on linear methods, namely Laplacian smoothing and (isotropic) mean curvature flow. Their main application is denoising and generation of fair surfaces as required in multi-resolution modeling.

7.1. General Goals

We distinguish two different goals of smoothing methods: The first is *denoising* measured data. For instance meshes acquired by range scanners typically show high frequency noise, i.e., small perturbations in the vertex positions, which do not correspond to shape features. Fig. 23 shows a typical example. Here, the goal is to smooth out these artifacts in such a way that the global shape, or the low frequency components, is preserved. In signal processing this is called low-pass filtering, well-known, e.g., in image processing. Denoising algorithms must be able to handle fairly huge data sets efficiently, as they may be applied directly after acquisition and before simplification (Section 9). This fact renders linear methods, i.e., those which only require numerical solution of a linear system, especially attractive. An additional requirement is often the preservation of certain surface features like sharp edges and corners, which should not be "blurred". However, this leads to non-linear methods.

A second goal is the design of high-quality, fair surfaces. This process is called *fairing*, and the resulting surfaces must satisfy certain aesthetic requirements. In order to find appropriate mathematical models these requirements are put essentially as *principle of the simplest shape* [Sap94], meaning that an aesthetic surface is free of unnecessary details such as noise or oscillations. Fig. 26 shows an example of fair surface design from boundary conditions. Mathematical formulations of this principle lead to the minimization of certain energy functionals, see Section 6, which are often inspired by physical processes such as spanning a membrane or bending a thin plate. The energy functionals are typically formulated in terms of intrinsic shape properties, i.e., quantities that do not depend on the particular parameterization (or triangulation in the discrete setting), such as curvatures (see Section 5). Hence the associated optimization problems are non-linear, and their numerical solution is more involved. Applications of fairing are for instance shape optimization or hole filling (see Section 4). For the latter, the hole is first filled with a template mesh, which is then subject to fairing while the transition at the hole boundary is required to be smooth.

Finally, smoothing is often applied in order to make triangulations more regular. This is a well-known technique to ensure numerical robustness of finite element methods (usually for planar domains in bivariate settings). For surfaces this means that the distribution of vertices over the mesh is optimized. This process is part of (isotropic) remeshing described in Section 10. In the following we review general approaches to mesh smoothing, their intuition and motivation.

7.2. Spectral Analysis and Filter Design

It is well-known from signal processing theory that Fourier transformation is a valuable tool for both, filter design and efficient implementation. For instance, every univariate signal function $f(t)$ is assumed to be a linear combination of periodic functions $e^{i\varphi t}$ (i.e., scaled and shifted sine waves) of different frequencies φ. Instead of observing the signal in the spatial domain one considers its spectrum in the frequency domain. Assuming that noise is associated with high frequencies, an ideal denoising filter would cut off such high frequencies prior to the inverse Fourier transform to the spatial domain. This is called a low-pass filter.

We will see that a similar notion of *geometric frequencies* can be established for surfaces and used for filter design. (We refer also to multi-scale techniques for surface deformation in Section 11.) However, contrary to image processing, analysis in the frequency domain will only serve as a theoretical tool and does not yield efficient implementations in general.

Let us for a moment consider the univariate case. The Fourier transform $F(\varphi)$ of a signal $f(t)$ is defined as

$$F(\varphi) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(t) e^{-i\varphi t} dt \ .$$

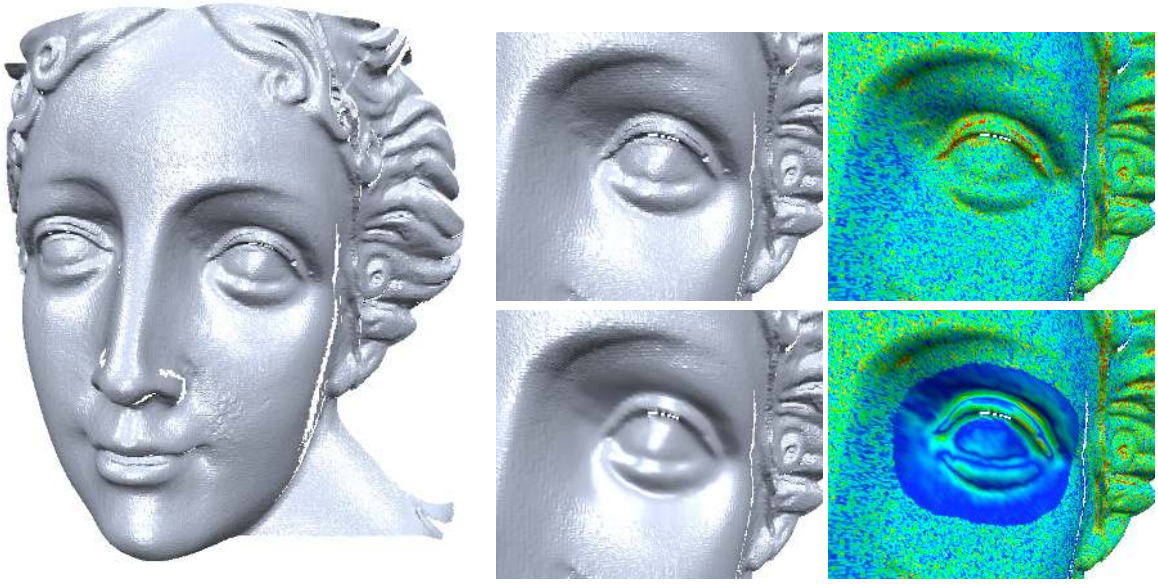


Figure 23: *This scan of a statue's face contains typical measurement noise, which can be removed by low-pass filtering the surface geometry. The bottom row shows selective smoothing, for better visualization only the eye region is considered. Mean curvature is superimposed as color-code in the right column.*

A low-pass filter would damp (or ideally cut off) high frequencies φ of F prior to the inverse transform, e.g., by multiplying F with a Gaussian. For (discrete) surfaces the situation is more difficult, we require some generalization of the basis functions of type $e^{i\varphi t}$. Considering the identity

$$\frac{\partial^2}{\partial t^2} e^{i\varphi t} = \Delta e^{i\varphi t} = -\varphi^2 e^{i\varphi t} ,$$

it follows immediately that $e^{i\varphi t}$ are eigenfunctions of the Laplace operator Δ with eigenvalues $-\varphi^2$. Therefore it seems natural to use eigenfunctions of the Laplace operator as basis also in the bivariate setting and for surfaces of arbitrary topology. As we know how to discretize the Laplacian on triangles meshes this will provide the generalization of Fourier transformation for filter design.

The discrete Laplacian operator (see also Section 5) on a piecewise linear surface, i.e., a triangle mesh, is expressed as

$$\Delta \mathbf{p}_i = \sum_{v_j \in \mathcal{N}_1(v_i)} \omega_{ij} (\mathbf{p}_j - \mathbf{p}_i) , \tag{18}$$

where for all vertices v_i weights are normalized such that

$$\sum_{v_j \in \mathcal{N}_1(v_i)} \omega_{ij} = 1 . \tag{19}$$

(Note that normalization and symmetry are not generally necessary for smoothing. In contrast, possibly required area terms destroy these properties, see also Section 5 and Section 12.) We can now write the discrete Laplacian operator as a matrix $\mathbf{L}$ with non-zero entries

$$\mathbf{L}_{ij} = \begin{cases} -1 , & i = j \\ w_{ij} , & v_j \in \mathcal{N}_1(v_i) \end{cases}$$

$\mathbf{L}$ is generally sparse, the number of non-zeros in each row is one plus the valence of the associated vertex. For the uniform discretization Δ_{uni} we choose weights $\omega_{ij} = \frac{1}{\#\mathcal{N}_1(v_i)}$, i.e., the Laplacian depends only on the mesh connectivity. Then $\mathbf{L}$ is symmetric and has real eigenvalues and eigenvectors.

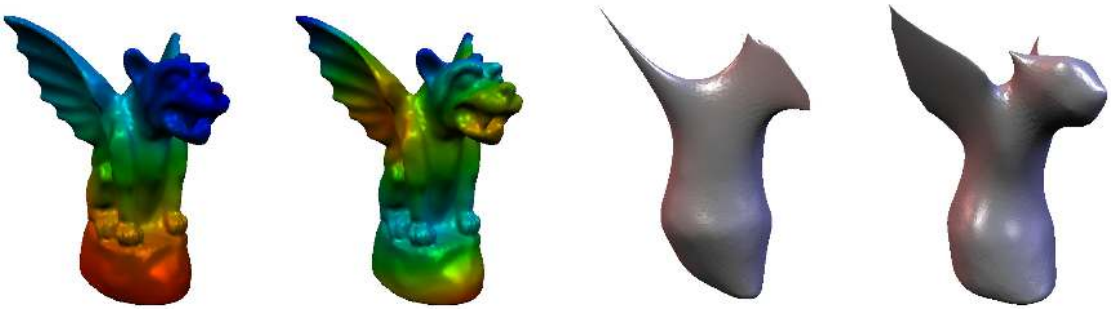


Figure 24: *Spectral analysis of a gargoyle model. The first* 20 *of* 10*k eigenvectors were computed. Left: The 2nd and 10th eigenvector of the associated discrete Laplace operator are visualized by the color codes. (Values are uniformly scaled). Right: Reconstruction of the model using only the first* 10 *and* 20 *eigenvectors, respectively. (Reconstructions are rescaled.)*

The eigenvectors of $\mathbf{L}$ form an orthogonal basis of $\mathbf{R}^n$, where n denotes the number of vertices, and the associated eigenvalues are commonly interpreted as *frequencies*. The projections of the coordinates $\mathbf{p}_x, \mathbf{p}_y, \mathbf{p}_z \in \mathbf{R}^n$ into this basis is called *spectrum* of the geometry. Given eigenvectors $\mathbf{e}_i$, the x-components $\mathbf{p}_x$ of the mesh geometry can now be expressed as

$$\mathbf{p}_x \;=\; \sum_{i=1}^{n} \alpha_i^x \mathbf{e}_i \;,$$

where the coefficients $\alpha_i^x = \mathbf{e}_i^T \mathbf{p}_x$, and similar for $\mathbf{p}_y, \mathbf{p}_z$. It shows that the eigenvectors associated with the first eigenvalues $0 \leq \lambda_1 \leq \cdots \leq \lambda_n$ correspond to low-frequency components: in other words, cancelling coefficients α_i^x associated with high-frequency components yields a smoothed version of the shape. Fig. 24 visualizes some eigenvectors on a model together with a synthesis using only very few low frequency components.

This is well-known from *spectral graph theory* [Chu97]: the projection into the linear space spanned by the eigenvectors provides a generalization of the Discrete Fourier Transform. This can also be seen immediately for the discrete univariate setting: here, the decomposition is equivalent to the discrete cosine transform (see, e.g., [Tau95b]).

For general surface meshes, their spectral decomposition defines a natural frequency domain. Taubin [Tau95b, Tau00] uses this fact to motivate *geometric signal processing* and to define low-pass filters for smoothing meshes (see also [TZG96]). In [KG00, SCT03] spectral analysis is applied for mesh compression, taking advantage of low-pass and high-pass filter properties, respectively. We refer to [Lév06] for a recent review of spectral mesh processing techniques.

Although the matrix $\mathbf{L}$ is generally sparse it is in practice *not feasible* to explicitly compute eigenvalues and eigenvectors even for moderately sized meshes: computational costs are too high and one has to pay close attention to numerical robustness. (In practice, the computation of *some* eigenvalues in a specified range is possible, as shown in Fig. 24.) Therefore in [KG00], meshes are partitioned without enforcing smoothness across patch boundaries, whereas in [SCT03, Tau95b], spectral analysis is applied as a theoretical tool.

Ideal low-pass filters are often too costly even in image processing. Instead of strictly truncating the frequency band, high frequencies are often damped, e.g., by weighting with an appropriate Gaussian kernel (often called Gaussian blurring). In a continuous setting, the Fourier transformation of a Gaussian kernel yields again a Gaussian. Therefore in the spatial domain this corresponds to convolution with a Gaussian or more general to some weighted averaging. The situation is similar for mesh filtering.

We illustrated the theoretical framework for ideal low-pass filtering on meshes. Unfortunately this is generally too expensive to be practical. Therefore, we will now focus on two major approaches to mesh smoothing: diffusion flow and energy minimization. Note that although different in motivation for particular instances, these two approaches are closely related, and they can be justified by the above observations.

7.3. Diffusion Flow

Diffusion processes constitute a powerful and well-understood tool for smoothing signals. They often arise as physical processes in the real world, which makes them intuitive to understand. A common example is heat distribution in an object, where the

local differences in temperature are equilibrated under conservation of energy. Let $x(u,t)$ denote the temperature at position u inside an object at time t, then the heat flow is given as $f = -\mu\nabla x$. Here, the diffusion constant $\mu > 0$ specifies the material conductivity. (Instead of a scalar in the isotropic case, we may set a positive definite symmetric matrix as diffusion tensor in general, see Section 7.5.1.) Furthermore, due to conservation of energy the continuity equation $\frac{\partial x}{\partial t} = -\operatorname{div} f$ applies (assuming no heat injection). Then the *heat equation* is expressed as the linear diffusion equation

$$\frac{\partial}{\partial t}x \;=\; \operatorname{div}\mu\nabla x\,. \tag{20}$$

In the following we will consider this type of diffusion equation for mesh smoothing: the vertex positions are subject to diffusion such that small differences, i.e., noise, are equilibrated. For the steady state we have zero flow $\frac{\partial x}{\partial t} = 0$ and hence $\Delta x = 0$. We remark that for appropriate settings the solution $x(u,t)$ to the diffusion equation is a convolution of the initial value $x(u,0)$ with a Gaussian kernel depending on the time step t.

In the following, we review discrete solutions of linear diffusion equations for smoothing triangle meshes. Particular approaches differ in the differential operator and its particular discretization, and different numerical integration schemes can be applied.

7.3.1. Laplacian Smoothing

Laplacian smoothing is a simple and very effective technique based on linear diffusion of vertex positions $\frac{\partial \mathbf{p}}{\partial t} = \mu\Delta\mathbf{p}$. Obviously, for triangle meshes this method depends on the discretization of the Laplace operator (see Section 5). The straightforward choice is a uniform discretization based on finite differences assuming a uniform triangulation.
Note that the uniform discretization smoothness geometry (shape) *and* triangulation, i.e., vertices move in normal direction as well as in their respective tangent planes.

7.3.2. Curvature Flow

Curvature is an intrinsic property of the surface that does not depend on parameterization (see Section 5). Such independence of the particular triangulation of a shape is favorable for smoothing: only the geometry of the shape is supposed to be smoothed while at the same time the shape of each individual triangle should be preserved as much as possible. This means that vertices should be displaced only in normal direction rather than in the associated tangent plane. Tangential drift occurs indeed for the uniform discretization of the Laplacian (see above), and in most applications it is regarded as an undesirable artifact.

Mean curvature flow [DMSB99] considers the flow equation

$$\frac{\partial \mathbf{p}}{\partial t} \;=\; -\mu H\mathbf{n}\,. \tag{21}$$

For smoothing, vertex positions $\mathbf{p}$ move along the surface normal $\mathbf{n}$ with speed proportional to the mean curvature $H = \frac{1}{2}(\kappa_1 + \kappa_2)$. As $H = \operatorname{div}\mathbf{n}$, speed is reduced if the normal field spreads out less in a local region, and in the extreme case vertices stay in place for zero curvature. Using the identity $\Delta_{\mathcal{S}}\mathbf{p} = -2H\mathbf{n}$, we replace the right hand side of (21) and apply the well-known discretization of the Laplace-Beltrami operator $\Delta_{\mathcal{S}}$ (see Section 5). This way, we can also interpret the mean curvature flow as diffusion using a more appropriate discretization of the Laplace operator on the surface (w.r.t. the initial mesh as parameter domain). The resulting linear diffusion equation reads as $\frac{\partial \mathbf{p}}{\partial t} = \mu\Delta_{\mathcal{S}}\mathbf{p}$. We remark that curvature flow has also been used in combination with parameterization regularization [OBB00].

7.3.3. Higher Order Flows

Higher order flows based on Δ^k (or $\Delta_{\mathcal{S}}^k$) are used due to better low-pass properties (see, e.g, [DMSB99]). In practice, bi-Laplacian smoothing ($k = 2$) is a good trade-off between efficiency and quality: In the frequency domain higher orders of the Laplace operator yield better truncation (damping) of high frequencies. However, the associated discrete linear operator is less sparse (see also Section 12). Note that higher order flows require (and are able to satisfy) higher order boundary conditions. This is similar to energy minimization methods discussed below.

7.3.4. Integration

A straightforward method for the numerical solution of the linear diffusion equations is *explicit* (or forward) Euler integration. This leads to an iterative algorithm using, e.g., the update rule

$$\mathbf{p}_i' \;=\; \mathbf{p}_i + \mu dt\,\Delta\mathbf{p}_i \tag{22}$$

on all vertex positions $\mathbf{p}_i$ for Laplacian smoothing. Updates can be applied simultaneously or sequentially [VMM99] in iterative algorithms of Jacobi or Gauss-Seidel type, respectively. In practice, direct solvers (see Section 12) in combination with implicit integration (see below) show superior efficiency and stability for most settings.

The above formula depends on the parameter μdt, which can be interpreted as time step and here should satisfy $0 < \mu dt < 1$ for stability reasons.

The explicit integration (22) of the (discrete) diffusion equation can be written in matrix form as

$$\mathbf{p}' = (\mathbf{I} + \mu dt\, \mathbf{L})\, \mathbf{p} ,$$

where $\mu dt < 1$ is required. Desbrun et al. [DMSB99] propose the use of a *backward Euler method* for *implicit* smoothing, which is unconditionally stable without limitations on the time step. Such implicit integration reads as

$$(\mathbf{I} - \mu dt\, \mathbf{L})\, \mathbf{p}' = \mathbf{p}$$

and requires the solution of a (sparse) linear system for the unknowns $\mathbf{p}'$ (see Section 12). The value of μdt can be chosen arbitrarily, and it roughly corresponds to the number of explicit integration steps.

7.4. Energy Minimization

Methods based on energy minimization frequently appear in mesh fairing and fair surface design (see, e.g., [Gre94, GLW96, KCVS98, MS92, WW92]). The idea is to penalize unaesthetic behavior of the shape. For this purpose different fairness functionals have been proposed. Ideally such functionals depend only on intrinsic surface properties, such as curvature, and not on a particular parameterization. For the discrete setting one can then expect the same geometric shape of the solution regardless of the initial triangulation.

Best known in this context is the *total curvature* of a surface $\mathcal{S}$

$$\int_{\mathcal{S}} \kappa_1^2 + \kappa_2^2 \, dA , \tag{23}$$

expressed as the area integral of the sum of squared principal curvatures (see, e.g., [MS92] and Section 6).

Parameter independence has a price, however: minimization problems are non-linear and the numerical computation of solutions (see, e.g., [WW94]) is generally too expensive to be practical for large meshes. For isometric parameterizations $\mathbf{x} : \Omega \to \mathbf{R}^3$, minimizing (23) is equivalent to minimizing

$$\iint_{\Omega} \|\mathbf{x}_{uu}\|^2 + 2\,\|\mathbf{x}_{uv}\|^2 + \|\mathbf{x}_{vv}\|^2 du\, dv . \tag{24}$$

This energy has a physical interpretation: it expresses the bending energy of a *thin plate* spanned across a domain Ω.

Generally such approaches linearize curvature terms by higher order derivatives for the sake of giving up parameter dependence. Still ad hoc minimization of (24) is rather involved. Fortunately, for some fairness functionals the minimizers are characterized by solutions of *linear* systems. In this case applying variational calculus [Kob97] yields the minimizer as solution of the associated Euler-Lagrange equation

$$\Delta^2 \mathbf{x} = 0 ,$$

subject to appropriate boundary conditions [KCVS98]. Note that this equation also characterizes the equilibrium of the linear diffusion $\frac{\partial \mathbf{x}}{\partial t} = -\mu \Delta^2 \mathbf{x}$ [DMSB99], and its discretization leads to a linear system. In Section 12 we discuss efficient solution of such systems.

Similarly, minimizing the membrane energy (25)

$$\iint_{\Omega} \|\mathbf{x}_u\|^2 + \|\mathbf{x}_v\|^2 du\, dv , \tag{25}$$

which captures the energy of a membrane spanned across a domain Ω leads to solving $\Delta \mathbf{x} = 0$.

For achieving higher order fairness the following well-known functional is minimized

$$\int_{\mathcal{S}} \left(\frac{\partial \kappa_1}{\partial \mathbf{e}_1} \right)^2 + \left(\frac{\partial \kappa_2}{\partial \mathbf{e}_2} \right)^2 dA \tag{26}$$

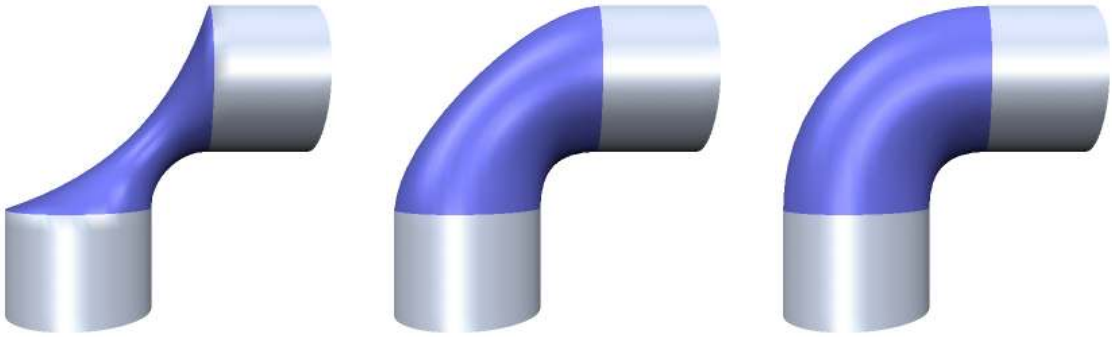


Figure 25: *The order k of the energy functional and of the corresponding Euler-Lagrange PDE $\Delta_S^k \mathbf{x} = 0$ defines the stiffness of the surface in the support region and the maximum smoothness C^{k-1} of the boundary conditions. From left to right: membrane surface ($k = 1$), thin-plate surface ($k = 2$), minimum variation surface ($k = 3$).*

to penalize variation of curvature, which yields *minimum variation surfaces* [MS92]. Giving up parameter independence corresponds to solving the sixth-order PDE $\Delta^3 \mathbf{x} = 0$.

The Euler-Lagrange equations associated with minimizers of various fairing functionals show their relation to steady state solutions of diffusion flow (and hence signal processing and low-pass filters). It follows that fairing indeed refers to designing fair surface which ideally depend only on the given boundary conditions: for surfaces derived from $\Delta^k \mathbf{x} = 0$, boundary constraints of order C^{k-1} are interpolated. Fig. 25 illustrates the application of different fairing functionals with appropriate boundary conditions for a simple cylindrical shape. This is in contrast to denoising which is usually far from the steady state. Note that for the solution of the arising linear systems appropriate boundary conditions have to be applied to guarantee the existence of solutions. (The Laplacian matrix does not have full rank.)

7.5. Extensions and Alternative Methods

We classified smoothing schemes into two categories depending on whether they are based on diffusion flow or energy minimization. Both categories lead to PDE discretization, and both are tightly connected as we focus on linear methods and the required simplifications. In the following we briefly review some (non-linear) extensions and alternative methods.

7.5.1. Anisotropic Diffusion

Denoising is supposed to smooth out small perturbations in a surface or outliers from measurements. The techniques discussed so far assume smooth surfaces and are not aware of surface features, i.e., sharp edges or creases and corners. However, most shapes are only piecewise smooth and denoising will also blur features as these are also represented by high-frequency components similar to what is assumed for noise.

This problem has been well-studied in image processing and a common approach to feature-preserving filtering is anisotropic diffusion [PM90] (see also [Wei98]). The basic idea is to consider the diffusion equation (20) and to replace the scalar diffusion constant μ by a data dependent *diffusion tensor* $\mathbf{D}$. This modification renders the equation non-linear and guides the directional (i.e., anisotropic) diffusion. A natural choice for $\mathbf{D}$ is the curvature tensor (in combination with an appropriate transfer function), which enables feature preservation or even enhancement: the speed of the flow is reduced in directions of high normal curvature, e.g., across sharp edges. There are various related approaches to feature preserving smoothing as for instance in [BX03, CDR00, DMSB00, HP04].

7.5.2. Normal Filtering

The basic idea of normal filtering methods is as follows: instead of filtering the spatial coordinates, the normal field of the surface is smoothed. The resulting normals are then integrated in order to reconstruct a smooth surface. Hence, in contrast to smoothing surfaces, or vertex positions, directly, their derivatives are subject to smoothing. This is usually achieved by a diffusion process [BO01, OBS02, TWBO02, Tau01, YZX*04]. We remark that normal smoothing is commonly applied as a preprocess for stabilization (mollification) in order to get reliable estimates for other methods (see, e.g., [JDD03]).

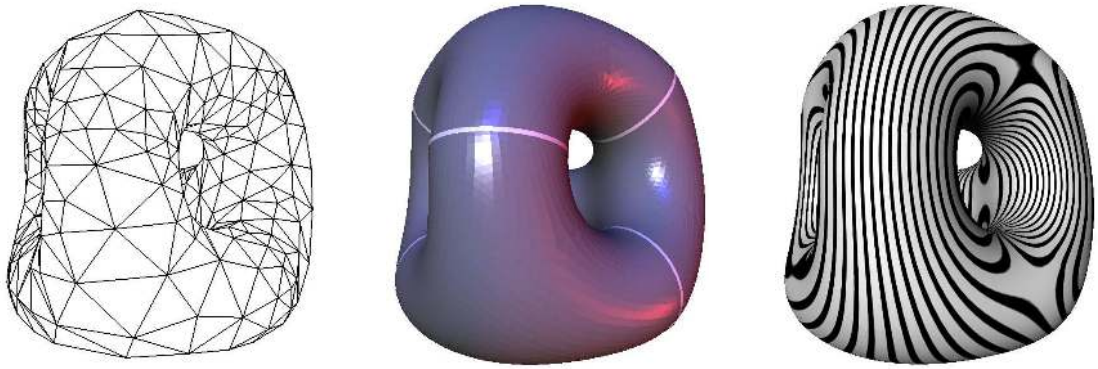


Figure 26: *Six circles with C^1 boundary-conditions are used to design a "tetra thing". Due to the symmetry the final solution is actually G^2 continuous in this case, which is indicated by the smooth reflection lines (see Section 6). Surfaces are constructed using the intrinsic fairing method [SK01] based on solving $\Delta_{\mathcal{S}} H = 0$, therefore the solution is independent of the triangulation (or parameterization, respectively).*

7.5.3. Statistical Methods

Smoothing can also seen from a statistical point of view: signal and noise are assumed to be stochastic processes with known spectral characteristics or known autocorrelation and cross-correlation. The Wiener filter is a well-known example from image processing. Local adaptive Wiener filtering has been adapted to denoising discrete surfaces [Ale02, PG01, PSZ01]. Also the following bilateral filtering relies on robust statistical estimations.

7.5.4. Bilateral Filtering

Bilateral filtering of images [TM98] (see also [Bar02] for relation to nonlinear diffusion) is a powerful feature-preserving filtering technique. The central idea is to consider both, the image domain (as for classical filtering) and its range: each pixel becomes a weighted average of *similar* pixels in the neighborhood, where "similar" is defined in terms of spatial distance *and* intensity.

In [FDC03, JDD03] bilateral filtering is adapted to denoising surface meshes, where spatial distance and local variation of normals is taken into account. In [FDC03] the normal displacement of vertex positions for smoothing is computed based on weighted averages of these measured. The non-iterative approach in [JDD03] does not require explicit connectivity information and applies (mollified) normals to predict vertex positions, which are used for weighting. The rationale behind this is that prediction fails near shape features, i.e., distances to such predicted points are larger.

7.5.5. Approaches based on non-linear PDEs

Such methods should depend exclusively on intrinsic properties, i.e., be independent of the parameterization. In [SK01] a PDE-based method was developed for design of fair surfaces. The method enables G^1 boundary constraints (prescribed as vertices and unit normals), such that the resulting shape is independent of the particular triangulation. This particular approach is based on solving the fourth-order non-linear PDE

$$\Delta_{\mathcal{S}} H = 0, \tag{27}$$

i.e., it depends purely on intrinsic properties. This can be interpreted as one possible nonlinear analogon to thin plate splines minimizing (24), and the equation characterizes the equilibrium of the Laplacian of curvature flow [CS99]. Due to the mean value property of the Laplacian the extremal mean curvatures are obtained at the boundaries. As a consequence there are no local extrema in the interior [SK00], and thus the principle of simplest shape requirement is satisfied. Notice that the numeric solution of the PDE requires high-quality discretization of the mean curvature (following [MS92], see Section 5). For efficiency reasons the fourth order PDE is factored into two second order problems.

Recently Bobenko and Schröder [BS05b] used discrete Willmore flow for denoising and fair surface design. The minimizer of the associated energy functional also minimizes (23) for certain settings.

7.6. Summary

We gave a brief overview of mesh smoothing techniques with focus on linear methods based on diffusion flow and energy minimization, revealing relations between the two approaches and relations to spectral analysis. These techniques are linear and hence very efficient and well-understood, see also Section 12 for efficient numerical solvers and overview of computational costs. They constitute basic tools for further geometry processing steps, e.g., for shape deformation Section 11. We listed several alternative techniques and summarized their main ideas. In conclusion we remark that there are several other aspects in smoothing that were not discussed here, such as volume preservation or existence of solutions (which is still unknown for minimization of many standard non-linear functionals).

8. Parameterization

Parameterization techniques establish mappings between surfaces. Such mappings are required by many applications in computer graphics and geometry processing as for instance texturing, compression, scattered data approximation, and remeshing (see Section 10). A central property of these mappings is distortion, i.e., the change of angles, areas and distances in the image. This property is commonly used to categorize parameterization methods.

In this section we review parameterization methods with focus on mappings to planar domains. The importance of parameterization techniques is reflected by the significant amount of work on the topic in recent years; we refer to the recent comprehensive survey by Floater and Hormann [FH05a].

8.1. Objectives

The central objective of parameterization techniques is to establish bijective mappings between surfaces and *parametric domains*. For example in the previous section, we referred to *parameterization* as the particular triangulation of a surface: this can be seen as a piecewise linear mapping from the triangulation as domain onto the approximated shape. In most practical settings simple standard domains like plane or sphere are chosen. Mappings are established between a surface (patch) and a homeomorphic domain, e.g., a genus-zero surface can be mapped to a sphere (see, e.g., Fig. 29). In practice, surfaces may be cut to introduce additional boundary loops. For instance, a sphere can be unfolded to a disk after opening it with one cut. Often surfaces may be partitioned into a number of disk-like patches, which are mapped separately. A well-known example is a texture atlas. In the following we restrict ourselves to mapping surface patches homeomorphic to a disk onto the plane.

The main concern of parameterization methods is the reduction of *parametric distortion*. Basic differential geometry reveals that isometric or length-preserving mappings are rare: they exist only for those surfaces which share intrinsic properties with the plane such as planes, cylinders, or ruled surfaces. Assume a regular surface $\mathcal{S} \subset \mathbb{R}^3$ with parametric representation

$$\mathbf{x}(u,v) = \begin{pmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{pmatrix}, \quad (u,v) \in \mathbb{R}^2 .$$

The mapping $\mathbf{x}$ is characterized by the first fundamental form $\mathbf{I} \in \mathbb{R}^{2\times 2}$ (see Section 5) which tells how distances — and hence angles and area — measured in the parametric domain are translated to distances on the surface. The mapping is *isometric* iff $\mathbf{I}$ is the identity, i.e., for the arc element ds we have $ds^2 = (du,dv)\,\mathbf{I}\,(du,dv)^T = du^2 + dv^2$. *Conformal mappings* preserve angles. Here, isotropic scaling of $\mathbf{I}$ by some scalar $\mu(u,v) \neq 0$ is allowed: $ds^2 = \mu(u,v)\,(du^2 + dv^2)$. *Equiareal mappings* preserve area, hence $\det \mathbf{I} = 1$ is required: for the area element we have $dA = du\,dv$. Note that consequently any mapping which preserves both, angles and area, also preserves distances and vice versa. Alternatively, these properties can be expressed in terms of the eigenvalues λ_1, λ_2 of $\mathbf{I}$ (or, as $\mathbf{I} = J^T J$, equivalently in terms of singular values of the Jacobian J for planar mappings). In summary, we have $\lambda_1 = \lambda_2$ for isometric mappings, $\lambda_1/\lambda_2 = 1$ for conformal mappings, and $\lambda_1\lambda_2 = 1$ for equiareal mappings (see also Section 5 and [FH05a]).

Many approaches use the above characterizations directly and minimize appropriate functionals in order to establish (approximations to) low-distortion maps. In many practical applications the goal is a balance between angle and area preservation.

Besides distortion there are several other important aspects. Most important are guarantees on the validity of the resulting parameterization. In fact, many methods are guaranteed to produce bijective mappings only under particular, rather restrictive boundary conditions — although they may perform well in many practical settings. A related aspect is the treatment of boundaries: are they fixed in the domain or allowed to evolve freely? Boundaries are often fixed to convex polygons in order to guarantee valid mappings. On the other hand, adding some degrees of freedom to boundaries may reduce distortion. Some applications require additional constraints such as fixing interior points in the domain. An example is morphing where semantic correspondence between mappings from a common domain to different surfaces is required.

In summary, there are many criteria for classifying parameterization methods: type of domain (e.g., plane, sphere), minimized distortion (angle and/or area), treatment of boundaries (fixed or free evolution), numerical solution (e.g., linear or non-linear), guarantees on validity and convergence. Consequently, many different methods can be found in the literature. In the following we summarize some of the main ideas.

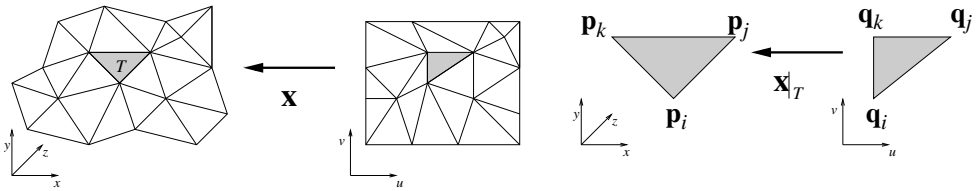


Figure 27: *Example of a piecewise linear mapping. The meshes share the same connectivity, and individual triangles are mapped.*

8.2. Discrete Mappings

We assume that surfaces are approximated by triangle meshes. Consequently, we are interested in discrete parameterizations, i.e., piecewise linear mappings. Given a surface mesh $\mathcal{M}$ with vertices $\mathbf{p}_i \in \mathbb{R}^3$ we represent such mapping as a second triangle mesh $\mathcal{M}'$ which shares the connectivity of $\mathcal{M}$. The vertices $\mathbf{q}_i$ of $\mathcal{M}'$ are located in the domain, e.g., $\mathbf{q}_i \in \Omega \subset \mathbb{R}^2$ for planar mappings. We assume interpolation $\mathbf{x}(\mathbf{q}_{i,1}, \mathbf{q}_{i,2}) = \mathbf{p}_i$ and obtain a bivariate linear mapping $\mathbf{f}|_T : \mathbb{R}^2 \to \mathbb{R}^2$ for every triangle T. By introducing local coordinate systems for surface triangles T, distortion can be measured easily for these atomic linear maps $\mathbf{f}|_T$ in terms of their Jacobians. Finally, $\mathcal{M}'$ represents a valid, bijective mapping if no degenerate triangles exist and pairwise intersections of triangles are empty. The latter condition is violated locally in case of fold-overs of neighboring triangles or globally in case of boundary self-intersections (see also Section 4). Fig. 27 illustrates this setting. Note that an implementation would require only one triangle mesh data structure to represent $\mathcal{M}$ and $\mathcal{M}'$. This data structure defines two types of coordinate attributes $\mathbf{p}$ and $\mathbf{q}$ per vertex. A typical example is to store position and texture coordinates per vertex.

An intuitive way to establish a planar map is to fix the boundary on the plane, e.g., by projection, and to apply smoothing: minimizing the membrane energy (25) yields a planar mesh and hence a mapping. This can be interpreted as flattening the surface into the plane and is consistent with the physical interpretation of the membrane energy. There is one pitfall, however: the result is not necessarily a valid mapping. In order to have guarantees on validity, i.e., no degeneracies and no foldovers, one has to carefully choose boundary conditions (and the discrete differential operator). In a continuous setting the above motivation can be seen as solving $\Delta_{\mathcal{S}}\mathbf{x} = 0$ subject to boundary conditions $\mathbf{x}|_{\partial\mathcal{S}} = \mathbf{x}_0(u,v)$ (see Section 7).

8.3. Angle Preservation

Conformal mappings preserve angles, i.e., angles of intersecting curves in the domain are the same as corresponding angles between surface curves on $\mathcal{S}$. In contrast to the continuous setting conformal maps can only be approximated in the discrete sense, where strict angle preservation is generally not possible.

Suppose $\mathbf{f} : \mathcal{S} \to \Omega \subset \mathbb{R}^2$, is a mapping from the surface $\mathcal{S}$ to the plane such that $\mathbf{f}(s,t) = (u,v)$. Here, we assume that coordinates (s,t) are given on the surface, and we are looking for planar coordinates (u,v). The above 3D setting, which flattens the surface by smoothing, corresponds to minimizing the Dirichlet energy

$$E_D(f) \;=\; \frac{1}{2}\int_{\mathcal{S}} \|\nabla_{\mathcal{S}} f\|^2 \,,$$

which is achieved for solutions to the above Laplace equations

$$\Delta_{\mathcal{S}}\, u \;=\; 0 \quad \text{and} \quad \Delta_{\mathcal{S}}\, v \;=\; 0 \,, \tag{28}$$

where $u|_{\partial\Omega} = u_0$ and $v|_{\partial\Omega} = v_0$. Here $\Delta_{\mathcal{S}}$ is the Laplace-Beltrami operator w.r.t. the surface $\mathcal{S}$ (see Section 5).

Generally, the parameterization problem can be stated as energy minimization, for instance considering functions (or various norms) of the first fundamental form, Dirichlet energy or conformal energy. The minimizers are often characterized by the solutions of certain partial differential equations, and finite element/finite difference methods can be applied to compute discrete solutions.

Minimizers of the Dirichlet energy or equivalently solutions to the Laplace equations are called *harmonic mappings*. They play an important role in parameterization since *conformal* mappings constitute a special case within the class of harmonic mappings: every conformal mapping is also harmonic. In the discrete planar setting, the construction of harmonic mappings leads to the solution of linear problems [EDD*95], and validity is guaranteed after fixing convex boundaries [Flo03b]. We refer to [FH05a] for a more detailed discussion (see also [PP93]).

Harmonic mappings can be computed efficiently, and, due to their simplicity, they are commonly used in practice. However, they are not angle preserving in general. Still, in the discrete setting, we can interpret them as fair approximations to conformal mappings: the solution to (28) yields a unique harmonic mapping, and any conformal mapping must satisfy the same conditions.

In the following we review approaches to the computation of discrete harmonic mappings. *Convex combination maps* are established by solving Laplace equations (28) using a certain class of discretizations of the Laplace operator (18) with positive weights ω_{ij}. Together with normalization (19) this means that every interior vertex (u_i, v_i) will be a convex combination of its neighbors and hence be located in their convex hull. For such settings, validity of discrete mappings is guaranteed if the boundary is mapped to a convex polygon in the domain. The special case of uniform weights leads to *barycentric mappings* which are well-studied in graph-theory [Tut63]. Floater [Flo97] designed *shape preserving weights* such that planar meshes are reproduced (or alternatively the map coincides with an affine mapping of the boundary). Recently, Floater proposed mean value coordinates [Flo03a] as a superior alternative. The design of the associated weights is based on the mean value property of harmonic functions: the function value at $\mathbf{x}_0$ is equal to the average of values on any circle centered around $\mathbf{x}_0$. In the discrete setting the goal is to find piecewise linear mappings that satisfy this property for each interior vertex. The resulting weights are positive, however, not symmetric in general.

Convex combination maps are advantageous for establishing harmonic mappings due to the positivity of the weights. This is important for guarantees on validity. Note that other weights (see below) work well in practice but require more restrictive conditions to guarantee one-to-one mappings [Flo03b].

In order to motivate angle preservation, discrete conformal maps can be computed as finite element approximations [HAT*00] or defined as minimizers of a conformal energy [PP93]. Lévy et al. [LPRM02a] express the problem in terms of the Cauchy-Riemann equations. Conformal mappings $\mathbf{f}(s,t) = (u(s,t), v(s,t))$ in the plane satisfy the Cauchy-Riemann equations

$$\frac{\partial u(s,t)}{\partial s} = \frac{\partial v(s,t)}{\partial t}, \quad \frac{\partial u(s,t)}{\partial t} = -\frac{\partial v(s,t)}{\partial s}. \tag{29}$$

Roughly speaking, this means that coordinate functions are defined by the gradients (u_s, u_t) and (v_s, v_t), which are equal in length and orthogonal. Here, u_s, u_t, v_s, v_t denote the respective partial derivatives $\frac{u(s,t)}{\partial s}$, etc., and (28) is obtained from differentiation of (29). While this applies only to planar, continuous settings, the idea in [LPRM02a] is to minimize a functional that penalizes violation of these conditions for a discrete surface: for every triangle this is the conformal energy $(u_s - v_t)^2 + (u_t + v_s)^2$. Minimization of the quadratic energy corresponds to satisfying the above conditions in least-squares sense and leads to the solution of a linear system. Although the linear systems are different in setup and conditioning, this approach is tightly coupled to the computation of discrete conformal parameterizations as proposed by Desbrun et al. [DMA02], where a Dirichlet energy is minimized (see also [CD02]). For interior vertices the latter corresponds to using cotangent weights for the discrete Laplace operator. (Note that these weights can be negative, see Section 5, which may lead to invalid solutions. The same applies to [LPRM02a].) We refer to [RL03] and Section 12 for details on efficient numerical solvers. Both methods apply so-called natural boundary conditions, which are conditions on the derivatives of the boundary. This enables the evolution of the boundary in the plane, where the solution depends on particular fixed vertices. Fig. 28 (left) and Fig. 29 (center) show examples of discrete conformal maps. The methods reviewed so far are based on the solution of linear systems and hence very efficient (see Section 12). However, they lack guarantees for general or evolving boundaries.

A popular non-linear method is the Most Isometric Parameterization (MIPS) of Hormann and Greiner [HG00]. It measures conformality for every triangle in terms of the condition number of the Jacobian of the linear atomic map $\mathbf{f}|_T$ w.r.t. the Frobenius norm, which can be written in terms of the eigenvalues of the first fundamental form (or singular values of the Jacobian). The energy functional associated with every linear piece is minimal if $\mathbf{f}|_T$ is conformal. We refer to [FH05a] for the relation to the conformal energy used in [PP93]. The MIPS algorithm enables evolution of the boundary and computes one-to-one mappings. Due to the non-linearity of the method, computation is involved and relatively expensive. A solution of a (fixed-boundary) linear method can be used as initial guess, e.g., for Newton methods. Hierarchical approaches enable efficient processing of larger meshes (see also [HGC99]).

Angle based flattening (ABF) [SdS00] is an alternative approach to establishing angle preserving mappings. ABF specifies the parameterization problem in terms of interior angles. This seems natural for computing conformal maps. ABF minimizes a functional which penalizes angular distortion of the planar mesh w.r.t. the angles of the original mesh. A set of linear and non-linear constraints on the planar angles guarantees the validity of the parameterization. Obvious conditions are positivity of angles, vertex consistency and triangle consistency, i.e., angles sum to 2π and π around interior vertices and within each triangle, respectively. In addition one more class of non-linear conditions is required to guarantee a consistency (see also [BV93]). In contrast to most other approaches, ABF enables free evolution of the boundary while no local foldovers can occur. Global self-intersections, where parts of the boundary overlap in the plane, are still possible and must be resolved in postprocessing steps; this is similar to other methods. In [ZRS04b] additional inequality constraints are proposed to control the local convexity of the

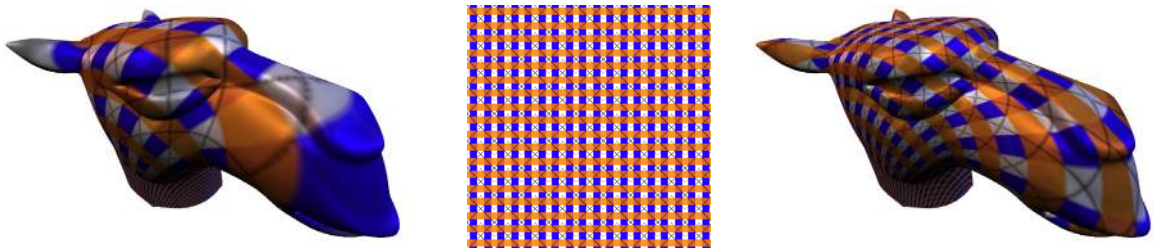


Figure 28: *Left: Discrete conformal parameterization [DMA02, LPRM02a] of a camel head model to the plane. Center: The texture is mapped from the plane onto the surface and visualizes distortion of angles and area. Right: Area distortion is improved using [ZRS05a].*

boundary in addition to a transformation of the minimization problem to improve ordering and sparsity of the system matrix. (A side effect is the easier setup of the system matrix.) The required non-linear optimization is relatively expensive compared to linear approaches. In order to make it practical for moderately sized meshes, in [SLMB04] hierarchical methods are applied, while in [ZRS04a] efficient iterative solvers are examined.

The recent approach in [KSS06] establishes discrete conformal maps based on circle patterns. Here, angles are also the subject of optimization, however, the associated minimization problem which drives the computation can be solved by quadratic programming. The method offers flexibility on the evolution of the boundary including explicit specification of singularities (see below).

8.4. Reducing Area Distortion

For many applications conformal mappings are not sufficient as they may suffer from severe area distortion. In fact, for fixed boundary settings it is typical that the size of triangles in the domain is scaled drastically far from the boundary. One possible solution is to introduce new cuts to provide more flexibility at the price of having discontinuities. In most situations it is preferable to ease angle preservation in favor of area preservation. (Equiareal mappings are not practical in general, see [FH05a]). The idea is to have a fair balance between angle and area preservation. Many methods start from conformal mappings as initial solutions for subsequent optimization.

Degener et al. [DMK03] modify the MIPS energy in [HG00] to account for area scaling. Sander et al. [SSGH01] define a functional that measures the "stretch" of a mapping based on singular values of the Jacobian of $\mathbf{f}|_T^{-1}$. The use of hierarchical solvers for the arising non-linear minimization is avoided in [YBS04] who apply an efficient quasi-Newton optimization. In [SCGL02] a modification of this functional is used and the global optimization is replaced by a greedy approach which may cut the surface while it proceeds. Variational methods are applied in [CLR04], where distortion measures are derived from principles of rational mechanics.

The above methods define appropriate energy functionals which to some extent penalize area distortion. The methods differ in the degree of conformality that is retained and in the smoothness of the solution which may not be included in the objective function, leading to visual cracks due to a rapid change in triangle shape, see [YBS04]. The associated minimization problems are non-linear, and their solution generally requires sophisticated solvers.

The following methods are based the solution of on linear problems. In order to establish area-preserving, so-called authalic, mappings [DMA02] derive an energy and associated weights ω_{ij} for local preservation of area. The global effects are, however, limited in general: global area scaling cannot be reliably predicted a priori based on local information only. Zayer et al. [ZRS05a] propose the use of discrete quasi-harmonic maps: instead of solving the Laplace system to minimize the Dirichlet energy the following quasi-harmonic equation is considered

$$\mathrm{div}_{\mathcal{S}}(\mathbf{C}\,\nabla_{\mathcal{S}}\,\mathbf{f}) \;=\; 0\,, \tag{30}$$

where the symmetric and positive definite tensor $\mathbf{C}$ is derived from the Jacobian of an initial discrete conformal mapping (e.g., from [DMA02, LPRM02a]). In the discrete setting the 2×2 matrix function $\mathbf{C}$ is piecewise constant and computed per triangle. Discretizations of the differential operator are provided based on cotangent weights (16) (better conformality) and mean value coordinates (positivity). We remark that Zayer et al. [ZRS05a] interpret both cotangent weights and mean value coordinates as different discretizations of the same differential operator. The approach leads to an iterative process which requires the solution

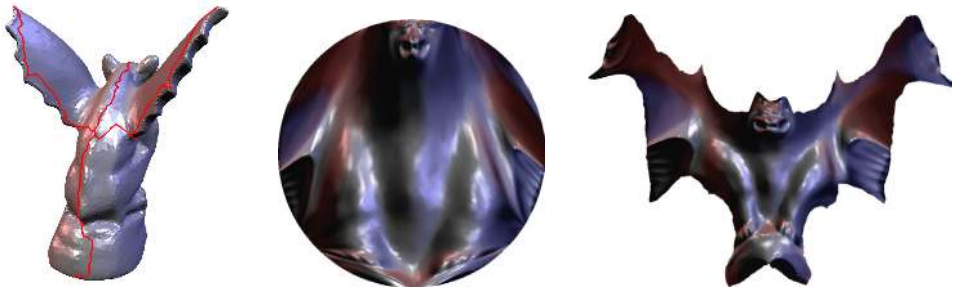


Figure 29: *The gargoyle model (see also Fig. 24) is opened with a cut shown as red lines (left). The center image shows a discrete conformal map visualized by the original 3D shading. Here, the boundary is fixed to a circle. The right image shows a free boundary map using [ZRS05b].*

of one linear system in each step. In practice the method can also start and recover from ill-shaped configurations including folds. For example one could fix the boundary on the plane and then just project interior vertices. As an empirical observation there is no change in the solution after a few iterations. Fig. 28 illustrates the effect of area scaling. In [ZRS05b] quasi-harmonic maps are applied for free boundary parameterization based on a Poisson setting (see also Section 11). In contrast to the natural boundary conditions applied in [DMA02, LPRM02b], no boundary vertices are treated special by fixing them in the plane, and the solution does not depend on such a particular choice. This method also requires only a few steps of solving linear systems. Fig. 29 shows an example.

8.5. Spherical Mappings

So far we discussed only mappings of disk-type surface patches to the plane. However, for many surfaces this requires artificial cuts which may lead to discontinuities across patches. Indeed, for surfaces of genus zero it seems much more natural to chose the sphere as the domain. The fact that many geometric models are homeomorphic to a sphere makes such spherical mappings an appealing geometry processing tool. Spherical maps have a long history. For instance, many early applications considered the spherical mapping of brain surfaces. Fig. 30 (center) shows an example of a spherical mapping.

The spherical parameterization problem can be stated as a minimization problem subject to the non-linear constraint

$$||\mathbf{p}_i||^2 = 1, \quad i = 1, \ldots, n \, .$$

The choice of objective function is similar as to the planar case. However, the additional constraint makes the problem more involved than its planar counterpart.

A straightforward approach is a Gauss-Seidel iteration of a local relaxation step based on tangential Laplacian smoothing followed by back projection onto the sphere. This type of solution was carried out in [Ale00, KVLS99a]. The result is used as starting point for computing a minimal Moebius transform in [GY02, GWC*04]. A principal problem is that a minimum is reached for degenerate configurations, e.g., with vertices slipping over the sphere until triangles collapse. Heuristics such as imposing stopping criteria or introducing additional boundary conditions cannot solve this problem in general. More promising is a careful analysis of the discrete objective function in order to compensate for the spherical setting [FSD05].

In [SGD03] the constrained minimization problem is expressed by adapting the planar angle based flattening method to the spherical setting. Gotsman et al. [GGS03] present a method that comes with theoretical guarantees and generates provably bijective maps. However, while theoretically interesting, both approaches are computationally too expensive to be of general use in practice. In [PH03], the "stretch" minimization is adapted to the spherical domain, where the algorithmic solution relies on hierarchical structures.

Alternative approaches resort to the existence of simple maps from the plane to the sphere. In [HAT*00] a triangle is cut from the mesh, and then the whole mesh is mapped into a triangular boundary. The resulting planar parameterization is lifted to the sphere by an inverse stereographic projection. Besides high distortion this method generally suffers from foldovers. This is due to the fact that the boundary in the planar domain is considered to extent to infinity. In order to overcome such limitations [SYGS05] cut the mesh into two halves and map each half to a circle. These two planar embeddings are mapped

Figure 30: *Mapping a cow model (left) to the sphere (center, using [ZRS06]) and to a base triangulation (right, following [LSS*98]). The maps are visualized by the original shading and original vertices in coarse base triangles, respectively.*

onto the sphere and serve as starting point for subsequent non-linear optimization on the sphere. This approach is motivated by [GGS03] and greatly reduces computation time.

In [BGK92] a different approach is taken: first an initial parameterization is established based on curvilinear coordinates. In order to achieve this, two poles have to be identified and the surface is cut along the date line from pole to pole. Second, the solution is further improved in the spherical domain by non-linear optimization which turns out to be too unstable for practical use. Hierarchical methods are applied in [QBH*00] to improve convergence, however, it is still impractical from a performance point of view. Zayer et al. [ZRS06] propose a related but efficient approach taking advantage of techniques developed for planar parameterizations: area distortion of an initial solution in curvilinear coordinates is improved by discrete quasi-harmonic maps [ZRS05a]. All this is achieved in the planar domain using appropriate linear solvers (see Section 12). Final smoothing on the spherical domain is restricted to neighborhoods of the cut date line.

8.6. Mapping Surfaces of Arbitrary Topology

Arbitrary topology surfaces are often cut to disk-like patches which are then mapped to the plane. For triangle meshes natural partitions can be computed by decimation (see Section 9): the simplified mesh, called the base mesh, is used as the domain, and vertices of the original surface mesh are mapped to the coarse base triangles. The result is a piecewise parameterization over the base mesh. Fig. 30 (right) shows an example. There is a variety of such approaches which generally follow the original algorithmic frameworks in [EDD*95, LSS*98]: discrete geodesics are used for partitioning and defining a base domain in [FHR02]. In [KLS03], a global optimization is applied to achieve smooth transitions between the patches. The choice of base mesh and objective can be tuned for spherical parameterization as done in [PH03].

Some applications require consistent mappings between surfaces using a common base mesh which is either given a priory [PSS01] or generated as the method proceeds [KS04, SAPH04].

Gu and Yau [GY03] compute discrete conformal structures for general surfaces. Their parameterization is based on a particular pair of holomorphic discrete 1-forms (see also [GGT05]). Ray et al. [RLL*05] apply non-linear optimization on periodic parameters to obtain a globally smooth parameterization where isoparametric lines are aligned to prescribed orthogonal vector fields. For all methods the coordinate functions generally must have critical points. The approach in [KSS06] allows one to explicitly place singularities at appropriate spots when designing a mapping, while [TACSD06] enable control of line singularities.

8.7. Alternative Objectives and Approaches

In the following we list alternative methods and possible extensions, which might be required by particular applications.

As already pointed out, some applications require consistent mappings of several surfaces to the same domain. In order to enforce semantic consistency, user provided constraints are required. Examples of such constraint parameterization approaches in planar domains are [Lév01, SG03a, KGG05].

Polycube maps [THCM04] provide a special kind of mapping for efficient texturing. Here, the base domain of a surface is approximated by a simple configuration of cubes.

Multidimensional scaling is a statistical method which was also applied to planar surface parameterization in [ZKK02]. The idea is to preserve distances, i.e., isometry, to a certain extent. However, the method does not scale well with data size and hence suffers from computational costs.

Guskov [Gus02] modifies Floater's shape preserving weights to achieve anisotropy for optimized grid generation. In [ZMT05] surface features are considered for alignment, while [TSS*04] include texture information in the optimization, which is based on stretch minimization for both approaches.

8.8. Summary

Parameterization is required in many applications in computer graphics and geometry processing, see for instance Section 10. The topic relies on discrete differential geometry (Section 5) to characterize mappings. Besides parametric distortion there are several other criteria for the categorization of the methods, and consequently many different approaches exist. We gave a brief overview of methods for establishing planar mappings and listed approaches to spherical maps and mapping surfaces of arbitrary topology.

9. Mesh Decimation

Mesh decimation describes a class of algorithms that transform a given polygonal mesh into another mesh with fewer faces, edges and vertices [GGK02]. The decimation procedure is usually controlled by user defined quality criteria which prefer meshes that preserve specific properties of the original data as well as possible. Typical criteria include geometric distance (e.g. Hausdorff-distance) or visual appearance (e.g. color difference, feature preservation, ...) [CMS98].

There are many applications for decimation algorithms. First, they obviously can be used to *adjust the complexity* of a geometric data set. This makes geometry processing a scalable task where differently complex models can be used on computers with varying computing performance. Second, since many decimation schemes work iteratively, i.e. they decimate a mesh by removing one vertex at a time, they usually can be inverted. Running a decimation scheme backwards means to reconstruct the original data from a decimated version by inserting more and more detail information. This inverse decimation can be used for *progressive transmission* of geometry data [Hop96]. Obviously, in order to make progressive transmission effective we have to use decimation operators whose inverse can be encoded compactly (cf. Fig. 33).

There are several different conceptual approaches to mesh decimation. In principle we can think of the complexity reduction as a one step operation or as an iterative procedure. The vertex positions of the decimated mesh can be obtained as a subset of the original set of vertex positions, as a set of weighted averages of original vertex positions, or by resampling the original piecewise linear surface. In the literature the different approaches are classified into

- Vertex clustering algorithms
- Incremental decimation algorithms
- Resampling algorithms

The first class of algorithms is usually very efficient and robust. The computational complexity is typically linear in the number of vertices. However, the quality of the resulting meshes is not always satisfactory. *Incremental algorithms* in most cases lead to higher quality meshes. The iterative decimation procedure can take arbitrary user-defined criteria into account, according to which the next removal operation is chosen. However, their total computation complexity in the average case is $O(n \log n)$ and can go up to $O(n^2)$ in the worst case, especially when a global error threshold is to be respected. Finally, *resampling techniques* are the most general approach to mesh decimation. Here, new samples are more or less freely distributed over the original piecewise linear surface geometry. By connecting these samples a completely new mesh is constructed. The major motivation for resampling techniques is that they can enforce the decimated mesh to have a special connectivity structure, i.e. subdivision connectivity (or semi-regular connectivity). By this they can be used in a straight forward manner to build multiresolution representations based on subdivision basis functions and their corresponding (pseudo-) wavelets [EDD*95]. The most serious disadvantage of resampling, however, is that *alias errors* can occur if the sampling pattern is not perfectly aligned to features in the original geometry. To avoid alias effects, many resampling schemes to some degree require manual pre-segmentation of the data for reliable feature detection. Resampling techniques will be discussed in detail in Section 10.

In the following sections we will explain the different approaches to mesh decimation in more detail. Usually there are many choices for the different ingredients and sub-procedures in each algorithm and we will point out the advantages and disadvantages for each class (see also [PGK02] for a comparison of different decimation techniques for point-sampled surfaces).

9.1. Vertex Clustering

The basic idea of vertex clustering is quite simple: for a given approximation tolerance ε we partition the bounding space around the given object into cells with diameter smaller than that tolerance. For each cell we compute a representative vertex position, which we assign to all the vertices that fall into that cell. By this clustering step, original faces degenerate if two or three of their corners lie in the same cell and consequently are mapped to the same position. The decimated mesh is eventually obtained by removing all those degenerate faces [RB93].

The remaining faces correspond to those original triangles whose corners all lie in different cells. Stated otherwise: if $\mathbf{p}$ is the representative vertex for the vertices $\mathbf{p}_0, \ldots, \mathbf{p}_n$ in the cluster P and $\mathbf{q}$ is the representative for the vertices $\mathbf{q}_0 \ldots, \mathbf{q}_m$ in the cluster Q then $\mathbf{p}$ and $\mathbf{q}$ are connected in the decimated mesh if and only if at least one pair of vertices $(\mathbf{p}_i, \mathbf{q}_j)$ was connected in the original mesh.

One immediately obvious draw-back of vertex clustering is that the resulting mesh might no longer be 2-manifold even if the original mesh was. Topological changes occur when the part of a surface that collapses into a single point is not homeomorphic to a disc, i.e., when two different sheets of the surface pass through a single ε-cell. However, this disadvantage can also be

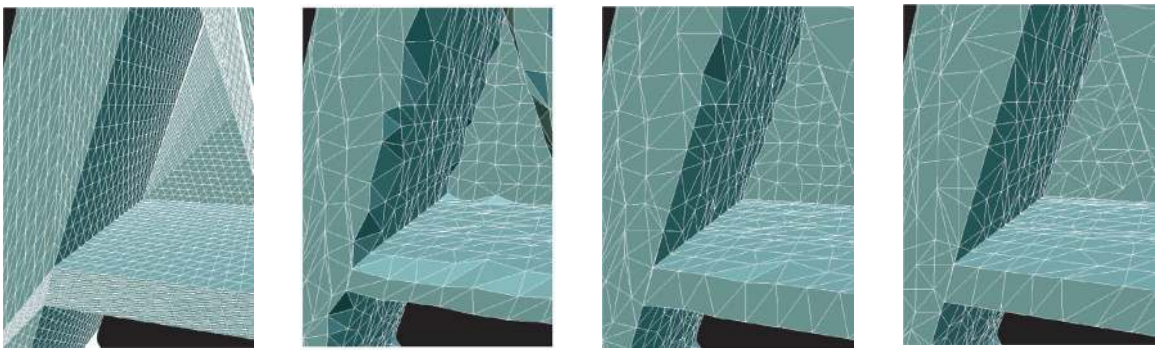


Figure 31: *Different choices for the representative vertex when decimating a mesh using clustering. From left to right: Original, average, median, quadric-based.*

considered as an advantage. Since the scheme is able to change the topology of the given model we can reduce the object complexity very effectively. Consider, e.g., applying mesh decimation to a 3D-model of a sponge. Here, any decimation scheme that preserves the surface topology cannot reduce the mesh complexity significantly since all the small holes have to be preserved.

The computational efficiency of vertex clustering is determined by the effort it takes to map the mesh vertices to clusters. For simple uniform spatial grids this can be achieved in linear time with small constants. Then for each cell a representative has to be found which might require fairly complicated computations but the number of clusters is usually much smaller than the number of vertices.

Another apparently nice aspect of vertex clustering is that it automatically guarantees a global approximation tolerance by defining the clusters accordingly. However, in practice it turns out that the actual approximation error of the decimated mesh is usually much smaller than the radius of the clusters. This indicates that for a given error threshold, vertex clustering algorithms do not achieve optimal complexity reduction. Consider, as an extreme example, a very fine planar mesh. Here decimation down to a single triangle without any approximation error would be possible. The result of vertex clustering instead will always keep one vertex for every ε-cell.

9.1.1. Computing Cluster Representatives

The way in which vertex clustering algorithms differ is mainly in how they compute the representative. Simply taking the center of each cell, the straight average, or the median of its members are obvious choices which, however, rarely lead to satisfying results (cf. Fig. 31).

A more reasonable choice is based on finding the optimal vertex position in the least squares sense. For this we exploit the fact that for sufficiently small ε the polygonal surface patch that lies within one ε-cell is expected to be piecewise flat, i.e., either the associated normal cone has a small opening angle (totally flat) or the patch can be split into a small number of sectors for which the normal cone has a small opening angle.

The optimal representative vertex position should have a minimum deviation from all the (regression) tangent planes that correspond to these sectors. If these approximate tangent planes do not intersect in a single point, we have to compute a solution in the least squares sense.

Consider one triangle t_i belonging to a specific cell, i.e., whose corner vertices lie in the same cell. The quadratic distance of an arbitrary point $\mathbf{x}$ from the supporting plane of that triangle can be computed by

$$(\mathbf{n}_i^T \mathbf{x} - d_i)^2 \quad ,$$

where $\mathbf{n}_i$ is the normal vector of t_i and d_i is the scalar product of $\mathbf{n}_i$ times one of t_i's corner vertices. The sum of the quadratic distances to all the triangle planes within one cell is given by

$$E(\mathbf{x}) \;=\; \sum_i (\mathbf{n}_i^T \mathbf{x} - d_i)^2 \quad . \tag{31}$$

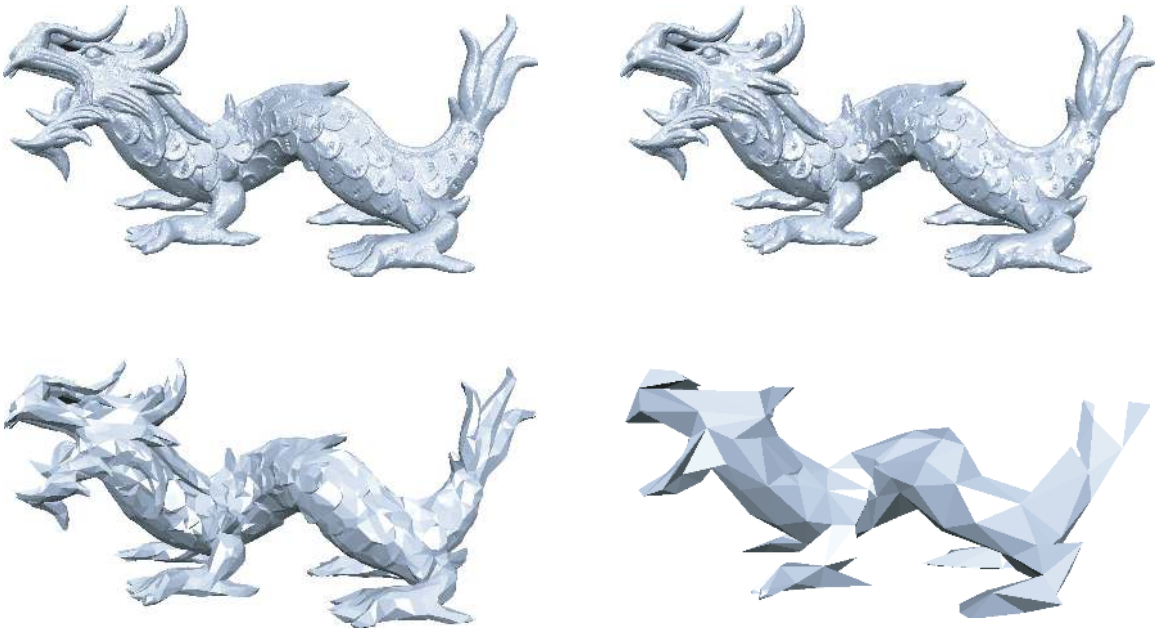


Figure 32: *Decimation of the dragon mesh consisting of 577.512 triangles (top left) to simplified version with* 10%*,* 1%*, and* 0.1% *of the original triangle count.*

The iso-contours of this error functional are ellipsoids and consequently the resulting error measure is called *quadric error metric (QEM)* [GH97, Lin00]. The point position where the quadric error is minimized is given by the solution of

$$\left(\sum_i \mathbf{n}_i \mathbf{n}_i^T\right) \mathbf{x} = \left(\sum_i \mathbf{n}_i d_i\right) \quad . \tag{32}$$

If the matrix has full rank, i.e. if the normal vectors of the patch do not lie in a plane, then the above equation could be solved directly. However, to avoid special case handling and to make the solution more robust, a pseudo-inverse based on a *singular value decomposition* should be used.

9.2. Incremental Mesh Decimation

Incremental algorithms remove one mesh vertex at a time (see Fig. 32). In each step, the best candidate for removal is determined based on user-specified criteria. Those criteria can be *binary* (= removal is allowed or not) or *continuous* (= rate the quality of the mesh after the removal between 0 and 1). Binary criteria usually refer to the global approximation tolerance or to other minimum requirements, e.g., minimum aspect ratio of triangles. Continuous criteria measure the *fairness* of the mesh in some sense, e.g., "round" triangles are better than thin ones, small normal jumps between neighboring triangles are better than large normal jumps.

Every time a removal has been executed, the surface geometry in the vicinity changes. Therefore, the quality criteria have to be re-evaluated. During the iterative procedure, this re-evaluation is the computationally most expensive part. To preserve the order of the candidates, they are usually kept in a *heap data structure* with the best removal operation on top. Whenever removal candidates have to be re-evaluated, they are deleted from the heap and re-inserted with their new value. By this, the complexity of the update-step increases only like $O(\log n)$ for large meshes if the criteria evaluation itself has constant complexity.

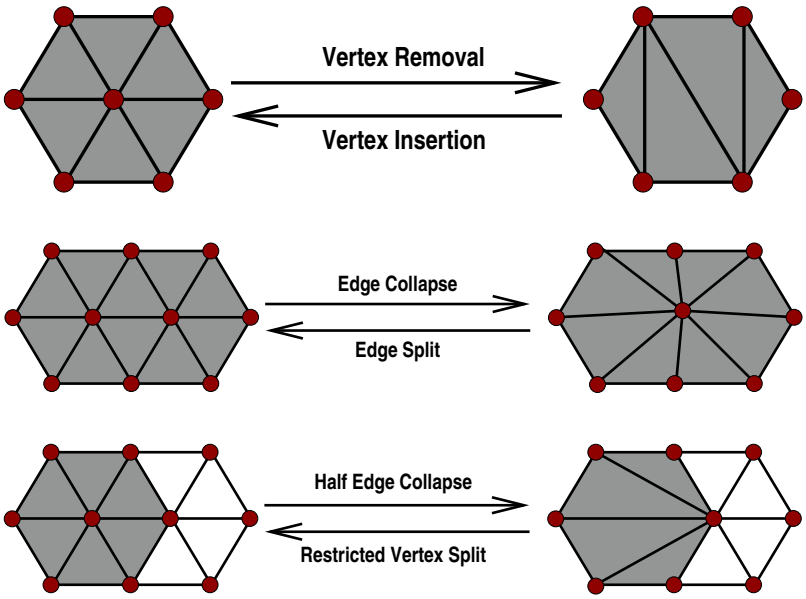


Figure 33: *Euler-operations for incremental mesh decimation and their inverses: vertex removal, full edge collapse, and half-edge collapse.*

9.2.1. Topological operations

There are several different choices for the basic removal operation. The major design goal is to keep the operation as simple as possible. In particular this means that we do not want to remove large parts of the original mesh at once but rather remove a single vertex at a time. Strong decimation is then achieved by applying many simple decimation step instead of a few complicated ones. If mesh consistency, i.e., topological correctness matters, the decimation operator has to be an *Euler-operator* (derived from the Euler formula for graphs) [HDD*93].

The first operator one might think of *deletes one vertex* plus its adjacent triangles. For a vertex with valence k this leaves a k-sided hole. This hole can be fixed by any polygon triangulation algorithm [SZL92]. Although there are several combinatorial degrees of freedom, the number of triangles will always be $k-2$. Hence the removal operation decreases the number of vertices by one and the number of triangles by two (cf. Fig. 33, top).

Another decimation operator takes two adjacent vertices $\mathbf{p}$, $\mathbf{q}$ and collapses the edge between them, i.e., both vertices are moved to the same new position $\mathbf{r}$ [Hop96] (cf. Fig. 33, middle). By this two adjacent triangles degenerate and can be removed from the mesh. In total this operator also removes one vertex and two triangles. The degrees of freedom in this *edge collapse* operator emerge from the freedom to choose the new position $\mathbf{r}$.

Both operators that we discussed so far are not unique. In either case there is some optimization involved to find the best local triangulation or the best vertex position. Conceptually this is not well-designed since it mixes the global optimization (which candidate is best according to the sorting criteria for the heap) with local optimization.

A possible way out is the so-called *half-edge collapse* operation: for an ordered pair ($\mathbf{p}$, $\mathbf{q}$) of adjacent vertices, $\mathbf{p}$ is moved to $\mathbf{q}$'s position [KCS98] (cf. Fig. 33, bottom). This can be considered as a special case of edge collapsing where the new vertex position $\mathbf{r}$ coincides with $\mathbf{q}$. On the other hand, it can also be considered as a special case of vertex deletion where the triangulation of the k-sided hole is generated by connecting all neighboring vertices with vertex $\mathbf{q}$.

The half-edge collapse has no degrees of freedom. Notice that ($\mathbf{p} \rightarrow \mathbf{q}$) and ($\mathbf{q} \rightarrow \mathbf{p}$) are treated as independent removal operations which both have to be evaluated and stored in the candidate heap. Since half-edge collapsing is a special case of the other two removal operations, one might expect an inferior quality of the decimated mesh. In fact, half-edge collapsing merely sub-samples the set of original vertices while the full edge collapse can act as a low-pass filter where new vertex positions are computed, e.g., by averaging original vertex positions. However, in practice this effect becomes noticeable only for extremely strong decimation where the exact location of individual vertices really matters.

The big advantage of half-edge collapsing is that for moderate decimation, the global optimization (i.e., candidate selection based on user specified criteria) is completely separated from the decimation operator which makes the design of mesh decimation schemes more orthogonal.

All the above removal operations preserve the mesh consistency and consequently the topology of the underlying surface. No holes in the original mesh can be closed, no handles can be eliminated completely. If a decimation scheme should be able to also simplify the topology of the input model, we have to use non-Euler removal operators. The most common operator in this class is the *vertex contraction* where two vertices $\mathbf{p}$ and $\mathbf{q}$ can be contracted into one new vertex $\mathbf{r}$ even if they are not connected by an edge [GH97, Sch97]. This operation reduces the number of vertices by one but it does keep the number of triangles constant. The implementation of mesh decimation based on vertex contraction requires flexible data structures that are able to represent non-manifold meshes since the surface patch around vertex $\mathbf{r}$ after the contraction might no longer be homeomorphic to a (half-)disc.

9.2.2. Distance measures

Guaranteeing an approximation tolerance during decimation is the most important requirement for most applications. Usually an upper bound ε is prescribed and the decimation scheme looks for the mesh with the least number of triangles that stays within ε to the original mesh. However, exactly computing the geometric distance between two polygonal mesh models is computationally expensive [KLS96, CRS98] and hence conservative approximations are used that can be evaluated quickly.

The generic situation during mesh decimation is that each triangle t_i in the decimated mesh is associated with a sub-patch $\mathcal{S}_i$ of the original mesh. Distance measures have to be computed between each triangle t_i and either the vertices or faces of $\mathcal{S}_i$. Depending on the application, we have to take the maximum distance or we can average the distance over the patch.

The simplest technique is error accumulation [SZL92]. For example each edge collapse operation modifies the adjacent triangles t_i by shifting one of their corner vertices from $\mathbf{p}$ or $\mathbf{q}$ to $\mathbf{r}$. Hence the distance of $\mathbf{r}$ to t_i is an upper bound for the approximation error introduced in this step. Error accumulation means that we store an error value for each triangle and simply add the new error contribution for every decimation step. The error accumulation can be done based on scalar distance values or on distance vectors. Vector addition takes the effect into account that approximation error estimates in opposite directions can cancel each other.

Another distance measure assigns distance values to the vertices $\mathbf{p}_j$ of the decimated mesh. It is based on estimating the squared average of the distances of $\mathbf{p}_j$ from all the supporting planes of triangles in the patches $\mathcal{S}_i$ which are associated with the triangles t_i surrounding $\mathbf{p}_j$. This is, in fact, what the quadric error metric does [GH97].

Initially we compute the error quadric E_j for each original vertex $\mathbf{p}_j$ according to (31) by summing over all triangles which are directly adjacent to $\mathbf{p}_j$. Since we are interested in the *average* squared distance, E_j has to be normalized by dividing through the valence of $\mathbf{p}_j$ Then, whenever the edge between two vertices $\mathbf{p}$ and $\mathbf{q}$ is collapsed, the error quadric for the new vertex $\mathbf{r}$ is found by $E_r = (E_p + E_q)/2$.

The quadric error metric is evaluated by computing $E_j(\mathbf{p}_j)$. Hence when collapsing $\mathbf{p}$ and $\mathbf{q}$ into $\mathbf{r}$, the optimal position for $\mathbf{r}$ is given by the solution of (32). Notice that due to the averaging step the quadric error metric does neither give a strict upper nor a strict lower bound on the true geometric error.

Finally, the most expensive but also the sharpest distance error estimate is the *Hausdorff-distance* [KLS96]. This distance measure is defined to be the maximum minimum distance, i.e., if we have two sets $\mathcal{A}$ and $\mathcal{B}$ then $H(\mathcal{A},\mathcal{B})$ is found by computing the minimum distance $d(\mathbf{p},\mathcal{B})$ for each point $\mathbf{p} \in \mathcal{A}$ and then taking the maximum of those values. Notice that in general $H(\mathcal{A},\mathcal{B}) \neq H(\mathcal{B},\mathcal{A})$ and hence the *symmetric Hausdorff-distance* is the maximum of both values.

If we assume that the vertices of the original mesh represent sample points measured on some original geometry then the faces have been generated by some triangulation pre-process and should be considered as piecewise linear approximations to the original shape. From this point of view, the correct error estimate for the decimated mesh would be the one-sided Hausdorff-distance $H(\mathcal{A},\mathcal{B})$ from the original sample points $\mathcal{A}$ to the decimated mesh $\mathcal{B}$.

To efficiently compute the Hausdorff-distance we have to keep track of the assignment of original vertices to the triangles of the decimated mesh. Whenever an edge collapse operation is performed, the removed vertices $\mathbf{p}$ and $\mathbf{q}$ (or $\mathbf{p}$ alone in the case of a half-edge collapse) are assigned to the nearest triangle in a local vicinity. In addition, since the edge collapse changes the shape of the adjacent triangles, the data points that previously have been assigned to these triangles, must be re-distributed. By this, every triangle t_i of the decimated mesh at any time maintains a list of original vertices belonging to the currently associated patch $\mathcal{S}_i$. The Hausdorff-distance is then evaluated by finding the most distant point in this list.

A special technique for exact distance computation is suggested in [CVM*96], where two offset surfaces to the original mesh are computed to bound the space where the decimated mesh has to stay in.

9.2.3. Fairness criteria

The distance measures can be used to decide which removal operation among the candidates is legal and which is not (because it violates the global error threshold ε). In an incremental mesh decimation scheme we have to provide an additional criterion which ranks all the legal removal operations. This criterion determines the ordering of the candidates in the heap.

One straightforward solution is to use the distance measure for the ordering as well. This implies that the decimation algorithm will always remove that vertex in the next step that increases the approximation error least. While this is a reasonable heuristic in general, we can use other criteria to optimize the resulting mesh for special application dependent requirements.

For example, we might prefer triangle meshes with faces that are as close as possible to equilateral. In this case we can measure the quality of a vertex removal operation, e.g., by the *longest edge to inner circle radius ratio* of the triangles after the removal.

If we prefer visually smooth meshes, we can use the maximum or average normal jump between adjacent triangles after the removal as a sorting criterion. Other criteria might include color deviation or texture distortion if the input data does not consist of pure geometry but also has color and texture attributes attached [CMR*99, COM98, GH98].

All these different criteria for sorting vertex removal operations are called *fairness criteria* since they rate the quality of the mesh beyond the mere approximation tolerance. If we keep the fairness criterion separate from the other modules in an implementation of incremental mesh decimation, we can adapt the algorithm to arbitrary user requirement by simply exchanging that one procedure. This gives rise to a flexible tool-box for building custom tailored mesh decimation schemes [EDD*95].

9.3. Out-of-core Methods

Mesh decimation is frequently applied to very large data sets that are too complex to fit into main memory. To avoid severe performance degradation due to virtual memory swapping, *out-of-core* algorithms have been proposed that allow an efficient decimation of polygonal meshes without requiring the entire data set to be present in main memory. The challenge here is to design suitable data structures that avoid random access to parts of the mesh during the simplification.

Lindstrom [Lin00] presented an approach based on vertex clustering combined with quadric error metrics for computing the cluster representatives (see Section 9.1). This algorithm only requires limited connectivity information and processes meshes stored as a triangle soup, where each triangle is represented as a triplet of vertex coordinates. Using a single pass over the mesh data an in-core representation of the simplified mesh is build incrementally. A dynamic hash table is used for fast localization and quadrics associated with a cluster are aggregated until all triangles have been processed. The final simplified mesh is then produced by computing a representative from the per-cluster quadrics and the corresponding connectivity information as described above.

Lindstrom and Silva [LS01] improve on this approach by removing the requirement for the output model to fit into main memory by using a multi-pass approach. Their method only requires a constant amount of memory that is independent of the size of the input and output data. This improvement is achieved by a careful use of (slower, but cheaper) disk space, which typically leads to performance overheads between a factor of two and five as compared to [Lin00]. To avoid storing the list of occupied clusters and associated quadrics in main memory, the required information from each triangle to compute the quadrics is stored to disk. This file is then sorted according to the grid locations using an external sort algorithm. Finally, quadrics and final vertex positions are computed in a single linear sweep over the sorted file. The authors also apply a scheme similar to the one proposed in [GH97] to better preserve boundary edges.

Wu and Kobbelt [WK04] proposed an streaming approach to out-of-core mesh decimation based edge collapse operations in connection with quadric error metric. Their method uses a fixed-size active working set and is independent of the input and output model complexity. In contrast to the previous two approaches for out-of-core decimation, their method allows to prescribe the size of the output mesh exactly and supports explicit control over the topology during the simplification. The basic idea is to sequentially stream the mesh data and incrementally apply decimation operations on an active working set that is kept in main memory. Assuming that the geometry stream is approximately pre-sorted, e.g., by one coordinate, the spatial coherency then guarantees that the working set can be small as compared to the total model size (see Fig. 34) For decimation they apply randomized multiple choice optimization, which has been shown to produce results of similar quality than the standard greedy optimization. The idea is to select a small random set of candidate edges for contraction and only collapse the edge with smallest quadric error. This significantly reduces computation costs, since no global heap data structure has to be maintained during the simplification process. In order to avoid inconsistencies during the simplification, edges can only be collapsed, if they are not part of the boundary between the active working set and the parts of the mesh that are held out-of-core. Since no

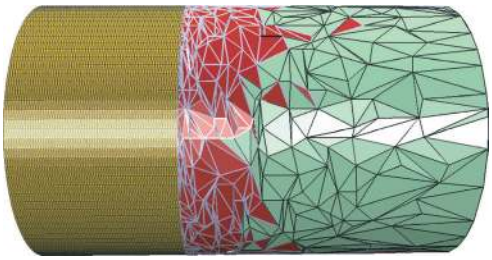


Figure 34: *This snapshot of a stream decimation shows the yet unprocessed part of the input data (left), the current in-core portion (middle) and the already decimated output (right). The data in the original file happened to be pre-sorted from right to left (from [WK04]).*

global connectivity information is available, this boundary cannot be distinguished from the actual mesh boundary of the input model. Thus the latter can only be simplified after the entire mesh has been processed, which can be problematic for meshes with large boundaries.

Isenburg et al. introduces *mesh processing sequences*, which represent a mesh as a fixed interleaved sequence of indexed vertices and triangles [ILGS03]. Processing sequences can be used to improve the out-of-core decimation algorithms described above. Both memory efficiency and mesh quality are improved for the vertex clustering method of [Lin00], while increased coherency and explicit boundary information help to reduce the size of the active working set in [WK04].

Shaffer and Garland [SG01] proposed a scheme that combines an out-of-core vertex clustering step with an in-core iterative decimation step. The central observation, which is also the rationale behind the randomized multiple choice optimization, is that the exact ordering of edge collapses is only relevant for very coarse approximations. Thus the decimation process can be simplified by combining many edge collapse operations into single vertex clustering operations to obtain an intermediate mesh, which then serves as input for the standard greedy decimation (Section 9.2). Shaffer and Garland use quadric error metrics for both types of decimation and couple the two simplification steps by passing the quadrics computed during clustering to the subsequent iterative edge collapse pass. This coupling achieves significantly improvements when compared to simply applying the two operations in succession.

10. Remeshing

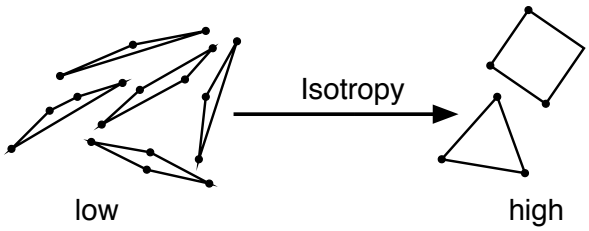
Remeshing is a key technique for mesh quality preservation in many geometric modeling algorithms, e.g., shape editing, animation, morphing, and numerical simulation. As such, it has received considerable attention in recent years and a wealth of remeshing algorithms have been developed. The goal of remeshing is on the one hand to reduce the complexity of an input mesh subject to certain quality criteria. This process is better known as *mesh decimation* or *mesh simplification,* a topic that is covered in Section 9 in more detail. The second goal of remeshing is to *improve the quality* of a model, such that it can be used as input for various downstream applications. Different applications, of course, imply different quality criteria and requirements. This corresponds to the following basic definition of the term "remeshing" that was given by Alliez et al. in their survey paper [AUGA05] (which we highly recommend as a reference for further reading on the topic): *Given a 3D mesh, compute another mesh whose elements satisfy some quality requirements, while approximating well the input.* Here the term approximation can be understood with respect to location as well as normal orientation.

In contrast to general mesh repair (see Section 4), the input of remeshing algorithms is usually assumed to already be a (part of a) manifold triangle mesh. The term mesh quality thus refers to non-topological properties, like the sampling density, regularity, size, alignment, orientation, and shape of the mesh elements. This section will in particular deal with these latter aspects of remeshing and present various methods that achieve this goal.

We will begin our discussion by structuring the different types of remeshing algorithms and by clarifying some concepts that are commonly used in the remeshing literature. In the following sections we will then discuss *isotropic* and *anisotropic* remeshing methods in more detail.

Local Structure The local structure of a mesh is described by the type, shape, orientation, and distribution of the mesh elements.

- *Element type:* The most common target element types in remeshing are *triangles* and *quadrangles.* Triangle meshes are usually easier to produce, while in quadrangular remeshing one often has to content oneself with results that are only *quad-dominant*. Note that in principle any quadrangle mesh can be converted trivially into a triangle mesh by inserting a diagonal into each quadrangle. On the other hand, triangle meshes can be converted to quads by splitting each triangle at its barycenter into three new triangles (1-to-3 split) and discarding the original mesh edges.
- *Element shape:* Elements can be classified as being either *isotropic* or *anisotropic*. The shape of isotropic elements is close to circular, thus a triangle/quadrangle is isotropic if it is close to equilateral/square. For triangles this "roundness" can be measured by dividing the length of the shortest edge by the radius of the circumcircle, see [She97]. Anisotropic elements typically are aligned with the principal curvature directions of the surface (see Section 5). Anisotropic meshes usually need fewer elements than their isotropic pendants to achieve the same approximation quality. Furthermore anisotropic elements better express the structure of geometric primitives (plane, cylinders, spheres, ...) inherent in many technical models. Isotropic elements on the other hand are sometimes favored in numerical applications (FEM), as the uniform shape of their elements often leads to a better conditioning of the resulting systems, see [She02] for a more detailed discussion.



- *Element density:* In a *uniform* distribution, the mesh elements are evenly spread across the entire model. In a *non-uniform* or *adaptive* distribution, the number of element varies, e.g., more and smaller elements are assigned to curved or to feature regions than to flat areas. Adaptive meshes need significantly fewer elements to achieve an approximation quality that is comparable to that of uniform meshes.
- *Element alignment:* Converting a piecewise smooth input surface into a (re-)mesh corresponds to a (re-)sampling process. Hence sharp features may be affected by alias-artifacts. In order to prevent this, elements should be aligned to sharp features such that they properly represent tangent discontinuities.

Global Structure A vertex in a triangle mesh is called *regular,* if its valence (i.e., number of neighboring vertices) is 6 for interior vertices or 4 for boundary vertices. In quadrangle meshes, the regular valences are 4 and 3, respectively. Vertices that are not regular are called *irregular*, *singular*, or *extraordinary.*

The global structure of a remesh can be classified as being either *completely regular, semi-regular, highly regular* or *irregular.*

- In a *completely regular* mesh all vertices are regular. A regular mesh can compactly be stored in a two-dimensional array which can be used to speed up the visualization (a so-called *geometry image*), see [GGH02, SWG*03, LHSW03].
- *Semi-regular* meshes are produced by regular subdivision of a coarse initial mesh. Thus the number of extraordinary vertices in a semi-regular mesh is small and constant [EDD*95, GVSS00, LSS*98, KVLS99b] under uniform refinement.
- In *highly regular* meshes most vertices are regular. In contrast to semi-regular meshes, highly regular meshes need not be the result of a subdivision process [SKR02, SG03b, AMD02, SAG03].
- *Irregular* meshes do not exhibit any kind of regularities in their connectivity.

Besides this topological characterization, the suitability of a remeshing algorithm usually depends on its ability to capture the global structure of the input geometry by aligning groups of elements to the dominant geometric features. Since this corresponds to the alignment of entire submeshes, e.g., to global curvature lines of geometric primitives, it is strongly related to mesh segmentation techniques [MK06].

Fully regular meshes can be generated only for a very limited number of input models, namely those that topologically are (part of) a torus. All other models have to be cut into one or more topological disks before processing (and then the global regularity is broken at the seams). Furthermore, special care has to be taken to correctly identify and handle the seams that result from the cutting. Semi-regular meshes are in particular suitable for multi-resolution analysis and modeling [ZSS97, GVSS00]. They define a natural parameterization of a model over a coarse base mesh. Thus, some algorithms for semi-regular remeshing are described in Section 8. Highly regular meshes require different techniques for multi-resolution analysis, but still they are well-suited for numerical simulations. In particular, mesh compression algorithms can take advantage of the mostly uniform valence distribution and produce a very efficient connectivity encoding [TG98].

Correspondences All remeshing algorithms compute sample points on or near the original surface. Most algorithms furthermore iteratively relocate sample points in order to improve the quality of the mesh. Thus, a key issue in all remeshing algorithms is to compute or maintain correspondences between sample points $\mathbf{p}$ on the remesh and their counterparts $\phi(\mathbf{p})$ on the input mesh. There are a number of approaches to address this problem:

- *Global parameterization:* The input model is globally parameterized onto a 2D domain. Sample points can then be easily distributed and relocated in the 2D domain and later be "lifted" to 3D.
- *Local parameterization:* The algorithm maintains a parameterization of a local geodesic neighborhood around $\phi(\mathbf{p})$. When the sample leaves this neighborhood, a new neighborhood has to be computed.
- *Projection:* The sample point is directly projected to the nearest point on the input model.

Global parameterization is generally expensive and may suffer from parametric distortion. Naive direct projection may produce local and global fold overs if the points are too far away from the surface. However, in practice the projection operator can be stabilized by constraining the movement of the sample points to their tangent planes. Although no theoretical guarantees can be provided, this makes sure that the samples do not move away too far from the surface, such that the projection can safely be evaluated. The local parameterization approach is stable and produces currently the best results, however, it needs expensive bookkeeping to track, cache, and reparameterize the local neighborhoods.

10.1. Isotropic Remeshing

In an isotropic mesh all triangles are approximately equilateral. One may further require a globally uniform vertex density or allow a smooth change in the triangle sizes (*gradation*). There are a number of algorithms for isotropic remeshing of triangle meshes:

- Turk proposed one of the first remeshing algorithms [Tur92]. The algorithm places a number of points on the input mesh and then uniformly distributes them by an attraction-repulsion mechanism. The algorithm works well, but has problems with models that contain sharp features or thin structures.
- Alliez et al. proposed an *interactive remeshing* technique [AMD02]. This algorithm first computes a global conformal parameterization of the input mesh and then uses an error-diffusion (dithering) technique to place new points in the 2D parameter domain. These points are then connected and lifted back to 3D. The algorithm allows to interactively control various remeshing criteria, e.g., to increase the sample density in highly curved regions or to incorporate feature edges. On the downside, due to the parameterization this algorithm can only work on genus-0 meshes that are not too much distorted. Otherwise the model has to be cut into topological disks.

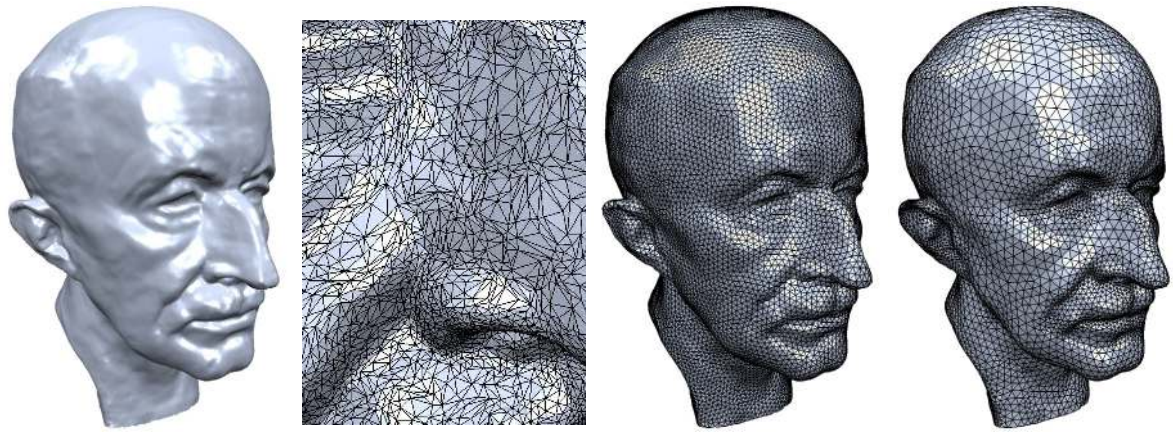


Figure 35: *Isotropic remeshing. Left and center left: Max Planck model at full resolution. Center right and right: Uniform and adaptive remeshes.*

- Surazhsky and Gotsman proposed an explicit surface remeshing algorithm that iterates through a sequence of area-based smoothing, regularization and angle-based smoothing steps to achieve an isotropic remeshing of the input model [SG03b]. The algorithm uses parameterized, local patches [VRS03] that overlap each other.
- A number of algorithms that are based on the computation of a generalized centroidal Voronoi diagram by Lloyd relaxation have been proposed recently [SAG03, AdVDI03, VC04, PC04]. The main idea is to iteratively move vertices to the center of their (geodesic) Voronoi cell. These algorithms achieve high-quality results, but typically need local parameterizations of neighborhoods and thus are somewhat harder to implement.
- Another class of remeshing schemes is based on global parameterizations. In [KLS03] triangle meshes are considered w.r.t. a base domain, while Ray et al. [RLL*05] generate quad-meshes. The recent semi-automatic approach in [TACSD06] uses a linear framework based on discrete harmonic forms. In [DBG*06] quad-meshes are generated based on spectral analysis (see also Section 7), which leads to a setup similar to [KLS03].

In this section we present a simple but efficient remeshing algorithm that produces isotropic triangle meshes. The algorithm was presented in [BK04b] and is a simplified version of [VRS03] and an extension to [KBS00]. It produces results that are comparable to the ones by the original algorithm, but has the advantage of being simpler to implement. In particular, it does not need a (global or local) parameterization or the involved computation of (geodesic) Voronoi cells as, e.g., [SAG03]. The algorithm takes as input a target edge length and then repeatedly splits long edges, collapses short edges, and relocates vertices until all edges are approximately of the desired target edge length. Thus the algorithm runs the following loop:

```
remesh(target_edge_length)
  low = 4/5 * target_edge_length
  high = 4/3 * target_edge_length
  for i = 0 to 10 do
    split_long_edges(high)
    collapse_short_edges(low,high)
    equalize_valences()
    tangential_relaxation()
    project_to_surface()
```

Notice that the proper thresholds $\frac{4}{5}$ and $\frac{4}{3}$ are essential to converge to a uniform edge length [BK04b]. The values are derived from considerations to make sure that the edge lengths are closer to the target lengths after a split or collapse operation than before. A hysteresis behavior is induced by the interleaved tangential smoothing operator.

The split_long_edges(high) function visits all edges of the current mesh. If an edge is longer than the given threshold high, the edge is split at its midpoint and the two adjacent triangles are bisected (2-4 split).

```
split_long_edges(high)
  while exists edge e with length(e)>high do
    split e at midpoint(e)
```

The collapse_short_edges(low, high) function collapses and thus removes all edges that are shorter than a threshold low. Here one has to take care of a subtle problem: by collapsing along chains of short edges the algorithm may create new edges that are arbitrarily long and thus undo the work that was done in split_long_edges(high). This issue is resolved by testing *before* each collapse whether the collapse would produce an edge that is longer than high. If so, the collapse is not executed.

```
collapse_short_edges(low, high)
  finished = false
  while exists edge e with length(e)<low and not finished do
    finished = true
    let e=(a,b) and let a[1],...,a[n] be the 1-ring of a
    collapse_ok = true
    for i = 1 to n do
      if length(b,a[i])>high then
        collapse_ok = false
    if collapse_ok then
      collapse a into b along e
      finished = false
```

The equalize_valences() function equalizes the vertex valences by flipping edges. The target valence target_val(v) is 6 and 4 for interior and boundary vertices, respectively. The algorithm tentatively flips each edge e and checks whether the deviation to the target valences decreases. If not, the edge is flipped back.

```
equalize_valences()
  for each edge e do
    let a,b,c,d be the vertices of the two triangles adjacent to e
    deviation_pre = abs(valence(a)-target_val(a)) + abs(valence(b)-target_val(b))
                  + abs(valence(c)-target_val(c)) + abs(valence(d)-target_val(d))
    flip(e)
    deviation_post = abs(valence(a)-target_val(a)) + abs(valence(b)-target_val(b))
                   + abs(valence(c)-target_val(c)) + abs(valence(d)-target_val(d))
    if deviation_pre ≤ deviation_post do
      flip(e)
```

The tangential_relaxation() function applies an iterative smoothing filter to the mesh. Here the vertex movement has to be constrained to the vertex' tangent plane in order to stabilize the following projection operator. Let $\mathbf{p}$ be an arbitrary vertex in the current mesh, let $\mathbf{n}$ be its normal, and let $\mathbf{q}$ be the position of the vertex as calculated by a smoothing algorithms with uniform Laplacian weights (see Section 7). The new position $\mathbf{p}'$ of $\mathbf{p}$ is then computed by projecting $\mathbf{q}$ onto $\mathbf{p}$'s tangent plane

$$\mathbf{p}' = \mathbf{q} + \mathbf{n}\mathbf{n}^T(\mathbf{p} - \mathbf{q}) \ .$$

Again, this can be easily implemented:

```
tangential_relaxation()
  for each vertex v do
    q[v] = the barycenter of v's neighbor vertices
  for each vertex v do
    let p[v] and n[v] be the position and normal of v, respectively
    p[v] = q[v] + dot(n[v],(p[v]-q[v]))*n[v]
```

Finally, the project_to_surface() function maps the vertices back to the surface.

Feature preservation A few simple rules suffice to make sure that the remeshing algorithm preserves the features of the input model, see Fig. 36. Here we assume, that the feature edges and vertices have already been marked in the input model, e.g., by automatic feature detection algorithms or by manual specification [VRS03, Bot05b].

- Corner vertices with more than two or exactly one incident feature edge have to be preserved and are excluded from all topological and geometric operations.
- Feature vertices may only be collapsed along their incident feature edges.
- Splitting a feature edge creates two new feature edges and a feature vertex.
- Feature edges are never flipped.
- Tangential smoothing of feature vertices is restricted to univariate smoothing along the corresponding feature lines.

Discussion As can be seen in Fig. 35 and Fig. 36, the algorithm above produces quite good results. It is also possible to incorporate additional regularization terms by adjusting the weights that are used in the smoothing phase. This allows to achieve a uniform triangle area distribution or to implement an adaptive remeshing algorithm that produces finer elements in regions of high curvature.

Figure 36: *Isotropic, feature sensitive remeshing of a CAD model.*

10.2. Anisotropic Remeshing

In an anisotropic mesh the elements align to the principal curvature directions, i.e., they are elongated along the minimum curvature direction and shortened along the maximum curvature direction (see Section 5). Anisotropic triangle remeshes of a given target complexity can easily be produced by incrementally decimating the input model down to a desired target complexity. No matter whether one uses quadric error metrics, (one-sided) Hausdorff-distance, or the normal deviation to rank the priorities of removal operations, the result will always be an anisotropic triangle mesh that naturally aligns to the principal curvature directions. The remeshes that are produced by this method satisfy the definition of being anisotropic, but unfortunately they do not convey the orthogonal structure of the curvature lines. To produce such a structure, it is usually better to first compute a quadrangular remesh. If necessary, this quadrangle mesh can then be triangulated, e.g., by inserting the shorter of the two diagonals of each quadrangle. There are a number of approaches for producing quadrangular remeshes of an input model. In the following sections, we will elaborate on a method that tracks an orthogonal net of principal curvature lines on a mesh to produce an anisotropic quadrangulation that is aligned to the curvature directions.

Recently, alternative approaches based on global parameterizations have been proposed, which are slightly less flexible due to their global dependences, but produce guaranteed all quad meshes. The curvature line tracing based techniques only produce quad-dominant meshes.

10.2.1. Anisotropic Polygonal Remeshing

In this section we describe a remeshing algorithm proposed by Alliez et al. [ACD*03]. In curved regions, it produces an anisotropic, quad-dominant remesh, while in flat regions an isotropic, Delaunay-like remeshing is created. Let $\mathcal{M}$ be the input which is assumed to be a genus-0, manifold triangle mesh with a single boundary loop. The algorithm proceeds in four stages:

1. *Parameterization:* In a preprocessing step, a global conformal parameterization $\phi : \Omega \rightarrow \mathcal{M}$ of the input mesh $\mathcal{M}$ is computed. This can be done by one of the methods presented in Section 8.
2. *Curvature Tensor Estimation:* A 3×3 curvature tensor is estimated for each vertex $v \in \mathcal{M}$ by averaging fundamental curvature tensors for each edge in a geodesic neighborhood around v (see Section 5). The two principal directions are derived from the curvature tensors and "flattened" into the parameter domain Ω. Finally, the vector fields are smoothed and the umbilic points are extracted.
3. *Resampling:* In anisotropic regions, a network of lines of curvature is traced within the parameter domain. The density of the network is controlled by a user-given error threshold and the local curvature estimates. In isotropic regions the algorithm produces a uniform distribution of sample points.
4. *Meshing:* The vertices of the final mesh are the union of the intersection points of curvature lines and the vertices that were scattered in the isotropic regions of the mesh. The edges of the final mesh are deduced from a straightening of the lines of curvature that were traced in step 3 and from a Delaunay triangulation of the vertices in the isotropic regions.

Curvature Tensor Estimation A 3×3 curvature tensor $\mathbf{T}(v)$ is estimated for each vertex $v \in \mathcal{M}$ by averaging edge-based curvature tensors in a geodesic neighborhood around v. Working with symmetric 3×3 matrices is particularly convenient since they can easily be averaged while still remaining symmetric. The eigenvalues and eigenvectors of $\mathbf{T}$ are good estimates of the surface normal in v, the two principal directions, and the corresponding principal curvatures κ_1 and κ_2 as described in Section 5. For efficiency reasons the tensor $\mathbf{T}$ is *pulled back* to the 2D parameter domain resulting in a 2D tensor $\mathbf{Q} = \mathbf{Q}(u,v)$ such that

$$\mathbf{Q} = \mathbf{P}^T \begin{bmatrix} \kappa_1 & 0 \\ 0 & \kappa_2 \end{bmatrix} \mathbf{P} \ ,$$

where $\mathbf{P} = \mathbf{P}(u,v)$ is an orthogonal matrix that encodes the two principal directions in parameter space. The 2D tensor field is then linearly interpolated across each triangle in parameter space. If a coarse remeshing of the input model is desired, it additionally might become necessary to smooth the tensor field, e.g., by a circular, isotropic Gaussian filter.

Finally the umbilic points are extracted. Umbilic points correspond to isotropic regions that do not have two distinct principal curvature directions, i.e., that are either flat or spherical. Thus an umbilic point (u,v) is characterized by

$$\mathbf{Q}(u,v) = \begin{bmatrix} \kappa & 0 \\ 0 & \kappa \end{bmatrix} \ .$$

It can be shown that there is at most one umbilic point per triangle. This point can be found by solving a linear 2×2 system.

Resampling In the second phase, the input mesh is resampled. Orthogonal lines of curvature are traced to resample the anisotropic regions of the input mesh. Spherical and flat regions are simply resampled by a set of evenly distributed points. Note that the whole resampling process takes place in the 2D parameter domain Ω.

Let $\mathbf{Q}(u,v)$ be the "flattened" metric tensor as described above. Then the eigenvector $\mathbf{q}_1(u,v)$ associated with the smallest eigenvalue κ_1 points in the direction of maximal (!) curvature. Let

$$\mathbf{c} : t \mapsto \mathbf{c}(t) = (u(t), v(t))$$

be a curve such that $\mathbf{c}'(t)$ is parallel to $\mathbf{q}_1(\mathbf{c}(t))$ for all t, then $\mathbf{c}(\cdot)$ is called a *line of maximum curvature*. Analogously, one can define *lines of minimum curvature*. Note that lines of maximum and minimum curvature always intersect orthogonally. A line of curvature is either closed or it starts and ends in umbilic points or at the domain boundary. After selecting a seed (u_0, v_0) the lines of curvature that pass through that seed can be traced, e.g., by an embedded fourth-order Runge-Kutta scheme with adaptive step size.

The optimal spacing between two lines of the same curvature type are determined as follows: Let us assume that the piecewise linear reconstruction between these two lines should not deviate by more than a user-given error threshold ε from the true surface. Alliez et al. derive the following formula for the spacing between the two curvature lines:

$$d(\kappa) = 2\sqrt{\varepsilon\left(\frac{2}{|\kappa|} - \varepsilon\right)} \ .$$

Thus for any point on a line of maximum curvature, the optimal distance to the next line of maximum curvature is $d_2 = d(\kappa_1)$, and analogously for lines of minimum curvature. As all computations are done in the 2D parameter domain these distances have to be multiplied by the local area stretching of the parameterization ϕ.

To actually sample the curvature lines, the algorithm maintains a list of *potential seeds*. Initially, this list contains all umbilic points. Then one iteratively selects a seed (u_0, v_0) from the list and starts tracing curvature (poly-)lines $(u_0, v_0), (u_1, v_1), \ldots$ as described above. For each of the vertices (u_i, v_i) two new seed vertices are placed orthogonally to the current curve at the optimal distance from the curvature line. Line tracing ends once

- the line reaches an umbilic point or the domain boundary or
- the line comes back to its starting seed or
- the line comes too close to an existing line of same curvature type

Meshing Eventually a mesh is extracted from the net of curvature lines. The vertices of the mesh are made of the intersections of the curvature lines and the points that were scattered in the flat and spherical regions of the model. A decimation process removes all superfluous samples, like dangling edges or vertices that are adjacent to exactly two line segments of the same type. Finally a Delaunay triangulation of the isotropic samples is produced and its edges are added to the output mesh. Note that during the decimation the algorithm makes sure that no fold-overs are produced.

Implementation The algorithm described above is best implemented using a *constrained Delaunay triangulation (CDT).* Initially, the boundary of the parameter domain Ω is triangulated. When tracing lines of curvature $(u_0, v_0), (u_1, v_1), \ldots$ the line segments are inserted into the CDT. The CDT in particular provides a fast nearest neighbor query that is needed in the re-sampling phase and an efficient way to compute the intersection points of the curvature lines in the meshing phase. As the implementation of a robust CDT is numerically non-trivial and requires robust predicates, an advanced geometry kernel, e.g., provided by CGAL, should be used.

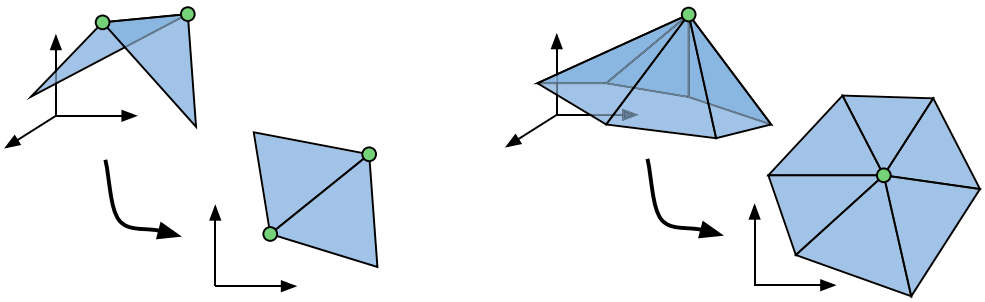
10.2.2. Direct Anisotropic Remeshing

Marinov and Kobbelt [MK04] propose a variant of Alliez et al.'s algorithm, that differs from the original work in two aspects (cf. Fig. 37):

- Curvature line tracking and meshing are all done in 3D space, so there is no need to compute a global parameterization such that objects of arbitrary genus can be processed.
- The algorithm is able to compute a quad-dominant, anisotropic remesh even in flat regions of the model, where there are no reliable curvature estimates by extrapolating directional information from neighboring anisotropic regions.

Curvature Estimation In addition to mere curvature directions, a confidence value for each face and vertex of the input mesh is estimated as well. The estimate is based on the coherence of the principal directions at the face's vertices. This confidence estimate is then used to propagate the curvature tensors from regions of high confidence (highly curved regions) into regions of low confidence (flat regions and noisy regions).

Resampling Curvature lines are traced directly on the 3D mesh, i.e., at any time a line sample position is identified by a tuple $(f, (u, v, w))$ where f is the index of a triangle and (u, v, w) are the barycentric coordinates of the sample within that triangle. To advance the current sample point, the face f and its neighborhood are locally flattened, either by a hinge map (if the curvature line crosses an edge of f) or by a polar map (if the curvature line crosses one of f's vertices).



Hinge map Polar map

When the traced line enters a region of low confidence, the algorithm switches the tracing mode: Instead of integrating along the principal curvatures, the line is simply extrapolated from its last sample points along a geodesic curve until it enters a region of high confidence again. At this point the line is then "snapped" to the most similar principal curvature direction.

In [DKG05], similar streamline tracing is used to generate quad-meshes. Here, vector fields are not based on curvature but instead defined by gradients of (user specified) harmonic functions, and streamlines are integrated on the surface.

10.2.3. Discussion

Due to the strong visual and structural importance of curvatures, remeshing algorithms that track these lines produce results that are similar to those that would have been created by a human designer. However, reliably estimating and tracking the principal curvatures on a discrete triangle mesh is not that easy, in particular for coarse or noisy meshes. Alliez et al.'s algorithm outsources most of the computationally hard work to a constrained Delaunay triangulation (e.g. the one provided by CGAL) by globally paramaterizing the whole input model. Apart from being hard to compute for large models, a global parameterization restricts the inputs to genus-0 manifolds with a single boundary loop. Higher genus objects have to be cut open along each handle. The approach of Marinov et al. is parameterization-free and has no restrictions on the topology of the input model. However, the extraction of the final mesh might lead to non-manifold configurations that have to be handled and fixed in a post-processing step.

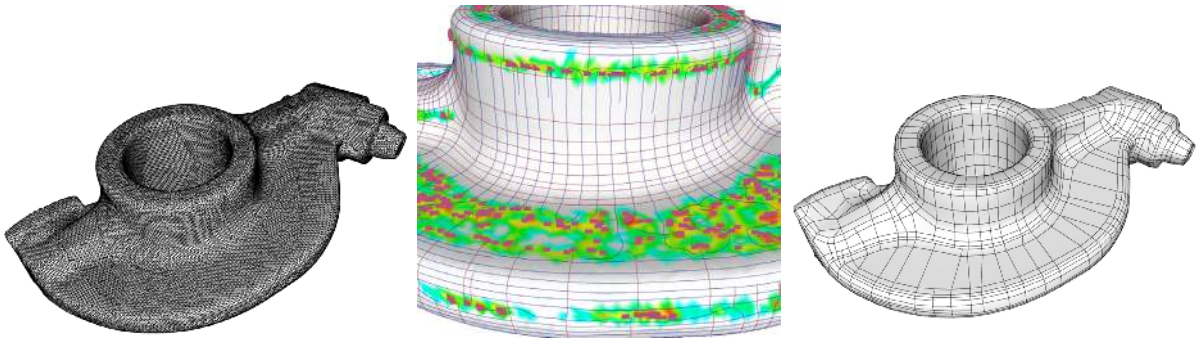


Figure 37: *Quad-dominant remeshing. Left: The input is a manifold triangle mesh. Middle: In regions of low confidence, the curvature lines are not well-defined. The algorithm bridges these regions by extrapolation and produces the result on the right.*

An alternative to streamline integration is the *iso-contouring* of suitable scalar functions. Contours can be seen as coordinate lines, i.e., the construction of such scalar functions is closely related to parameterization (see Section 8) or finding $u-$ and v-coordinates on the surface. Ray et al. [RLL*05] establish periodic functions which are aligned with input vector fields. Tong et al [TACSD06] enable the design of quadrangulations by specifying singularity lines on the mesh and solving a linear system.

10.3. Variational Shape Approximation

Variational shape approximation (VSA) is a relatively new approach to remeshing (and to shape approximation in general) introduced by Cohen-Steiner et al. [CAD04]. VSA is highly sensitive to features and symmetries and produces anisotropic remeshings of high approximation quality. In VSA the input shape is approximated by a set of proxies. The approximation error is iteratively decreased by clustering faces into best-fitting regions. In contrast to the remeshing methods presented in the previous sections, VSA does not require a parameterization of the input or local estimates of differential quantities. Apart from remeshing, VSA techniques can also be used in mesh segmentation.

Let $\mathcal{M}$ be a triangle mesh and let $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_k\}$ be a partition of $\mathcal{M}$ into k regions, i.e., $\mathcal{R}_i \subset \mathcal{M}$ and

$$\mathcal{R}_1 \cup \cdots \cup \mathcal{R}_k = \mathcal{M} \ .$$

Furthermore let $\mathcal{P} = \{P_1, \ldots, P_k\}$ be a set of *proxies*. A proxy $P_i = (\mathbf{x}_i, \mathbf{n}_i)$ is simply a plane in space through the point $\mathbf{x}_i$ with normal direction $\mathbf{n}_i$. Cohen-Steiner et al. consider two metrics that measure a generalized distance of a region $\mathcal{R}_i$ to its proxy P_i. The standard $\mathcal{L}^2$ metric is defined as

$$\mathcal{L}^2(\mathcal{R}_i, P_i) = \int_{\mathbf{x} \in \mathcal{R}_i} ||\mathbf{x} - \pi_i(\mathbf{x})||^2 d\mathbf{x} \ ,$$

where $\pi_i(\mathbf{x}) = \mathbf{x} - \mathbf{n}_i \mathbf{n}_i^T (\mathbf{x} - \mathbf{x}_i)$ is the orthogonal projection of $\mathbf{x}$ onto P_i. They also introduce a new shape metric $\mathcal{L}^{2,1}$ that is based on a measure of the normal field

$$\mathcal{L}^{2,1}(\mathcal{R}_i, P_i) = \int_{\mathbf{x} \in \mathcal{R}_i} ||\mathbf{n}(\mathbf{x}) - \mathbf{n}_i||^2 d\mathbf{x} \ .$$

The goal of variational shape approximation is then the following: Given a number k and an error metric E (i.e., either $E = \mathcal{L}^2$ or $E = \mathcal{L}^{2,1}$) find a set $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_k\}$ of regions and a set $\mathcal{P} = \{P_1, \ldots, P_k\}$ of proxies such that the global distortion

$$E(\mathcal{R}, \mathcal{P}) = \sum_{i=1}^{k} E(\mathcal{R}_i, P_i) \tag{33}$$

is minimized. For remeshing purposes one can then extract a remesh of the original input from the proxies.

In the following we will describe and compare two algorithms for computing an (approximate) minimum of Eq. (33). The first algorithm is due to Cohen-Steiner et al. and uses Lloyd-clustering to produce the regions $\mathcal{R}_i$. The second method is a greedy approximation to VSA with additional injectivity guarantees.

10.3.1. Lloyd Clustering

Cohen-Steiner et al. [CAD04] use a method to minimize Eq. (33) that is inspired by Lloyd's clustering algorithm, which has been used for mesh segmentation in [SWG*03]. The algorithm iteratively alternates between a *geometry partitioning phase* and a *proxy fitting* phase. In the geometry partitioning phase the algorithm computes a set of regions that best fit a given set of proxies. In the proxy fitting phase, the partitioning is kept fixed and the proxies are adjusted.

Geometry partitioning In the geometry partitioning phase, the algorithm modifies the set $\mathcal{R}$ of regions to achieve a lower approximation error Eq. (33) while keeping the proxies $\mathcal{P}$ fixed. It does so by first selecting a number of seed triangles and then greedily growing new regions $\mathcal{R}_i$ around these seeds.

First the algorithm picks the triangle t_i from each region $\mathcal{R}_i$ that is most similar to its associated proxy P_i. This can easily be done by iterating once over all triangles t in $\mathcal{R}_i$ and finding the one that minimizes $E(t,P_i)$.

After initializing $\mathcal{R}_i = \{t_i\}$, the algorithm simultaneously grows the sets $\mathcal{R}_i$. A priority queue contains candidate pairs (t,P_i) of triangles and proxies. The priority of a triangle/proxy pair (t,P_i) is naturally given as $E(t,P_i)$. For each seed triangle t_i its neighboring triangles r are found and the pairs (r,P_i) are inserted into the queue. The algorithm then iteratively removes pairs (t,P_i) from the queue, checks whether t has already been conquered by the region growing process, and if not assigns t to $\mathcal{R}_i$. Again the unconquered neighbor triangles r of t are selected and the pairs (r,P_i) are inserted into the queue. This process is iterated until the queue is empty and all triangles are assigned to a region. Note that a given triangle can appear up to three times simultaneously in the queue. One could of course check for each triangle, whether it already is in the queue and if so take appropriate measures. Instead of this expensive check the algorithm rather keeps a status bit *conquered* for each triangle and checks this bit before assigning a triangle to a region. The following pseudo-code summarizes the geometry partitioning procedure:

partition($\mathcal{R} = \{\mathcal{R}_1,\ldots,\mathcal{R}_k\}, \mathcal{P} = \{P_1,\ldots,P_k\}$)

```
// find the seed triangles and initialize the priority queue
queue = ∅
for i = 1 to k do
   select the triangle t ∈ R_i that minimizes E(t, P_i)
   R_i = {T}
   set t to conquered
   for all neighbors r of t do
      insert (r, P_i) into queue

// grow the regions
while the queue is not empty do
   get next (t, P_i) from the queue
   if t is not conquered then
      set t to conquered
      R_i = R_i ∪ {t}
      for all neighbors r of t do
         if r is not conquered then
            insert (r, P_i) into queue
```

To initialize the algorithm one randomly picks k triangles $t_1,\ldots,t_k$ on the input model, sets $\mathcal{R}_i = \{t_i\}$ and initializes $P_i = (\mathbf{x}_i,\mathbf{n}_i)$ where $\mathbf{x}_i$ is an arbitrary point on t_i and $\mathbf{n}_i$ is t_i's normal. Then regions are grown as in the geometry partitioning phase.

Proxy fitting In the proxy fitting phase, the partition $\mathcal{R}$ is kept fixed while the proxies $P_i = (\mathbf{x}_i,\mathbf{n}_i)$ are adjusted in order to minimize Eq. (33). For the $\mathcal{L}^2$ metric the best proxy is the area weighted least-squares fitting plane. It can be found using standard principal component analysis. When using the $\mathcal{L}^{2,1}$ metric, the proxy normal $\mathbf{n}_i$ is just the area-weighted average of the triangle normals. The base point $\mathbf{x}_i$ is irrelevant for $\mathcal{L}^{2,1}$, but is set to the barycenter of $\mathcal{R}_i$ for remeshing purposes.

Extracting the remesh From an optimal partitioning $\mathcal{R} = \{\mathcal{R}_1,\ldots,\mathcal{R}_k\}$ and corresponding proxies $\mathcal{P} = \{P_1,\ldots,P_k\}$ one can now extract an anisotropic remesh as follows: First, all vertices in the original mesh that are adjacent to three or more different regions are identified. These vertices are projected onto each proxy and their average position is computed. These so-called anchor vertices are then connected by tracing the boundaries of the regions $\mathcal{R}$. The resulting faces are triangulated by performing a "discrete" Delaunay triangulation. Details of this non-trivial meshing scheme are given in Cohen-Steiner's paper.

10.3.2. Greedy Approximation

In [MK05] a greedy algorithm to compute an approximate minimum of Eq. (33) is proposed (see Fig. 38). It's main advantages are:

- The algorithm naturally generates a multi-resolution hierarchy of shape approximations (Fig. 38).
- The output is guaranteed to be free of fold-overs and degenerate faces.

On the downside, due to its greedy approach, it is more likely that the algorithm gets stuck in a local minimum (although this is rarely observed in practice). Furthermore, its implementation is involved and requires the robust computation of Delaunay triangulations.

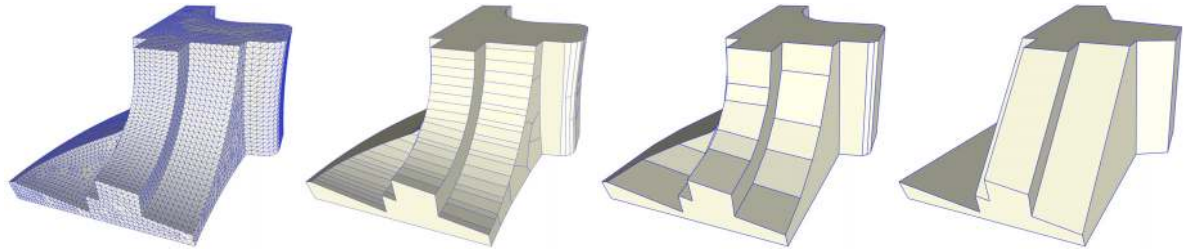


Figure 38: *A multi-resolution hierarchy of differently detailed meshes that was created by variational shape approximation.*

Setup In addition to the partition $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_k\}$ and the proxies $\mathcal{P} = \{P_1, \ldots, P_k\}$, the algorithm maintains a set of polygonal faces $\mathcal{F} = \{f_1, \ldots, f_k\}$. Each face f_i can be an arbitrary connected polygon, i.e., it has an outer boundary and possibly a number of inner boundaries around interior holes. At the beginning of the algorithm we initialize the sets $\mathcal{R}, \mathcal{P}$, and $\mathcal{F}$ as follows:

- $\mathcal{R}_i = \{t_i\}$, i.e., each triangle makes up a region on its own.
- The proxy of $\mathcal{R}_i$ is set to $P_i = (\mathbf{x}_i, \mathbf{n}_i)$ where $\mathbf{x}_i$ is an arbitrary point on t_i and $\mathbf{n}_i$ is t_i's normal.
- $f_i = t_i$, in particular the projection of f_i onto P_i is injective.

Algorithm Invariant The goal of the algorithm is to guarantee a valid shape approximation that is free of fold-overs and degenerate faces. This is achieved by maintaining the following invariant at all times during the run of the algorithm:

> *Injectivity constraint:* The projection of f_i onto P_i is injective.

Note that the initial settings for the sets $\mathcal{R}$, $\mathcal{P}$, and $\mathcal{F}$ satisfy this constraint.

Due to the injectivity constraint, one is able to extract a valid triangle mesh at all times during the run of the algorithm. To produce a triangulation $\mathcal{D}_i$ of a face f_i one simply projects f_i onto P_i (which is a plane), performs a (planar) constrained Delaunay triangulation there, and lifts the triangles of the Delaunay triangulation back to f_i.

Greedy Optimization The partitioning is now greedily optimized in a loop that stops when a predefined maximum error or a predefined number of regions is reached. In each iteration one selects (subject to the injectivity constraint) two regions $\mathcal{R}_i$ and $\mathcal{R}_j$ and merges them into a new region $\mathcal{R}' = \mathcal{R}_i \cup \mathcal{R}_j$. (The order in which the merging is performed is described in the next paragraph.) Then a new proxy $P' = (\mathbf{x}', \mathbf{n}')$ is computed as an area-weighted average of P_i and P_j

$$\mathbf{n}' = \frac{a_i \mathbf{n}_i + a_j \mathbf{n}_j}{||a_i \mathbf{n}_i + a_j \mathbf{n}_j||} \quad \text{and} \quad \mathbf{x}' = \frac{a_i \mathbf{x}_i + a_j \mathbf{x}_j}{a_i + a_j} \ ,$$

where $a_i = area(\mathcal{R}_i)$. Finally, a new face f' is computed by identifying and removing the common boundary edges of f_i and f_j. The algorithm then checks for valence two vertices: If it finds an interior valence two vertex, it is immediately removed. Boundary valence-two vertices are only removed, if their distance from the proxy is smaller than a user-defined threshold.

Note again, that all the operations described above (merging of faces, removal of valence two vertices) are only performed if the injectivity constraint is not violated by the operation!

Merge priorities For each adjacent pair $\mathcal{R}_i$ and $\mathcal{R}_j$ of regions we could compute the shape measure $E(\mathcal{R}', P')$ as described in Eq. (33) and order the region pairs by increasing shape error. In order to speed up the algorithm, the exact $\mathcal{L}^2$ measure is approximated by

$$L^2(f') = \mathcal{L}^2(\mathcal{D}_i, P') + \mathcal{L}^2(\mathcal{D}_j, P') \quad .$$

Since $\mathcal{D}_i$ usually contains much less triangles than $\mathcal{R}_i$ this will significantly speed up the algorithm. The $\mathcal{L}^{2,1}$ error is replaced by

$$L^{2,1}(f') = a_i ||\mathbf{n}_i - \mathbf{n}'||^2 + a_j ||\mathbf{n}_j - \mathbf{n}'||^2 \quad ,$$

where $a_i = area(\mathcal{R}_i)$ as before. The two error measures are combined into a single, scale-independent measure

$$E(f') = \left(1 + L^2(f')\right) \cdot \left(1 + L^{2,1}(f')\right) \quad ,$$

which does not require any user selected weight parameters.

10.3.3. Discussion

Cohen-Steiner's algorithm is fast, efficient, and generally produces high quality results with low approximation error. However, the mesh extraction step might produce degenerate triangles and fold-overs. The extensions presented by Marinov produce a hierarchy of reconstructions which are guaranteed to be free of fold-overs. However, due to the greedy approach, Marinov's algorithm is more likely to get stuck in a local optimum. To achieve acceptable running times, they furthermore have to resort to an approximation of the true $\mathcal{L}^2$ or $\mathcal{L}^{2,1}$ errors.

Generalizations In [WK05] the variational shape approximation approach is taken a step further by allowing for proxies other than simple planes, e.g., spheres, cylinders, and rolling-ball blends. Apart from requiring fewer primitives to achieve a certain reconstruction quality, this method can also recover the "semantic structure" of an input model to some extend, see Fig. 39. In [YLW06] quadrics are used as proxies, and in [JKS05] a similar idea is used to decompose the input mesh into nearly developable segments.

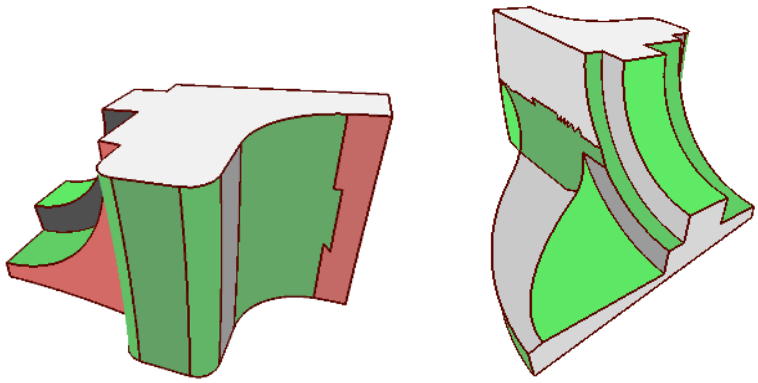


Figure 39: *Hybrid Variational Surface Approximation: In addition to planes, Wu and Kobbelt also use more general proxies like spheres, cylinders, and rolling ball blends. These proxies allow to recover the semantic structure of the input model.*

In [MK06] a quad-dominant remeshing algorithm is proposed that exploits the mesh segmentation $\mathcal{R}$ produced by VSA. First sample points are uniformly distributed on the boundaries of the patches $\mathcal{R}_i$ and each patch is parameterized over a 2D domain. There, each pair of sample points is connected by a cubic curve that minimizes its bending energy. A discrete optimization algorithm selects a subset of the cubics that produces the most well-shaped elements. The resulting quad-dominant mesh is then projected back to 3D. This algorithm is easily able to bridge flat or noisy regions of the input mesh in a robust manner.

10.4. Mesh Segmentation

As stated initially, remeshing has local aspects as well as global ones. Locally, the shape, orientation, and alignment of individual element matters while globally, the overall geometric structure of the object has to be represented by the orientation

and alignment of entire regular sub-patches [MK06]. To address the local aspects, harmonic parameterizations and curvature analysis are the appropriate mathematical tools. To recover the global structure, segmentation techniques are required. Since a detailed description of the various approaches to mesh segmentation are beyond the scope of these notes, we only briefly sketch the fundamental techniques that are typically used.

- *Lloyd-relaxation:* Variational shape approximation and all variants and extensions can be considered as segmentation techniques as well.
- *Region growing / level sets:* By omitting the proxy re-fitting, variational shape approximation degenerates into a classical region growing technique. By using more sophisticated growing strategies that also take surface curvature into account, these techniques can be interpreted as level sets which live as scalar fields on a manifold.
- *Snakes:* If an initial segmentation is given manually or through some pre-processing, it can be refined by a snake-based approach where the initial solution is improved by minimizing an energy functional, which accounts for local fitting, e.g., to a curvature maximum, and segmentation boundary smoothness. Snakes effectively prevent jagged segment boundaries.
- *Graph-cut:* Level sets and snakes can only guarantee convergence to a local optimum. If a surface region for the boundary between two segments can be identified, the global optimum can be found by re-formulating the segmentation problem as a minimum graph-cut problem. In [KT03] the graph-cut is computed for the dual graph of the triangle mesh with edge-weights depending on the local surface curvature.

11. Shape Deformations

In this section we will describe and compare different kinds of surface deformation techniques. We do not discuss approaches for surface design from scratch, like for instance [AWC04, ACWK04], but rather focus on deforming given surfaces in a controlled manner. We will first discuss freeform deformations, either surface-based (Section 11.1) or space deformations (Section 11.2), which allow to deform a given surface in a smooth manner. However, under global deformations these techniques do not deform fine surface details in a natural manner. Intuitive detail handling is provided by multiresolution hierarchies (Section 11.3), which can enhance any freeform deformation technique, or by deformations based on differential coordinates (Section 11.4).

11.1. Surface-Based Freeform Deformations

For surface-based freeform deformations we are looking for a displacement function $\mathbf{d} : \mathcal{S} \rightarrow \mathbf{R}^3$ which maps the given surface $\mathcal{S}$ to its deformed version $\mathcal{S}'$:

$$\mathcal{S}' := \{\mathbf{p} + \mathbf{d}(\mathbf{p}) \mid \mathbf{p} \in \mathcal{S}\} \quad .$$

Especially in engineering applications it is of major importance to be able to exactly control the deformation process, i.e., to specify displacements for a set of constrained points $\mathcal{C}$:

$$\mathbf{d}(\mathbf{p}_i) = \mathbf{d}_i \,, \quad \forall \mathbf{p}_i \in \mathcal{C} \quad .$$

Since we are targetting interactive shape deformations, another important aspect is the amount of user interaction or guidance required to specify a desired deformation function $\mathbf{d}$.

11.1.1. Tensor-Product Spline Surfaces

The traditional surface representation for CAGD are spline surfaces that are controlled by the intuitive control point metaphor and provide high quality smooth surfaces. A single tensor-product spline patch is defined as

$$\mathbf{f}(u,v) \;=\; \sum_{i=0}^{m} \sum_{j=0}^{m} \mathbf{c}_{ij} N_i^n(u) N_j^n(v) \quad ,$$

i.e., each control point $\mathbf{c}_{ij}$ is associated with a smooth basis function $N_{ij}(u,v) := N_i^n(u) N_j^n(v)$. As a consequence, a translation of a control point adds a smooth bump of rectangular support to the surface (cf. Fig. 40, left). Every more sophisticated modeling operation has to be composed from these smooth elementary modifications, such that the displacement function has the form

$$\mathbf{d}(u,v) \;=\; \sum_{i=0}^{m} \sum_{j=0}^{m} \delta\mathbf{c}_{ij} N_{ij}(u,v) \quad ,$$

where $\delta\mathbf{c}_{ij}$ denotes the change of control point $\mathbf{c}_{ij}$. As a consequence, the support of the deformation is the union of the supports of individual basis functions. 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 Fig. 40.

It was also shown in Section 2.1.1 that tensor-product spline surfaces are restricted to rectangular domains, and that complex surfaces therefore have to be composed by a large number of spline patches. Specifying complex deformations in terms of control point movements thus involves a lot of user interaction, since smoothness constraints across patch boundaries have to be considered during the whole deformation process. Also notice that prescribing constraints $\mathbf{d}(u_i, v_i) = \mathbf{d}_i$ requires to solve a linear system for the control point displacements $\delta\mathbf{c}_{ij}$. These system can be over- as well as under-determined, and hence are typically solved by least squares and least norm techniques. However, in the first case, the system cannot be solved exactly, and in the latter case the minimization of control point displacements does not neccessarily lead to fair deformations, which would require to minimize some fairness energy (Section 7.4).

11.1.2. Transformation Propagation

The main drawback of spline-based deformations is that the underlying mathematical surface representation is identical to the basis functions that are used for the surface deformation. To overcome this limitation, the deformation basis functions consequently should be independent of the actual surface representation.

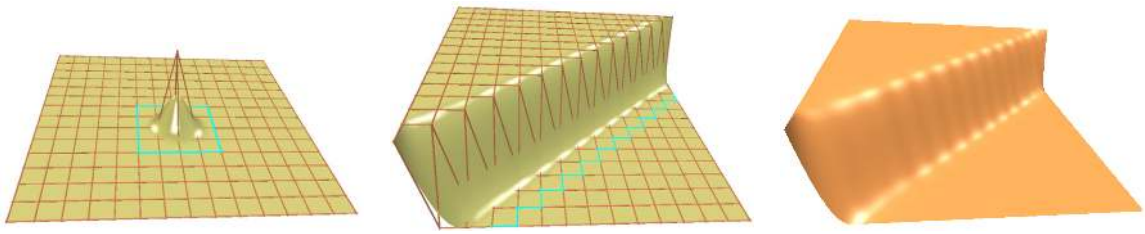


Figure 40: *A modeling example using a bi-cubic tensor-product spline surface. Each control point is associated with a smooth basis function of fixed rectangular support (*left*). This fixed support and the fixed regular placement of the control points, resp. basis functions, prevents a precise support specification (*center*) and can lead to alias artifacts in the resulting surface, that are revealed by more sensitive surface shading (*right*).*

A popular approach falling into this category works as follows (cf. Fig. 41): in a first step the user specifies the support of the deformation (the region which is allowed to change) and a handle region within it. A direct transformation applied to this handle region is then to be smoothly interpolated within the support region in order to blend between the transformed handle and the fixed part of the surface.

This smooth blend is controlled by a scalar field $s : \mathcal{S} \rightarrow [0,1]$, which $s(\mathbf{p})$ is 1 at the handle (full deformation), 0 outside the support (no deformation), and blends between 1 and 0 within the support region. One way to construct the scalar field is to compute geodesic distances $\text{dist}_0(\mathbf{p})$ and $\text{dist}_1(\mathbf{p})$ from $\mathbf{p}$ to the fixed part and the handle region, respectively, and to define

$$s(\mathbf{p}) = \frac{\text{dist}_0(\mathbf{p})}{\text{dist}_0(\mathbf{p}) + \text{dist}_1(\mathbf{p})} ,$$

similar to [BK03a, PKKG03]. However, in case of concave handle regions the geodesic distance fields will not be smooth. Instead, harmonic fields can be used, which provide guaranteed smoothness [ZRKS05]. In this case a linear Laplacian system has to be solved for the values of $s(\mathbf{p}_i)$ at the free vertices $\mathbf{p}_i$ within the support region:

$$\begin{aligned} \Delta_{\mathcal{S}}\, s(\mathbf{p}_i) &= 0 \quad \mathbf{p}_i \in \text{support} \\ s(\mathbf{p}_i) &= 1 \quad \mathbf{p}_i \in \text{handle} \\ s(\mathbf{p}_i) &= 0 \quad \mathbf{p}_i \in \text{fixed} \ . \end{aligned}$$

As an additional benefit the scalar field $s(\mathbf{p})$ can further be enhanced by a transfer function $t(s(\mathbf{p}))$, which provides more control of the blending process [BK03a, PKKG03]. The damping of the handle transformation is then performed separately on the rotation, scale/shear, and translation components, for instance like in [PKKG03]. In case the individual transformation components are not given, they can be computed by polar decomposition [SD92].

Figure 41: *After specifying the blue support region and the green handle regions (*left*), a smooth scalar field is constructed that is 1 at the handle and 0 outside the support (*center*). This scalar field is used to propagate and damp the handle's transformation (*right*).*

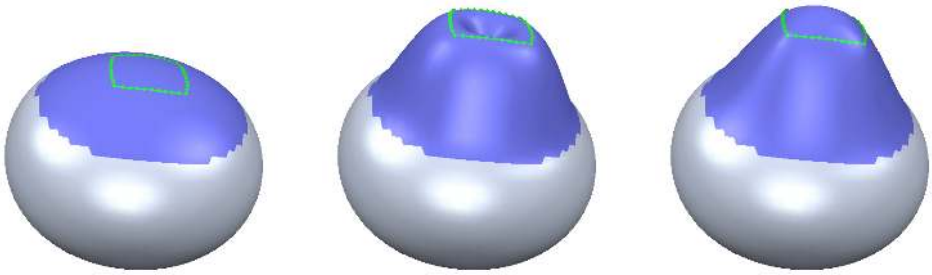


Figure 42: *A sphere is deformed by lifting a closed handle polygon (*left*). Propagating this translation based on geodesic distance causes a dent in the interior of the handle polygon (*center*). The more intuitive solution of a smooth interpolation (*right*) cannot be achieved with this approach; it was produced by boundary constraint modeling.*

As shown in Fig. 42, the major problem with this approach is that distance-based propagations cannot yield the geometrically most intuitive solution, which would be a smooth interpolation of the (transformed) handle region by a high quality smooth surface.

11.1.3. Boundary Constraint Modeling

Smooth surface deformation functions $\mathbf{d}(\cdot)$ with prescribed boundary constraints $\mathbf{d}(\mathbf{p}_i) = \mathbf{d}_i$ are most elegantly modeled by an energy minimization principle [MS92, WW92, KCVS98, BK04a]. 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. A popular example is the thin-plate energy of the displacement function $\mathbf{d}$ (Section 7):

$$E_{TP}(\mathbf{d}) = \int_{\mathcal{S}} \|\mathbf{d}_{uu}(\mathbf{x})\|^2 + 2\|\mathbf{d}_{uv}(\mathbf{x})\|^2 + \|\mathbf{d}_{vv}(\mathbf{x})\|^2 \, d\mathbf{x} \quad .$$

The optimal surface is the one that minimizes this energy while satisfying the prescribed boundary conditions. These constraints are given by the first two rings of fixed vertices and handle vertices, which define C^0 and C^1 constraints at the boundary of the support region (cf. Fig. 43, left). In order to efficiently compute the solution of this optimization problem, variational calculus is applied to derive the Euler-Lagrange PDE that characterizes the minimizer of E_{TP} [Kob97]:

$$\begin{array}{rcll} \Delta^2 \mathbf{d}(\mathbf{p}_i) & = & 0 & \mathbf{p}_i \in \text{support} \\ \mathbf{d}(\mathbf{p}_i) & = & \mathbf{d}_i & \mathbf{p}_i \in \text{handle} \\ \mathbf{d}(\mathbf{p}_i) & = & \mathbf{0} & \mathbf{p}_i \in \text{fixed} \end{array} \quad . \tag{34}$$

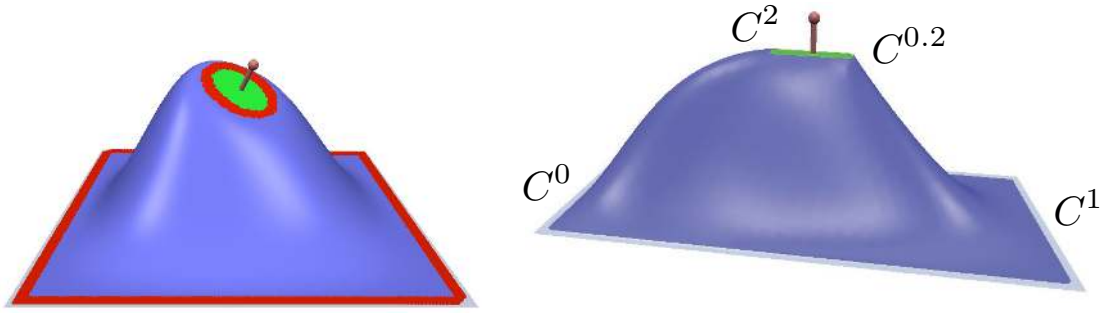


Figure 43: *Minimizing the thin-plate energy allows for C^1 boundary constraints, which are given by the first two rings of fixed and handle vertices (*left*). The framework even allows for anisotropic bending behavior and per-vertex control of boundary continuity (*right*).*

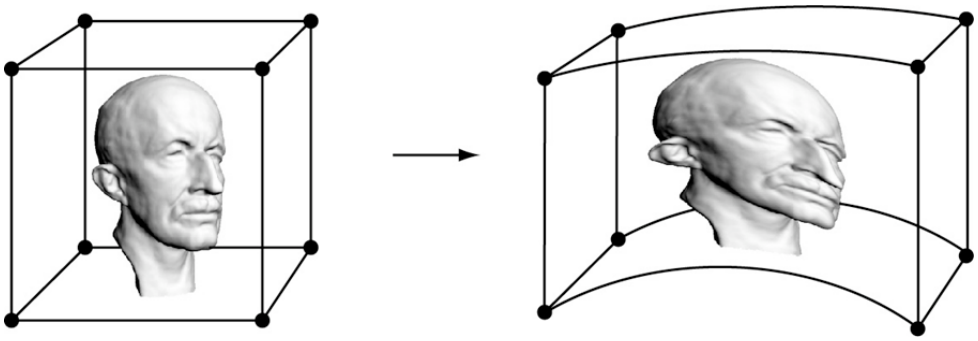


Figure 44: *Freeform space deformations warp the space around an object, and by this deform the embedded object itself.*

Hence, the optimal deformation function $\mathbf{d}$ can directly be computed by solving a bi-Laplacian system. The advantage of this formulation is that it allows to take arbitrary constraints $\mathbf{d}(\mathbf{p}_i) = \mathbf{d}_i$ into account and that the optimal solution is known to have certain smoothness properties.

Interactively transforming the handle region changes the boundary constraints of the optimization. Since this also changes the right-hand side of Eq. (34), this system has to be solved each frame. In Section 12 we therefore discuss efficient linear system solvers which are particularly suited for this multiple right-hand side problem. Also notice that restricting to affine transformation of the handle region (which is usually sufficient) allows to precompute basis functions of the deformation, such that instead of solving the linear system each frame, only these basis functions have to be evaluated [BK04a]. Also notice that the boundary constraint modeling approach is much more flexible. As also shown in [BK04a], different energy functions can be chosen to control the stiffness and bending behavior, which can even be anisotropic. Moreover, the continuity at the support's boundary can be controlled per vertex (cf. Fig. 43, right).

11.2. Space Deformations

The surface-based approaches described in Section 11.1 compute a smooth deformation field *on* the surface $\mathcal{S}$. If the underlying surface representation is a triangle mesh, computing the deformation field typically requires to solve a linear Laplacian system on $\mathcal{S}$. An apparent drawback of such methods is that their computational effort and numerical robustness are strongly related to the complexity and quality of the surface tessellation.

In the presence of degenerate triangles the discrete Laplacian operator is not well-defined and thus the involved linear systems become singular. Similarly, topological problems like gaps or non-manifold configurations lead to problems as well. In such cases quite some effort has to be spent to still be able to compute smooth deformations for the numerically problematic meshes, like eliminating degenerate triangles (Section 4) or even remeshing the complete surface (Section 10). Even when the mesh quality is sufficiently high, extremely complex meshes will result in linear systems which cannot be solved simply due to their size.

The above problems are avoided by volumetric *space deformation* techniques, that deform the whole 3D space and by this implicitly deform the embedded object (cf. Fig. 44). In contrast to surface-based methods, they use a tri-variate deformation function $\mathbf{d} : \mathbb{R}^3 \to \mathbb{R}^3$ to transform all points of the original surface $\mathcal{S}$ to the modified surface $\mathcal{S}' = \{\mathbf{p} + \mathbf{d}(\mathbf{p}) \mid \mathbf{p} \in \mathcal{S}\}$. Satisfying displacement constraints $\mathbf{d}(\mathbf{p}_i) = \mathbf{d}_i$ now amounts to finding a volumetric function $\mathbf{d} : \mathbb{R}^3 \to \mathbb{R}^3$ interpolating them, instead of an interpolation on a manifold $\mathcal{S}$ as for surface-based approaches. Analogously to surface-based techniques, we will see that approaches based on a global energy minimization typically lead to highest quality results.

Since the space deformation function $\mathbf{d}$ does not depend on a particular surface representation, this uniform deformation framework can also be applied to all explicit surface representations, e.g., by transforming all vertices of a triangle mesh or all points of a point-sampled model. However, it does not allow for as fine grained surface control (anisotropic bending, boundary continuity) as the surface-based approaches.

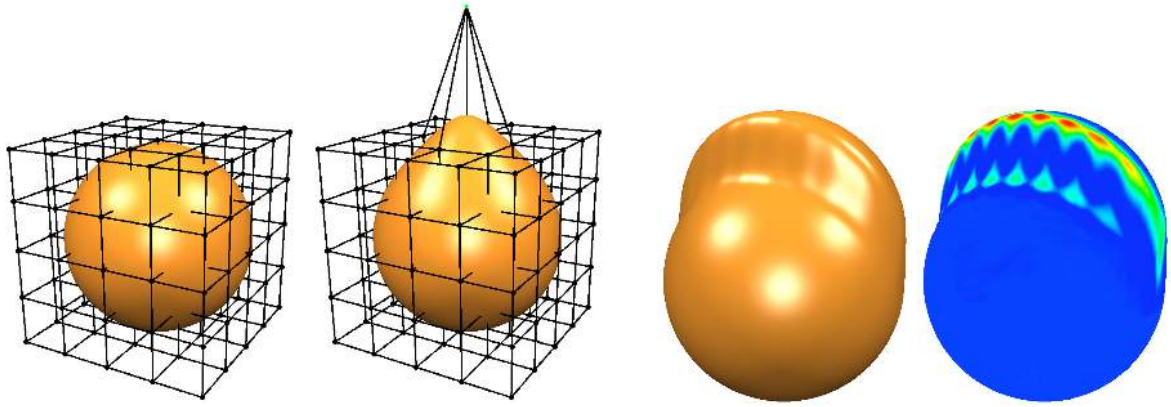


Figure 45: *In the freeform deformation approach a regular 3D control lattice is used to specify a volumetric displacement function (*left*). Similar to tensor-product spline surfaces, the tri-variate tensor-product splines can also lead to alias artifacts in the deformed surface (*right*).*

11.2.1. Freeform Deformation

The classical freeform deformation (FFD) method [SP86] represents the space deformation by a tensor-product Bezier or spline function

$$\mathbf{d}(u,v,w) \;=\; \sum_i \sum_j \sum_k \delta\mathbf{c}_{ijk} N_i^l(u) N_j^n(v) N_k^m(w) \quad .$$

Because of the same reasons as for spline surfaces (Section 11.1.1), these approaches require complex user-interactions and can cause aliasing problems, as shown in Fig. 45. In order to satisfy given displacement constraints, the inverse FFD method [HHK92] solves a linear system for the required movement of grid points $\mathbf{c}_{ijk}$. This system may be over- as well as under-determined and hence is solved by least-squares or least-norm methods, respectively. However, the first cannot exactly interpolate constraints, and the latter minimizes control point movements, which does not necessarily imply a fair deformation of low curvature energy.

11.2.2. Transformation Propagation

Handle transformations can be propagated analogously to the surface-based techniques described in Section 11.1.2 by constructing the scalar field $s(\cdot)$ based on Euclidean distances, instead of geodesic distances [PKKG03]. While this typically leads to inferior results compared to geodesic-based propagation, this method even works if a surface-based propagation fails due to topological problems like gaps or holes, or even is a point-sampled model

Besides from that, the limitations of the surface-based propagation also apply to this method. A smooth interpolation of arbitrary constraints might not be possible, and the resulting surface fairness is typically inferior to techniques based on energy minimization.

11.2.3. RBF Boundary Constraint Modeling

Surface-based boundary constraint modeling employs a deformation function $\mathbf{d} : \mathcal{S} \to \mathbb{R}^3$ which smoothly interpolates user-defined displacement constraints $\mathbf{d}(\mathbf{p}_i) = \mathbf{d}_i$. An optimally fair function is found by minimizing an energy functional.

For space deformations, we therefore want to find a smoothly interpolating tri-variate deformation function $\mathbf{d} : \mathbb{R}^3 \to \mathbb{R}^3$ which minimizes some analogous energies. Radial basis functions (RBFs) are known to be well suited for such kinds of scattered data interpolation problems. A tri-variate RBF deformation is defined in terms of centers $\mathbf{c}_j \in \mathbb{R}^3$ and weights $\mathbf{w}_j \in \mathbb{R}^3$ as

$$\mathbf{d}(\mathbf{x}) \;=\; \sum_j \mathbf{w}_j \; \varphi\left(\left\|\mathbf{c}_j - \mathbf{x}\right\|\right) + \mathbf{p}(\mathbf{x}) \quad , \tag{35}$$

where $\varphi\left(\left\|\mathbf{c}_j - \cdot\right\|\right)$ is the basis function corresponding to the jth center $\mathbf{c}_j$ and $\mathbf{p}(\mathbf{x})$ is a polynomial of low degree used to guarantee polynomial precision.

Figure 46: *Using multiple independent handle components allows to stretch the hood while rigidly preserving the shape of the wheel houses. This 3M triangle model consists of 10k individual connected components, which are neither two-manifold nor consistently oriented.*

The choice of φ has a strong influence on the computational complexity and the resulting surface's fairness: while compactly supported radial basis functions lead to sparse linear systems and hence can be used to interpolate several hundred thousands of data points [MYC*01, OBS04], they do not provide the same degree of fairness as basis functions of global support [CBC*01]. It was shown by Duchon [Duc77] that for the basis function $\varphi(r) = r^3$ and quadratic polynomials $\mathbf{p}(\cdot) \in \Pi_2$, the function (35) is triharmonic ($\Delta^3 \mathbf{d} = 0$) and hence minimizes the energy

$$\int_{\mathbb{R}^3} \|\mathbf{d}_{xxx}(\mathbf{x})\|^2 + \|\mathbf{d}_{xxy}(\mathbf{x})\|^2 + \ldots + \|\mathbf{d}_{zzz}(\mathbf{x})\|^2 \, d\mathbf{x} \quad .$$

Notice that these trivariate functions are conceptually equivalent to the minimum variation surfaces of [MS92] and the triharmonic surfaces used in [BK04a], and hence provide the same degree of fairness.

In order to construct an RBF interpolating the constraints $\mathbf{d}(\mathbf{p}_i) = \mathbf{d}_i$, the centers are placed on the constraints ($\mathbf{c_i} = \mathbf{p}_i$) and a linear system is solved to find the RBF's weights $\mathbf{w}_i$ and the coefficients of the quadratic polynomial $\mathbf{p}(\mathbf{x})$ [BK05b]. Due to the global support of the triharmonic basis function $\varphi(r) = r^3$ this linear system is dense, which implies cubic complexity for standard solvers.

However, Botsch and Kobbelt [BK05b] propose an incremental least squares method that efficiently solves the linear system up to a prescribed error bound. Using this solver to pre-compute deformation basis functions allows to interactively deform even complex models. Moreover, evaluating these basis functions on the graphics card further accelerates this approach and provides real-time space deformations at a rate of 30M vertices/sec. As shown in Fig. 46, even complex surfaces consisting of disconnected patches can be handled by this technique, whereas all surface-based techniques would fail in this situation.

11.3. Multiresolution Deformations

The previous section introduced surface-based and spatial freeform deformations, which provide smooth deformations of given surfaces. However, as shown in Fig. 47, these approaches typically do not correctly handle fine-scale surface details. Preservating these details under global deformations is provided by multiresolution techniques, which are described in the following.

Multiresolution or multi-scale techniques perform a frequency decomposition of the object in order to provide global deformations with intuitive detail preservation. Section 7 described that signal processing techniques, such as low-pass filtering, 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 one now wants to explicitly modify the low frequencies and preserve the high frequency details, resulting in the desired multiresolution deformation. Fig. 48 shows a simple 2D example of this concept.

The complete multiresolution editing process is depicted in Fig. 49. 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: $\mathcal{S} = \mathcal{B} \oplus \mathcal{D}$.

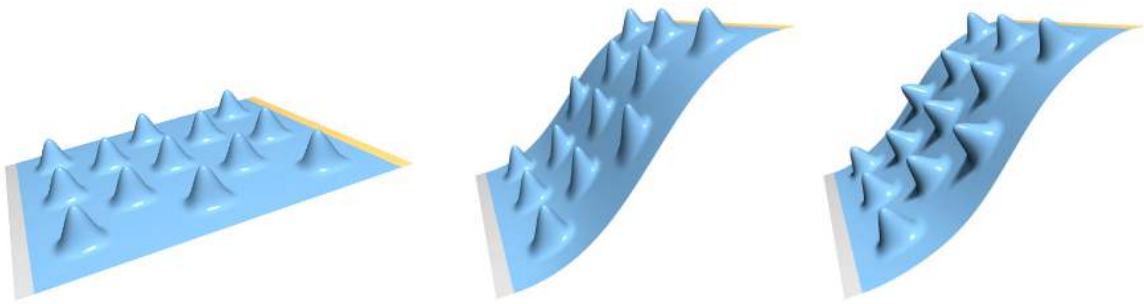


Figure 47: *Fine-scale surface details are distorted by freeform deformations due to the lack of local frame rotations. For this pure translation, all surface points move in the same direction. In contrast, multiresolution deformations correctly preserve the bumps under global deformations.*

The special operators $\ominus$ and $\oplus$ are called the *decomposition* and the *reconstruction* operator of the multiresolution framework, respectively. 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 deformation $\mathcal{S}' = \mathcal{B}' \oplus \mathcal{D}$.

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. 49.

A complete multiresolution deformation framework therefore has to provide the three basic operators shown in Fig. 49: the decomposition operator (*detail analysis*), the freeform editing operator (*shape deformation*), and the reconstruction operator (*detail synthesis*). The decomposition is typically performed by mesh smoothing (Section 7), and freeform deformations have been discussed in Section 11.1 and Section 11.2. The missing component is a suitable representation for the geometric detail $\mathcal{D} = \mathcal{S} \ominus \mathcal{B}$, for which we describe displacement vectors and displacement volumes in the following.

11.3.1. 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 $\mathbf{h} : \mathcal{B} \to \mathbb{R}^3$, that associates a displacement vector $\mathbf{h}(\mathbf{b})$ 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 $\mathcal{S} = \{\mathbf{b} + \mathbf{h}(\mathbf{b}) \mid \mathbf{b} \in \mathcal{B}\}$.

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. 50, 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. 50, 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 with position $\mathbf{p}_i \in \mathcal{S}$ has an associated base point $\mathbf{b}_i \in \mathcal{B}$. The corresponding displacement vector $\mathbf{h}_i := (\mathbf{p}_i - \mathbf{b}_i)$ is then stored in the local frame of $\mathcal{B}$ at the point $\mathbf{b}_i$ [ZSS97, GSS99]. The problem of these general displacement vectors is that the tangential component leads to ambiguities in the local frame encoding and might also cause non-intuitive detail 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 $\mathcal{S}$ might be 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

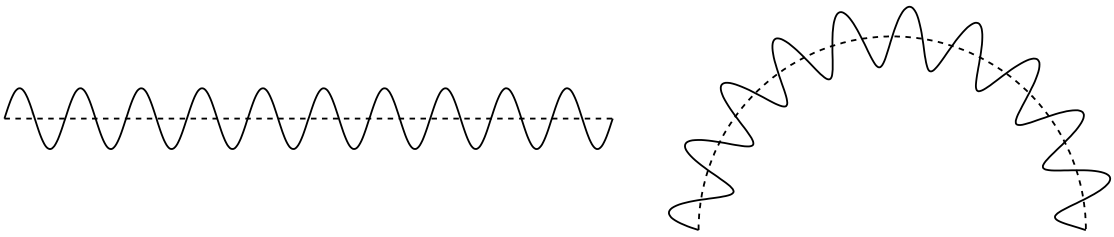


Figure 48: *A multiresolution deformation of a sine wave. A frequency decomposition yields the dashed line as its low frequency component (*left*). Bending this line and adding the higher frequencies back onto it results in the desired global shape deformation (*right*).*

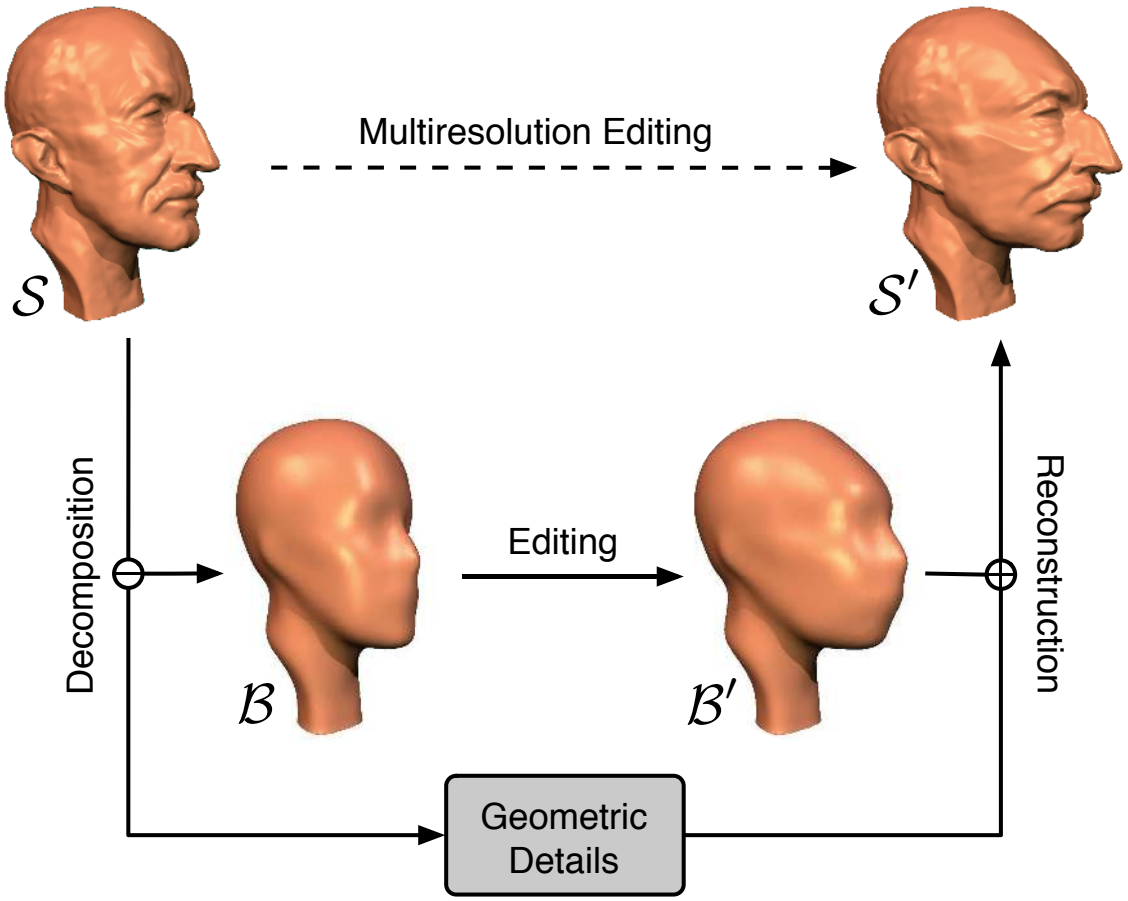


Figure 49: *A general multiresolution editing framework consists of three main operators: the* decomposition *operator, that separates the low and high frequencies, the* editing *operator, that deforms the low frequency components, and the* reconstruction *operator, that adds the details back onto the 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.*

$\mathbf{b}_i \in \mathcal{B}$ (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.

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.

11.3.2. 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. 51).

A more severe problem of uncoupled displacement vectors is that they do not provide any mechanism to prevent local self-intersections. As shown in Fig. 52, these difficulties typically arise when the base surface is deformed in a concave manner. Where a local self-intersection occurs, the surface 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.

Both problems, the unnatural change of volume and local self-intersections, are addressed by displacement *volumes* instead of displacement *vectors* [BK03b]. 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 but incompressible materials. The multiresolution model will deform like a soft but incompressible layer attached to a rigid skeleton (cf. Fig. 51). Displacement volumes can also effectively avoid local self-intersections (where the surface of a prism would inter-penetrate itself), since prisms can shear, i.e., their top triangles can move tangentially, without changing their volume (cf. Fig. 52).

Notice, however, that the improved detail preservation comes at the considerably higher computational cost of a non-linear detail reconstruction process. Hence, displacement vectors should be used during interactive deformation, the result of which can afterwards be optimized based on displacement volumes.

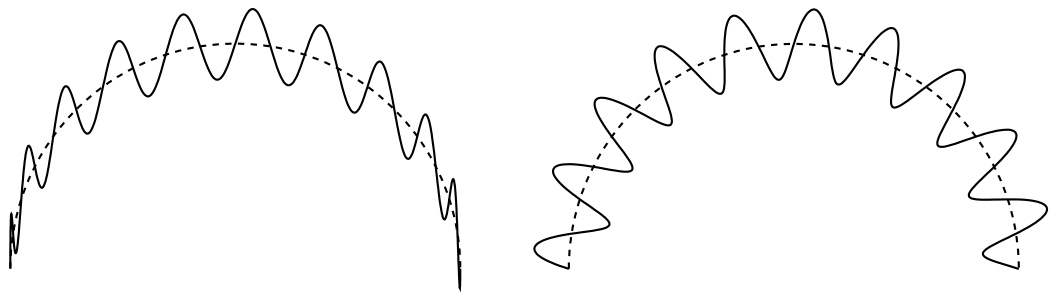


Figure 50: *Representing the displacements w.r.t. the global coordinate system does not lead to the desired result (*left*). The geometrically intuitive solution is achieved by storing the detail w.r.t. local frames which rotate according to the local tangent plane's rotation of $\mathcal{B}$ (*right*).*

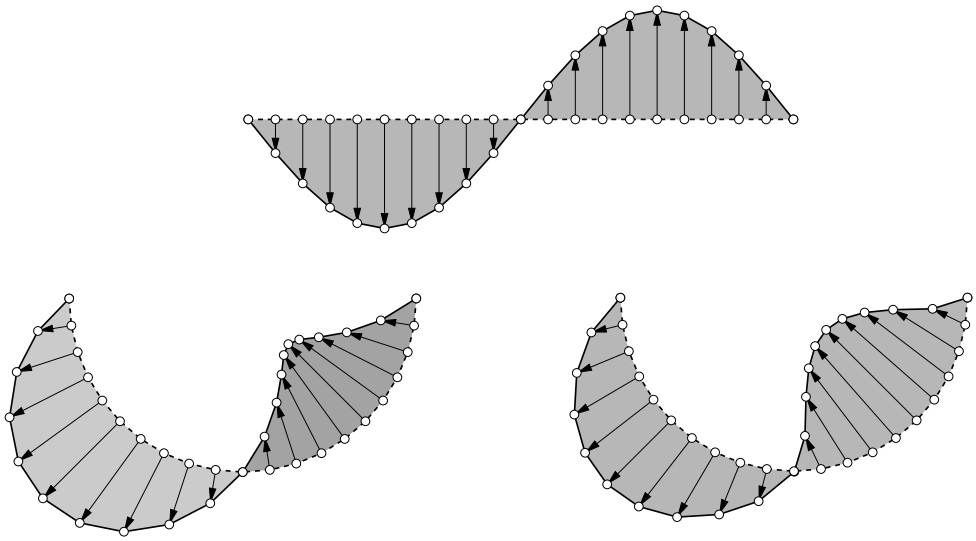


Figure 51: *A multiresolution deformation of a sine wave is done by bending its base line (*dashed*) and reconstructing the corresponding detailed surface (*solid*). Since displacement* vectors *are handled individually, the resulting surface shows an unnatural change of the volume enclosed between base and detailed surface (*bottom left*). Displacement* volumes *provide a natural coupling of the displacements, that prevents local self-intersections (*bottom right*).*

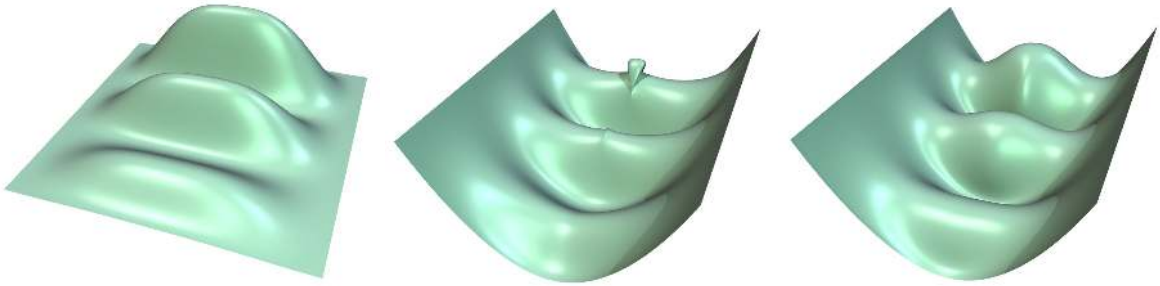


Figure 52: *Detail reconstruction based on displacement* vectors *may lead to a non-plausible change of volume and even to local self-intersections for concave modifications (*center*). Displacement* volumes *instead reconstruct a more natural, non-intersecting surface (*right*).*

11.4. Deformations Based on Differential Coordinates

While multiresolution or multi-scale hierarchies are an effective tool for enhancing freeform deformations by fine-scale detail preservation, the hierarchy generation can become quite involved for geometrically or topologically complex models. To avoid the explicit multi-scale decomposition, another class of methods modifies differential properties of the surface instead of its spatial coordinates, and then reconstructs a deformed surface having the desired differential coordinates.

From this class, the methods of [YZX*04, ZRKS05] deform a surface in terms of its gradient field. For a piecewise linear function $s : \mathcal{S} \to \mathbb{R}$, defined by its values $s_i := s(\mathbf{p}_i)$ at the mesh vertices, the gradient $\nabla s : \mathcal{S} \to \mathbb{R}^3$ is a constant 3-vector $\mathbf{g}_j$ within each triangle f_j. If instead of a scalar function s the piecewise linear coordinate function $\mathbf{p}(v_i) = \mathbf{p}_i \in \mathbb{R}^3$ is considered, then the gradient within a face f_j is a constant 3×3 matrix

$$\nabla \mathbf{p}|_{f_j} =: \mathbf{G}_j \in \mathbb{R}^{3\times 3} \quad .$$

For a mesh with V vertices and F triangles, the gradient operator can be expressed by a $3F \times V$ matrix $\mathbf{G}$:

$$\begin{pmatrix} \mathbf{G}_1 \\ \vdots \\ \mathbf{G}_F \end{pmatrix} = \mathbf{G} \cdot \begin{pmatrix} \mathbf{p}_1^T \\ \vdots \\ \mathbf{p}_V^T \end{pmatrix} .$$

These gradients are then modified explicitly, yielding new gradients $\mathbf{G}'_j$ per triangle f_j. Reconstructing a mesh having these desired gradients is an overdetermined problem, and therefore is solved in a weighted least squares sense using the normal equations [GL89b]

$$\mathbf{G}^T\mathbf{D}\mathbf{G} \cdot \begin{pmatrix} \mathbf{p}_1'^T \\ \vdots \\ \mathbf{p}_V'^T \end{pmatrix} = \mathbf{G}^T\mathbf{D} \cdot \begin{pmatrix} \mathbf{G}'_1 \\ \vdots \\ \mathbf{G}'_F \end{pmatrix} ,$$

where $\mathbf{D}$ is a diagonal matrix containing the face areas as weighting factors. Since the matrix $\mathbf{G}^T\mathbf{D}$ corresponds to the divergence operator, and since $\operatorname{div} \nabla = \Delta$, this system actually is a Poisson equation

$$\Delta \cdot \begin{pmatrix} \mathbf{p}_1'^T \\ \vdots \\ \mathbf{p}_V'^T \end{pmatrix} = \operatorname{div} \cdot \begin{pmatrix} \mathbf{G}'_1 \\ \vdots \\ \mathbf{G}'_F \end{pmatrix} .$$

Hence, these methods prescribe a guidance gradient field $(\mathbf{G}'_1, \ldots, \mathbf{G}'_F)$, compute its divergence, and solve a Poisson system for the modified mesh vertices.

The missing component is a technique for modifying the gradients $\mathbf{G}_j$. For this [YZX*04, ZRKS05] use gradients of affine deformations, i.e., their rotation and scale/shear components, for transforming the surface gradients $\mathbf{G}_j$, which is similar to [SP04]. As a consequence, these methods work well for rotations, but are insensitive to translations: adding a translation to a given deformation does not change its gradient, and thus has no influence on the resulting surface gradients. But as there is a (non-linear) connection between translations and local rotations of gradients, these methods yield counter-intuitive results for modifications containing large translations. Although [YZX*04] proposed a special treatment of pure translations, deformations containing both rotations as well as translations remain problematic.

Other methods directly deform the Laplacians of the vertices instead of gradient fields [LSC*04]. They compute initial Laplacian coordinates $\delta_i = \Delta(\mathbf{p}_i)$ and deform them to $\delta'_i := \mathbf{T}_i\delta_i$ using transforms $\mathbf{T}_i$. Since the Laplacian — as the mean curvature normal — consists of gradient information (its direction) and curvature information (its length), it would theoretically require the Hessian of the deformation in addition to its gradient to update the Laplacian coordinates. For instance, while it is straightforward to bend a cylinder by properly rotating the gradients of triangles, the same is very difficult by direct manipulation of Laplacian coordinates, since their target lengths, i.e., the curvature of the resulting surface, would have to be known beforehand.

Because of that, Sorkine et al. [SCL*04] implicitly solve for the local rotations of vertex neighborhoods as well by minimizing the following energy functional

$$E(\mathbf{p}'_1, \ldots, \mathbf{p}'_V) = \sum_{i=1}^{V} \left\| \mathbf{T}_i\delta_i - \Delta(\mathbf{p}'_i) \right\|^2 + \sum_{i \in \mathcal{C}} \left\| \mathbf{p}'_i - \mathbf{u}_i \right\|^2 ,$$

where $\mathbf{u}_i$ are the target positions for the constrained vertices $\mathbf{p}_i$, $i \in \mathcal{C}$. However, for the sake of computational efficiency they linearize the local frame transformations $\mathbf{T}_i$, which on the one hand allows to formulate the optimization as a single linear system, but one the other hand also leads to artifacts in case of large rotations, as shown in [LSLC05].

In [LSLC05], Lipman et al. minimize surface bending by preserving the relative orientations of per-vertex local frames. This is done by first solving a linear least squares system for the modified per-vertex local frame orientations, and from those reconstruct the modified vertex positions in a second step. However, since the first system does not consider the positional constraints, one has to ensure that the positional constraints and the orientation constraints are compatible. While their method works very well even for large rotations, it exhibits the same translation-insensitivity as the gradient-based methods.

12. Numerics

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, since exploiting their special structure allows for the most efficient and most robust implementations. Examples of such matrices are Laplacian system (to be analyzed in Section 12.1) and general least squares systems. However, the general case of a non-symmetric indefinite system is outlined afterwards in Section 12.6.

Following [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.

For the following discussion we restrict ourselves to sparse spd problems $\mathbf{Ax} = \mathbf{b}$, with $\mathbf{A} = \mathbf{A}^T \in \mathbb{R}^{n\times n}$, $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$, and denote by $\mathbf{x}^*$ the exact solution $\mathbf{A}^{-1}\mathbf{b}$. The general case of non-symmetric indefinite systems is then outlined in Section 12.6.

12.1. Laplacian Systems

Since Laplacian systems play a major role in several geometry processing applications, like smoothing (Section 7), conformal parametrization (Section 8), and shape deformation (Section 11), we will shortly describe general Laplacian matrices first.

In each row the matrix $\Delta_{\mathcal{S}}$ contains the weights for the discretization of the Laplace-Beltrami of a function $f : \mathcal{S} \to \mathbb{R}$ at one vertex v_i (see Section 5):

$$\Delta_{\mathcal{S}} f(v_i) \;=\; \frac{2}{A(v_i)} \sum_{v_j \in \mathcal{N}_1(v_i)} \left(\cot\alpha_{ij} + \cot\beta_{ij}\right)\left(f(v_j) - f(v_i)\right) \quad .$$

This can be written in matrix notation as

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

where $\mathbf{D}$ is a diagonal matrix of normalization factors $\mathbf{D}_{ii} = 2/A(v_i)$, and $\mathbf{M}$ is a symmetric matrix containing the cotangent weights. Since the Laplacian of a vertex v_i is defined *locally* in terms of its one-ring neighbors, the matrix $\mathbf{M}$ 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 $\mathcal{N}_1(v_i)$.

For a closed mesh, Laplacian systems $\Delta_{\mathcal{S}}^k \mathbf{P} = \mathbf{B}$ of any order k can be turned into symmetric ones by moving the first diagonal matrix $\mathbf{D}$ to the right-hand side:

$$\mathbf{M}(\mathbf{DM})^{k-1}\mathbf{P} \;=\; \mathbf{D}^{-1}\mathbf{B} \quad . \tag{36}$$

Boundary constraints are typically employed by restricting the values at certain vertices, 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.

12.2. Dense Direct Solvers

Direct linear system solvers are based on a factorization of the matrix $\mathbf{A}$ 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 $\mathbf{A}$ 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 $\mathbf{A} = \mathbf{LL}^T$, with $\mathbf{L}$ 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 $\mathbf{A}$ [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 $\mathbf{A}$ 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. 54, top row).

12.3. Iterative Solvers

In contrast to dense direct solvers, iterative methods are able to exploit the sparsity of the matrix $\mathbf{A}$. 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 $\mathbf{r}^{(i)} := \mathbf{A}\mathbf{x}^{(i)} - \mathbf{b}$. These two are related by the *residual equations* $\mathbf{A}\mathbf{e}^{(i)} = \mathbf{r}^{(i)}$, leading to an upper bound $\left\|\mathbf{e}^{(i)}\right\| \leq \left\|\mathbf{A}^{-1}\right\| \cdot \left\|\mathbf{r}^{(i)}\right\|$, i.e., the norm of the inverse matrix has to be estimated or approximated in some way (see [BBC*94]).

In the case of spd matrices the method of conjugate gradients (CG) [GL89b, She94] is suited best, since it provides guaranteed convergence with monotonically decreasing error. For a spd matrix $\mathbf{A}$ the solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$ is equivalent to the minimization of the quadratic form

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

The CG method successively minimizes this functional along a set of linearly independent $\mathbf{A}$-*conjugate* search directions, such that the exact solution $\mathbf{x}^* \in \mathbb{R}^n$ is found after at most n steps (neglecting rounding errors). The complexity of each CG iteration is mainly determined by the matrix-vector product $\mathbf{A}\mathbf{x}$, 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 $\mathbf{A}$, 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 $\tilde{\mathbf{A}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}$ is solved instead, with $\tilde{\mathbf{A}} = \mathbf{P}\mathbf{A}\mathbf{P}^T$, $\tilde{\mathbf{x}} = \mathbf{P}^{-T}\mathbf{x}$, $\tilde{\mathbf{b}} = \mathbf{P}\mathbf{b}$, using a regular pre-conditioning matrix $\mathbf{P}$, that is chosen such that $\tilde{\mathbf{A}}$ is well conditioned [GL89b, BBC*94]. However, the matrix $\mathbf{P}$ is restricted to have a simple structure, since an additional linear system $\mathbf{P}\mathbf{z} = \mathbf{r}$ 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 and possibly ill-conditioned systems.

12.4. Multigrid Iterative Solvers

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 $\mathbf{A}\mathbf{e}_1 = \mathbf{r}_1$ on $\mathcal{M}_1$, where $\mathbf{r}_1 = R_{0\to1}\mathbf{r}_0$ is the residual on $\mathcal{M}_0$ transferred 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

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

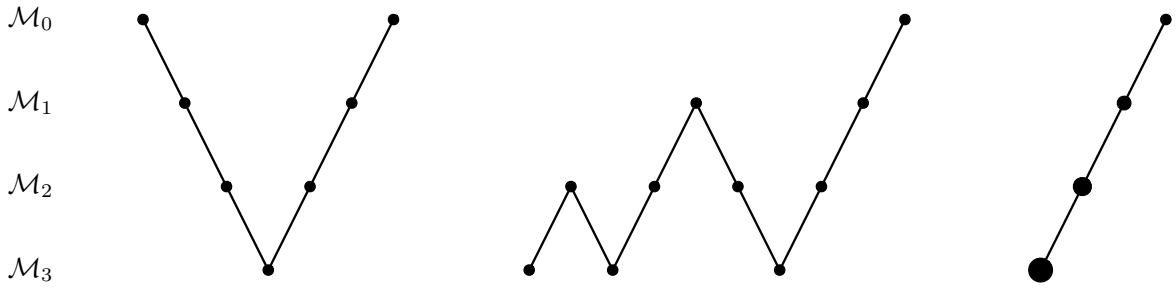


Figure 53: *A schematic comparison in terms of visited multigrid levels for V-cycle (*left*), full multigrid with one V-cycle per level (*center*), and cascading multigrid (*right*).*

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 $\mathbf{A}_k\mathbf{x}_k = \mathbf{b}_k$ 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. 53.

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 [AKS05] some more efficient alternatives to this kind of hierarchy are described.

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. Additionally, appropriate numbers of iterations per hierarchy level have to be chosen. These numbers have to be chosen either by heuristic or experience, since they not only depend on the problem (structure of $\mathbf{A}$), but also on its specific instance (values of $\mathbf{A}$). A detailed overview of these techniques is given in [AKS05].

12.5. 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 $\mathbf{A}$ 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 $\mathbf{A}$ will be preserved. If the matrix $\mathbf{A} = \mathbf{L}\mathbf{L}^T$ has a certain bandwidth β then so has its factor $\mathbf{L}$. An even stricter bound is that the so-called *envelope* (the leading zeros of each row) is preserved [GL81]. 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 $\mathrm{nz}(\mathbf{A})$ of $\mathbf{A}$ [GL81]. Since usually $\mathrm{nz}(\mathbf{A}) = O(n)$, this is the same linear complexity as for multigrid solvers. However, in particular for multiple right-hand side problems, sparse direct methods turned out to be more efficient compared to multigrid solvers.

If matrices are 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 $\mathbf{A} \leftarrow \mathbf{P}^T\mathbf{A}\mathbf{P}$ using a permutation matrix $\mathbf{P}$, i.e., a re-ordering of the mesh vertices. Although this problem is NP complete, several good heuristics exist, of which we will outline the most commonly used in the following. All of these methods work on the undirected *adjacency graph* $\mathrm{Adj}(\mathbf{A})$ corresponding to the non-zeros of $\mathbf{A}$, i.e., two nodes $i, j \in \{1, \ldots, n\}$ are connected by an edge if and only if $\mathbf{A}_{ij} \neq 0$.

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 greedy breadth-first manner. Reverting this permutation further improves the re-ordering, leading to the *reverse Cuthill-McKee* method (RCMK) [LS76]. The result $\mathbf{P}^T\mathbf{A}\mathbf{P}$ of this matrix re-ordering is depicted in the second row of Fig. 54.

Since no special pivoting is required for the Cholesky factorization, the non-zero structure of its matrix factor $\mathbf{L}$ can symbolically be derived from the non-zero structure of the matrix $\mathbf{A}$ alone, or, equivalently, from its adjacency graph. The *minimum degree* algorithm (MD) and its variants [GL89a, Liu85] directly work on the graph interpretation of the Cholesky factorization and try to minimize fill-in elements $\mathbf{L}_{ij} \neq 0 = \mathbf{A}_{ij}$. While the resulting re-orderings do not yield a band-structure (which implicitly limits fill-in), they usually lead to better results compared to RCMK (cf. Fig. 54, third row).

The last class of re-ordering approaches is based on graph partitioning. A matrix $\mathbf{A}$ whose adjacency graph has m separate connected components 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. 54. Besides the obvious fill-in reduction, these systems also allow for easy parallelization of both the factorization and the solution.

Analogously to the dense direct solvers, the factorization can be exploited to solve for different right hand sides in a very efficient manner, since only the back-substitution has to be performed again. Moreover, for sparse direct methods no additional parameters have to be chosen in a problem-dependent manner (like iteration numbers for iterative solvers). The only degree of freedom is the matrix re-ordering, which only depends on the symbolic structure of the problem and therefore can be chosen quite easily. A highly efficient implementation is publicly available in the TAUCS library [TCR03] or recently in COLMOD [DH05].

12.6. Non-Symmetric Indefinite Systems

When the assumptions about the symmetry and positive definiteness of the matrix $\mathbf{A}$ 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.

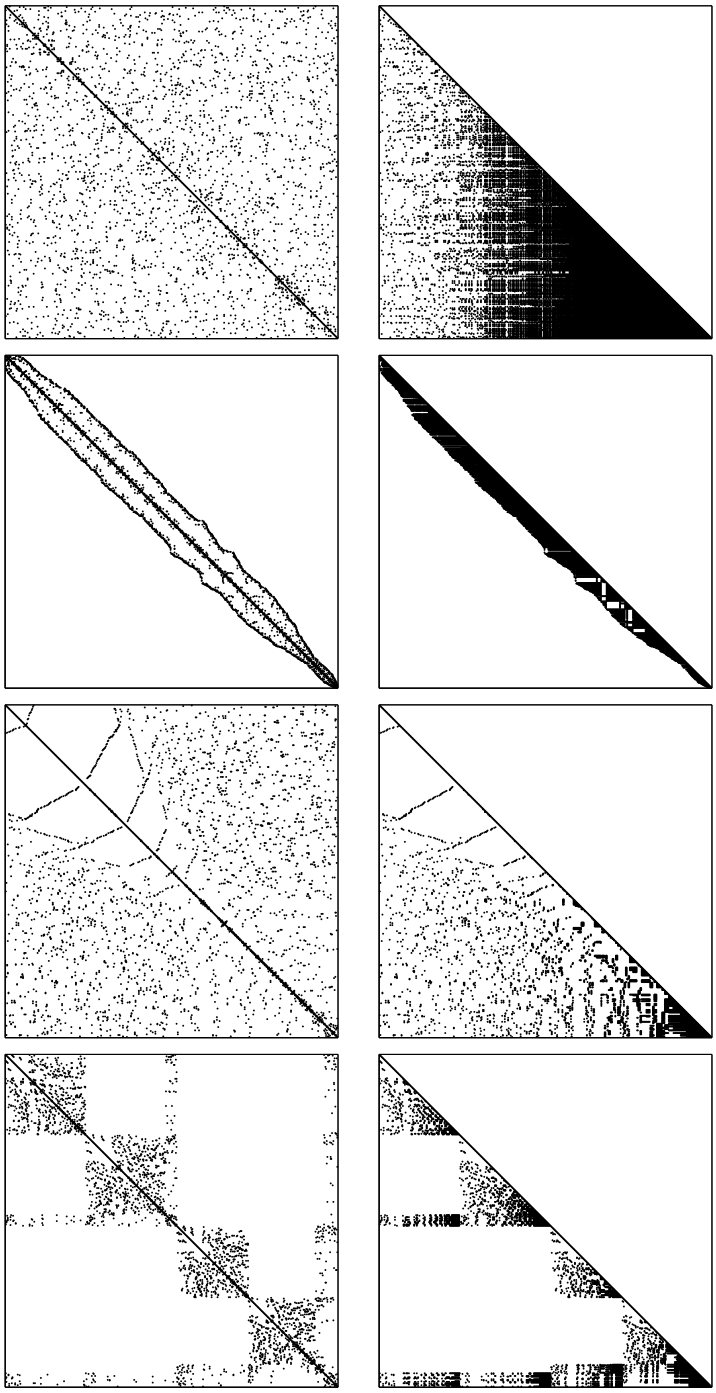


Figure 54: *The top row shows the non-zero pattern of a typical* 500×500 *matrix* $\mathbf{A}$ *and its Cholesky factor* $\mathbf{L}$*, corresponding to a Laplacian system on a triangle mesh. Although* $\mathbf{A}$ *is highly sparse (3502 non-zeros), the factor* $\mathbf{L}$ *is dense (36k non-zeros). The reverse Cuthill-McKee algorithm minimizes the envelope of the matrix, resulting in 14k non-zeros of* $\mathbf{L}$ *(*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.*

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 considerably more complex factorization. A highly efficient implementation of a sparse LU factorization is provided by the SuperLU library [DEG*99].

12.7. 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 $\mathbf{LDL}^T$ factorization as preconditioner. The cascading multigrid solver of [BK04a] (*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.

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.

Table 1 shows timings for the different solvers on Laplacian systems $\Delta_S \mathbf{P} = \mathbf{B}$ of 10k to 50k and 100k to 500k unknowns. 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).

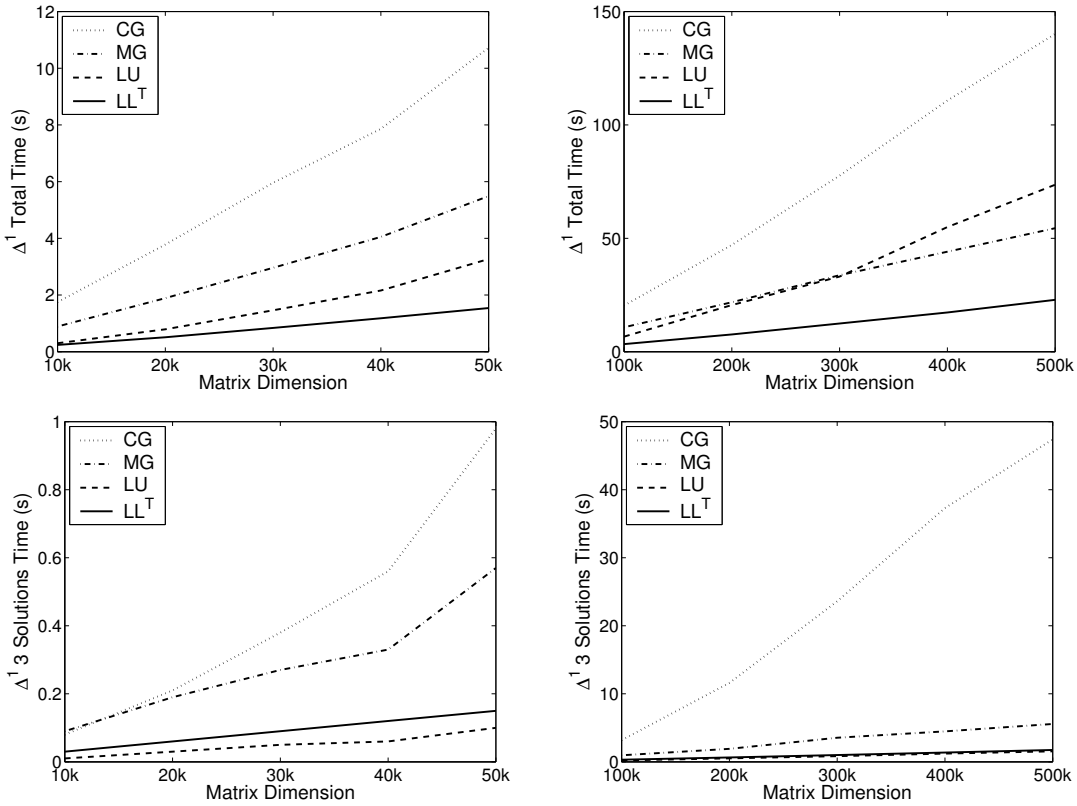
Precomputation: Preconditioning (*iterative*), building the hierarchy by mesh decimation (*multigrid*), matrix re-ordering and sparse factorization (*direct*).

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 break-down of the LU solver, such that the multigrid solver is more efficient in terms of total computation time for the larger systems.

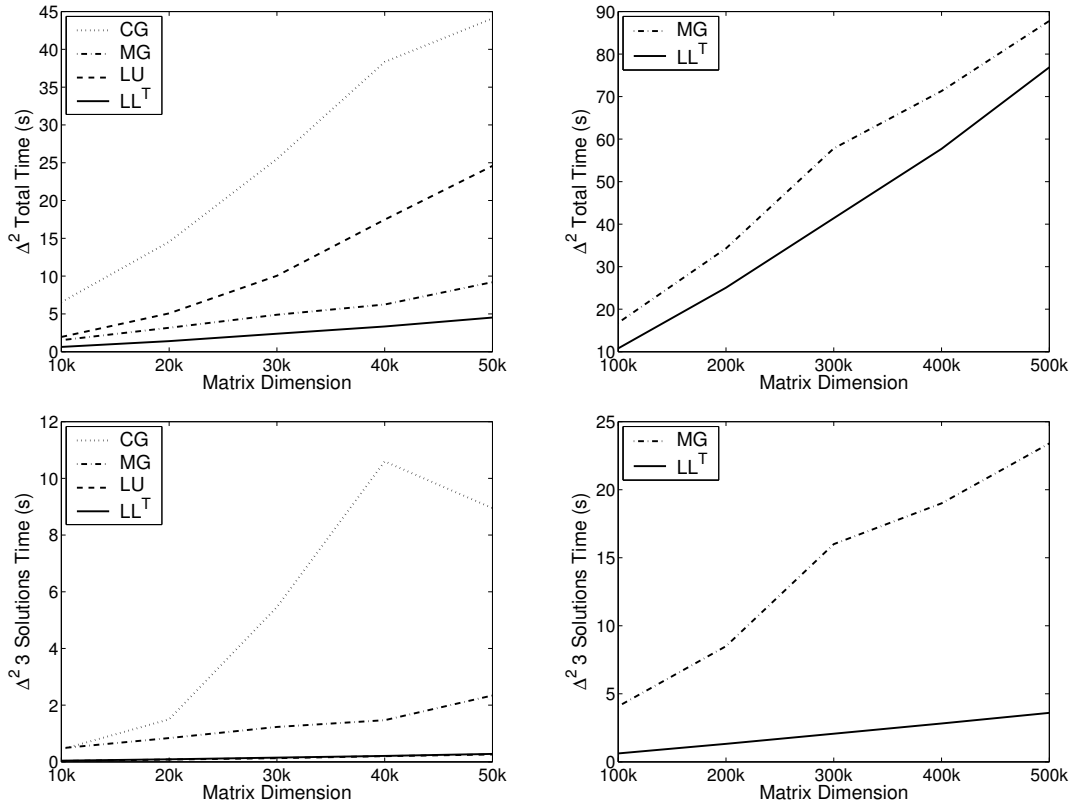
Interactive applications often require to solve the same linear system for several right-hand sides (e.g. once per frame), which typically reflects the change of boundary constraints due to user interaction. For such problems the solution times, i.e., the third columns of the timings, are more relevant, 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 2 shows the same experiments for bi-Laplacian systems $\Delta_S^2 \mathbf{P} = \mathbf{B}$ 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 experiments.



Size	Iterative CG	Multigrid	*LU*	LL^T
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 1: *Comparison of different solvers for Laplacian systems $\Delta_{\mathcal{S}}\mathbf{P} = \mathbf{B}$ of 10k to 50k and 100k to 500k free vertices* $\mathbf{P}$*. The three timings for each solver represent matrix setup, pre-computation, and three solutions for the x, y, and z components of* $\mathbf{P}$*. 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 CG	Multigrid	*LU*	LL^T
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 2: *Comparison of different solvers for bi-Laplacian systems $\Delta_S^2 \mathbf{P} = \mathbf{B}$ of 10k to 50k and 100k to 500k free vertices* $\mathbf{P}$*. The three timings for each solver represent matrix setup, pre-computation, and three solutions for the components of* $\mathbf{P}$*. 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.*

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.

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 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.

Speaker Biographies

Stephan Bischoff graduated in 1999 with a master's in computer science from the University of Karlsruhe, Germany. He then worked at the graphics group of the Max-Planck-Institute for Computer Science in Saarbrücken, Germany. In 2001 he joined the Computer Graphics Group at the Aachen University of Technology, Germany, where he is working as a research associate with Prof. Dr. Leif Kobbelt and is currently pursuing his PhD. He is an experienced speaker and presented courses at Eurographics and Shape Modeling International. His research interests focus on freeform shape representations for efficient geometry processing, topology control techniques for level-set surfaces, reconstruction of medical data sets and the restoration and healing of CAD models.

Mario Botsch is a post-doctoral research associate at the Computer Graphics Laboratory of ETH Zürich. He received his MS in Mathematics and Computer Science from the University of Erlangen, Germany, in 1999. From 1999 to 2000 he worked as research associate at the Max-Planck Institute for Computer Science in Saarbrücken, Germany. From 2001 to 2005 he worked as research associate and PhD candidate with Prof. Dr. Leif Kobbelt at the RWTH Aachen University of Technology, from where he received his PhD in 2005. He has served on the program committees of Solid and Physical Modeling and the Symposium on Point-Based Graphics, of which he is papers co-chair this year. He is an experienced speaker and presented courses at Eurographics and Shape Modeling International. His research interests include geometry processing in general, and mesh generation, mesh optimization, and multiresolution shape editing in particular.

Leif Kobbelt is a full Professor of Computer Science and the Head of the Computer Graphics group at the RWTH Aachen University of Technology, Germany. His research interests include all areas of Computer Graphics and Geometry Processing with a focus on multiresolution and freeform modeling, 3D model optimization, as well as the efficient handling of polygonal mesh data. He was a senior researcher at the Max-Planck Institute for Computer Science in Saarbrücken, Germany, from 1999 to 2000 after he received his Habilitation degree from the University of Erlangen, where he worked from 1996 to 1999. In 1995/96 he spent a post-doc year at the University of Wisconsin, Madison. He received his PhD and MS degrees from the University of Karlsruhe, Germany, in 1994 and 1992, respectively. Dr. Kobbelt's research work during the last years resulted in numerous publications in top scientific journals and international conferences. He is invited regularly to give keynote presentations and tutorial lectures. For his contributions he received several scientific awards. He has ongoing collaborations with colleagues in Europe, North America, and Asia, and frequently serves on international program committees. He organized and co-chaired several workshops and conferences.

Mark Pauly is an assistant professor at the computer science department of ETH Zurich, Switzerland. From August 2003 to March 2005 he was a postdoctoral scholar at Stanford University, where he also held a position as visiting assistant professor during the summer of 2005. He received his Ph.D. degree in 2003 from ETH Zurich and his M.S. degree in computer science in 1999 from the Technical University of Kaiserslautern, Germany. Dr. Pauly has served on various program committees including ACM SIGGRAPH, Eurographics, and the Symposium on Geometry Processing, and has co-chaired the Symposium on Point-Based Graphics. He is an experienced speaker and has previously presented courses at SIGGRAPH and Eurographics. His research interests include geometry processing, multi-scale shape modeling and analysis, physics-based animation, and computational geometry.

Christian Rössl is a postdoctoral associate with the Institut National de Recherche en Informatique et en Automatique (INRIA) in Sophia-Antipolis, France. He received his MS in Computer Science from the University of Erlangen, Germany, in 1999. From 1999 to 2005 he worked as a research associate and PhD candidate with Prof. Dr. Hans-Peter Seidel at the Computer Graphics department of the Max-Planck Institute for Computer Science in Saarbrücken, Germany, from where he received his PhD in 2005. He is an experienced speaker and presented a course at Eurographics. His research interests focus on geometry processing and scientific visualization, including shape modeling and analysis, mesh generation, surface parametrization and spline models.

References

[AB03] AANÆS H., BÆRENTZEN J. A.: Pseudo-normals for signed distance computation. In *Proc. of Vision, Modeling and Visualization 03* (2003), pp. 407–413.

[ACD*03] ALLIEZ P., COHEN-STEINER D., DEVILLERS O., LÉVY B., DESBRUN M.: Anisotropic polygonal remeshing. In *Proc. of ACM SIGGRAPH 03* (2003), pp. 485–493.

[ACWK04] ANGELIDIS A., CANI M.-P., WYVILL G., KING S.: Swirling-Sweepers: constant-volume modeling. In *Proc. of Pacific Graphics 04* (2004), pp. 10–15.

[AdVDI03] ALLIEZ P., DE VERDIÈRE E. C., DEVILLERS O., ISENBURG M.: Isotropic surface remeshing. In *Proc. of Shape Modeling International* (2003), pp. 49–58.

[AK04] AMENTA N., KIL Y.: Defining point-set surfaces. In *Proc. of ACM SIGGRAPH 04* (2004).

[AKS05] AKSOYLU B., KHODAKOVSKY A., SCHRÖDER P.: Multilevel Solvers for Unstructured Surface Meshes. *SIAM Journal on Scientific Computing 26*, 4 (2005), 1146–1165.

[Ale00] ALEXA M.: Merging polyhedral shapes with scattered features. In *The Visual Computer*, vol. 16(1). Springer, 2000, pp. 26–37.

[Ale02] ALEXA M.: Wiener filtering of meshes. In *Shape Modeling International* (2002), pp. 51–60.

[AMD02] ALLIEZ P., MEYER M., DESBRUN M.: Interactive geometry remeshing. In *Proc. of ACM SIGGRAPH 02* (2002), pp. 347–354.

[AUGA05] ALLIEZ P., UCELLI G., GOTSMAN C., ATTENE M.: Recent advances in remeshing of surfaces. State-of-the-art report, 2005.

[AWC04] ANGELIDIS A., WYVILL G., CANI M.-P.: Sweepers: Swept user-defined tools for modeling by deformation. In *Proc. of Shape Modeling International 04* (2004), pp. 63–73.

[BAFS94] BURCHARD H. G., AYERS J. A., FREY W. H., SAPIDIS N. S.: Approximation with aesthetic constraints. In *Designing Fair Curves and Surfaces* (1994), pp. 3–28.

[Bar02] BARASH D.: A fundamental relationship between bilateral filtering, adaptive smoothing, and the nonlinear diffusion equation. *IEEE Trans. Pattern Anal. Mach. Intell 24*, 6 (2002), 844–847.

[Bau72] BAUMGART B. G.: Winged-edge polyhedron representation. Technical Report STAN-CS320, Computer Science Department, Stanford University, 1972.

[BBC*94] BARRETT R., BERRY M., CHAN T. F., DEMMEL J., DONATO J., DONGARRA J., EIJKHOUT V., POZO R., ROMINE C., DER VORST H. V.: *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[BBK05] BOTSCH M., BOMMES D., KOBBELT L.: Efficient linear system solvers for geometry processing. In *11th IMA conference on the Mathematics of Surfaces* (2005).

[BBVK04] BOTSCH M., BOMMES D., VOGEL C., KOBBELT L.: GPU-based tolerance volumes for mesh processing. In *Proc. of Pacific Graphics 04* (2004).

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

[BGK92] BRECHBÜHLER C., GERIG G., KÜBLER O.: Towards representation of 3d shape: global surface parametrization. In *IAPR: Proc. of the international workshop on Visual form: analysis and recognition* (1992), pp. 79–88.

[BHM00] BRIGGS W. L., HENSON V. E., MCCORMICK S. F.: *A Multigrid Tutorial*, 2nd ed. SIAM, 2000.

[BK97] BAREQUET G., KUMAR S.: Repairing CAD models. In *Proc. IEEE Visualization* (1997), pp. 363–370.

[BK03a] BENDELS G. H., KLEIN R.: Mesh forging: editing of 3D-meshes using implicitly defined occluders. In *Symposium on Geometry Processing* (2003), pp. 207–217.

[BK03b] BOTSCH M., KOBBELT L.: Multiresolution surface representation based on displacement volumes. In *Proc. of Eurographics 03* (2003), pp. 483–491.

[BK04a] BOTSCH M., KOBBELT L.: An intuitive framework for real-time freeform modeling. In *Proc. of ACM SIGGRAPH 04* (2004), pp. 630–634.

[BK04b] BOTSCH M., KOBBELT L.: A remeshing approach to multiresolution modeling. In *Proc. of Eurographics symposium on Geometry Processing 04* (2004), pp. 189–196.

[BK05a] BISCHOFF S., KOBBELT L.: Structure preserving CAD model repair. *Computer Graphics Forum (Proc. Eurographics 05) 24*, 3 (2005), 527–536.

[BK05b] BOTSCH M., KOBBELT L.: Real-time shape editing using radial basis functions. In *Proc. of Eurographics 05* (2005), pp. 611–621.

[BNK02] BORODIN P., NOVOTNI M., KLEIN R.: Progressive gap closing for mesh repairing. In *Advances in Modelling, Animation and Rendering*, Vince J., Earnshaw R., (Eds.). Springer Verlag, 2002, pp. 201–213.

[BO01] BELYAEV A. G., OHTAKE Y.: Nonlinear diffusion of normals for crease enhancement. In *Vision Geometry X, SPIE Annual Meeting* (2001), pp. 42–47.

[Bot05a] BOTSCH M.: Extended marching cubes implementation. http://www-i8.informatik.rwth-aachen.de/software/software.html, 2002–2005.

[Bot05b] BOTSCH M.: *High Quality Surface Generation and Efficient Multiresolution Editing Based on Triangle Meshes*. PhD thesis, RWTH Aachen University, 2005.

[BPK05] BISCHOFF S., PAVIC D., KOBBELT L.: Automatic restoration of polygon models. *Transactions on Graphics 24*, 4 (2005), 1332–1352.

[BS95] BAREQUET G., SHARIR M.: Filling gaps in the boundary of a polyhedron. *Computer-Aided Geometric Design 12*, 2 (1995), 207–229.

[BS05a] BOBENKO A. I., SPRINGBORN B. A.: A discrete Laplace-Beltrami operator for simplicial surfaces. In *preprint* (2005).

[BS05b] BOBERNKO A. I., SCHRÖDER P.: Discrete Willmore flow. In *Symposium on Geometry Processing* (2005), pp. 101–110.

[BSM05] BOTSCH M., SOVAKAR A., MARINOV M.: OpenMesh implementation. http://www.openmesh.org, 2002–2005.

[BV93] BATTISTA G. D., VISMARA L.: Angles of planar triangular graphs. *SIAM Journal on Discrete Mathematics 9*, 3 (1993), 349–359.

[BX03] BAJAJ C. L., XU G.: Anisotropic diffusion of surfaces and functions on surfaces. *ACM Transactions on Graphics 22*, 1 (2003), 4–32.

[BZK04] BORODIN P., ZACHMANN G., KLEIN R.: Consistent normal orientation for polygonal meshes. *Proc. Computer Graphics International (CGI)* (2004), 18–25.

[CAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. In *Proc. of ACM SIGGRAPH 04* (2004), pp. 905–914.

[CBC*01] CARR J. C., BEATSON R. K., CHERRIE J. B., MITCHELL T. J., FRIGHT W. R., MCCALLUM B. C., EVANS T. R.: Reconstruction and representation of 3D objects with radial basis functions. In *Proc. of ACM SIGGRAPH 01* (2001), pp. 67–76.

[CD02] COHEN-STEINER D., DESBRUN M.: Hindsight: LSCM and DNCP are one and the same, 2002.

[CDR00] CLARENZ U., DIEWALD U., RUMPF M.: Anisotropic geometric diffusion in surface processing. In *IEEE Visualization* (2000), pp. 397–406.

[Chu97] CHUNG F.: *Spectrac Graph Theory*. American Mathematical Society, 1997.

[CKS98] CAMPAGNA S., KOBBELT L., SEIDEL H.-P.: Directed edges — a scalable representation for triangle meshes. *ACM Journal of Graphics Tools 3*, 4 (1998).

[CL93] CARBAL B., LEEDOM L. C.: Imaging vector fields using line integral convolution. In *SIGGRAPH 93 Conference Proceedings* (1993), pp. 263–274.

[CL96] CURLESS B., LEVOY M.: A volumetric method for building complex models from range images. In *Proc. of ACM SIGGRAPH 96* (1996), pp. 303–312.

[CLR04] CLARENZ U., LITKE N., RUMPF M.: Axioms and variational problems in surface parameterization. In *Computer Aided Geometric Design* (2004), pp. 727–749.

[CM69] CUTHILL E., MCKEE J.: Reducing the bandwidth of sparse symmetric matrices. In *Proc. of the 24th ACM National Conference* (1969), pp. 157–172.

[CM03] COHEN-STEINER D., MORVAN J.-M.: Restricted Delaunay triangulations and normal cycle. In *Proc. of ACM symposium on Compututational geometry '03* (2003), pp. 237–246.

[CMR*99] CIGNONI P., MONTANI C., ROCCHINI C., SCOPIGNO R., TARINI M.: Preserving attribute values on simplified meshes by re–sampling detail textures. In *The Visual Computer* (1999), pp. 519–539.

[CMS98] CIGNONI P., MONTANI C., SCOPIGNO R.: A comparison of mesh simplification algorithms. In *Computers & Graphics* (1998), pp. 37–54.

[COM98] COHEN J., OLANO M., MANOCHA D.: Appearance–preserving simplification. In *Proc. of SIGGRAPH 98* (1998), pp. 115–122.

[Cox89] COXETER H. S. M.: *Introduction to Geometry*, 2nd ed. Wiley, 1989.

[CP03] CAZALS F., POUGET M.: Estimating differential quantities using polynomial fitting of osculating jets. In *Symposium on Geometry Processing* (2003), pp. 177–187.

[CRS98] CIGNONI P., ROCCHINI C., SCOPIGNO R.: Metro: measuring error on simplified surfaces. *Computer Graphics Forum 17(2)* (1998), 167–174.

[CS99] CHOPP D., SETHIAN J.: Motion by intrinsic laplacian of curvature. In *Interfaces and Free Boundaries 1* (1999).

[CVM*96] COHEN J., VARSHNEY A., MANOCHA D., TURK G., WEBER H., AGARWAL P., BROOKS, JR. F. P., WRIGHT W.: Simplification envelopes. In *Proc. of ACM SIGGRAPH 96* (1996), pp. 119–128.

[DBG*06] DONG S., BREMER P.-T., GARLAND M., PASCUCCI V., HART J.: Spectral surface quadrangulation. *ACM Transactions on Graphics (Proc. SIGGRAPH)* (2006).

[dC76] DO CARMO M. P.: *Differential Geometry of Curves and Surfaces*. Prentice Hall, 1976.

[DEG*99] DEMMEL J. W., EISENSTAT S. C., GILBERT J. R., LI X. S., LIU J. W. H.: A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications 20*, 3 (1999), 720–755.

[DH05] DAVIS T. A., HAGER W.: Cholmod: supernodal sparse cholesky factorization and update/downdate. http://www.cise.ufl.edu/research/sparse/cholmod, 2005.

[DKG05] DONG S., KIRCHER S., GARLAND M.: Harmonic functions for quadrilateral remeshing of arbitrary manifolds. *Computer Aided Geometry Design, Special Issue on Geometry Processing 22*, 5 (2005), 392–423.

[DMA02] DESBRUN M., MEYER M., ALLIEZ P.: Intrinsic parameterizations of surface meshes. *Computer Graphics Forum (Proc. Eurographics) 21*, 3 (2002), 209–218.

[DMGL02] DAVIS J., MARSCHNER S., GARR M., LEVOY M.: Filling holes in complex surfaces using volumetric diffusion. In *Proc. International Symposium on 3D Data Processing, Visualization, Transmission* (2002), pp. 428–438.

[DMK03] DEGENER P., MESETH J., KLEIN R.: An adaptable surface parameterization method. *Proc. 9th International Meshing Roundtable* (2003), 201–213.

[DMSB99] DESBRUN M., MEYER M., SCHRÖDER P., BARR A. H.: Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proc. of ACM SIGGRAPH 99* (1999), pp. 317–324.

[DMSB00] DESBRUN M., MEYER M., SCHRÖDER P., BARR A. H.: Anisotropic Feature-Preserving denoising of height fields and images. In *Proc. of Graphics Interface* (2000), pp. 145–152.

[Duc77] DUCHON J.: Spline minimizing rotation-invariant semi-norms in Sobolev spaces. In *Constructive Theory of Functions of Several Variables*, Schempp W., Zeller K., (Eds.), no. 571 in Lecture Notes in Mathematics. Springer Verlag, 1977, pp. 85–100.

[EDD*95] ECK M., DEROSE T., DUCHAMP T., HOPPE H., LOUNSBERY M., STUETZLE W.: Multiresolution analysis of arbitrary meshes. In *Proc. of ACM SIGGRAPH 95* (1995), pp. 173–182.

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

[FB88] FORSEY D. R., BARTELS R. H.: Hierarchical B-spline refinement. In *Proc. of ACM SIGGRAPH 88* (1988), pp. 205–212.

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

[FDC03] FLEISHMAN S., DRORI I., COHEN-OR D.: Bilateral mesh denoising. *ACM Transactions on Graphics (Proc. SIGGRAPH) 22*, 3 (2003), 950–953.

[FH03] FLORIANI L. D., HUI A.: A scalable data structure for three-dimensional non-manifold objects. In *Symposium on Geometry Processing* (2003), pp. 72–82.

[FH05a] FLOATER M. S., HORMANN K.: Surface parameterization: a tutorial and survey. In *Advances in Multiresolution for Geometric Modelling*, Dodgson N. A., Floater M. S.,, Sabin M. A., (Eds.), Mathematics and Visualization. Springer, Berlin, Heidelberg, 2005, pp. 157–186.

[FH05b] FLORIANI L. D., HUI .: Data structures for simplicial complexes: An analysis and A comparison. In *Symposium on Geometry Processing* (2005), pp. 119–128.

[FHR02] FLOATER M. S., HORMANN K., REIMERS M.: Parameterization of manifold triangulations. In *Approximation Theory X: Abstract and Classical Analysis*, Chui C. K., Schumaker L. L.,, Stöckler J., (Eds.). Vanderbilt University Press, 2002, pp. 197–209.

[Flo97] FLOATER M. S.: Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design 14*, 3 (1997), 231–250.

[Flo03a] FLOATER M. S.: Mean value coordinates. *Computer Aided Geometric Design 20*, 1 (2003), 19–27.

[Flo03b] FLOATER M. S.: One-to-one piecewise linear mappings over triangulations. *Math. Comput. 72*, 242 (2003), 685–696.

[FPRJ00] FRISKEN S., PERRY R., ROCKWOOD A., JONES T.: Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proc. of ACM SIGGRAPH 00* (2000), pp. 249–254.

[FSD05] FRIEDEL I., SCHRÖDER P., DESBRUN M.: Unconstrained spherical parameterization. In *SIGGRAPH technical sketch* (2005).

[GGH02] GU X., GORTLER S. J., HOPPE H.: Geometry images. In *Proc. of ACM SIGGRAPH 02* (2002), pp. 355–361.

[GGK02] GOTSMAN C., GUMHOLD S., KOBBELT L.: Simplification and compression of 3d meshes. In *Tutorials on multiresolution in geometric modeling*, A. Iske E. Quak M. F., (Ed.). Springer, 2002.

[GGS03] GOTSMAN C., GU X., SHEFFER A.: Fundamentals of spherical parameterization for 3D meshes. *ACM Transactions on Graphics (Proc. SIGGRAPH) 22*, 3 (2003), 358–363.

[GGT05] GORTLER S. J., GOTSMAN C., THURSTON D.: One-forms on meshes and applications to 3d mesh parameterization. *Computer Aided Geometric Design* (2005).

[GH97] GARLAND M., HECKBERT P.: Surface simplification using quadric error metrics. In *Proc. of ACM SIGGRAPH 97* (1997), pp. 209–216.

[GH98] GARLAND M., HECKBERT P.: Simplifying surfaces with color and texture using quadric error metrics. In *IEEE Visualization Conference Proceedings* (1998).

[GI04] GOLDFEATHER J., INTERRANTE V.: A novel cubic-order algorithm for approximating principal directions vectors. *ACM Transactions on Graphics 23*, 1 (2004), 45–63.

[GK03] GRESS A., KLEIN R.: Efficient representation and extraction of 2-manifold isosurfaces using kd-trees. In *Proc. 11th Pacific Conference on Computer Graphics and Applications (PG 2003)* (2003), pp. 364–376.

[GL81] GEORGE A., LIU J. W. H.: *Computer solution of large sparse positive definite matrices*. Prentice Hall, 1981.

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

[GL89b] GOLUB G. H., LOAN C. F. V.: *Matrix Computations*. Johns Hopkins University Press, Baltimore, 1989.

[GLW96] GREINER G., LOOS J., WESSELINK W.: Data dependent thin plate energy and its use in interactive surface modeling. In *Proc. of Eurographics 96* (1996), pp. 175–186.

[Gre94] GREINER G.: Variational design and fairing of spline surfaces. In *Proc. of Eurographics 94* (1994), pp. 143–154.

[GS85] GUIBAS L., STOLFI J.: Primitives for the manipulation of general subdivisions and computation of voronoi diagrams. *ACM Transactions on Graphics 4*, 2 (Apr. 1985), 74–123.

[GSS99] GUSKOV I., SWELDENS W., SCHRÖDER P.: Multiresolution signal processing for meshes. In *Proc. of ACM SIGGRAPH 99* (1999), pp. 325–334.

[GTLH01] GUÉZIEC A., TAUBIN G., LAZARUS F., HORN B.: Cutting and stitching: Converting sets of polygons to manifold surfaces. *IEEE Transactions on Visualization and Computer Graphics 7*, 2 (2001), 136–151.

[Gus02] GUSKOV I.: An anisotropic mesh parameterization scheme. In *IMR* (2002), pp. 325–332.

[GVSS00] GUSKOV I., VIDIMCE K., SWELDENS W., SCHRÖDER P.: Normal meshes. In *Proc. of ACM SIGGRAPH 00* (2000), pp. 95–102.

[GW01] GUSKOV I., WOOD Z. J.: Topological noise removal. In *Proc. of Graphics Interface 2001* (2001), pp. 19–26.

[GWC*04] GU X., WANG Y., CHAN T. F., THOMPSON P. M., YAU S.-T.: Genus zero surface conformal mapping and its application to brain surface mapping. *IEEE Trans. Med. Imaging 23*, 8 (2004), 949–958.

[GY02] GU X., YAU S.-T.: Computing conformal structures of surfaces. *Communications in Information and Systems 2* (2002), 121–146.

[GY03] GU X., YAU S.-T.: Global conformal surface parameterization. In *Symposium on Geometry Processing* (2003), pp. 127–137.

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

[HAT*00] HAKER S., ANGENENT S., TANNENBAUM A., KIKINIS R., SAPIRO G., HALLE M.: Conformal surface parameterization for texture mapping. *IEEE Transactions on Visualization and Computer Graphics 6*, 2 (2000), 181–189.

[HDD*93] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Mesh optimization. In *SIGGRAPH 93 Conference Proceedings* (1993), pp. 19–26.

[HG00] HORMANN K., GREINER G.: MIPS: An efficient global parametrization method. In *Curve and Surface Design: Saint-Malo 1999*, Laurent P.-J., Sablonnière P.,, Schumaker L. L., (Eds.). Vanderbilt University Press, 2000, pp. 153–162.

[HGC99] HORMANN K., GREINER G., CAMPAGNA S.: Hierarchical parametrization of triangulated surfaces. In *Proc. of Vision, Modeling, and Visualization* (1999), pp. 219–226.

[HHK92] HSU W. M., HUGHES J. F., KAUFMAN H.: Direct manipulation of free-form deformations. In *Proc. of ACM SIGGRAPH 92* (1992), pp. 177–184.

[HHS*92] HAGEN H., HAHMANN S., SCHREIBER T., NAKAYIMA Y., WÖRDENWEBER B., HOLLEMANN-GRUNDSTEDT P.: Surface interrogation algorithms. *IEEE Computer Graphics and Applications 12*, 5 (1992), 53–60.

[Hop96] HOPPE H.: Progressive meshes. In *Proc. of ACM SIGGRAPH 96* (1996), pp. 99–108.

[HP04] HILDEBRANDT K., POLTHIER K.: Anisotropic filtering of non-linear surface features. *Computupter Graphics Forum (Proc. Eurographics) 23*, 3 (2004), 391–400.

[HSZ87] HARALICK R. M., STERNBERG S. R., ZHUANG X.: Image analysis using mathematical morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence 9*, 4 (1987), 532–550.

[HWC*05] HO C.-C., WU F.-C., CHEN B.-Y., CHUANG Y.-Y., OUHYOUNG M.: Cubical marching squares: Adaptive feature preserving surface extraction from volume data. In *Proc. of Eurographics 05* (2005).

[ILGS03] ISENBURG M., LINDSTROM P., GUMHOLD S., SNOEYINK J.: Large mesh simplification using processing sequences. In *Proc. of IEEE Visualization* (2003), pp. 465–472.

[JDD03] JONES T. R., DURAND F., DESBRUN M.: Non-iterative, feature-preserving mesh smoothing. *ACM Transactions on Graphics (Proc. SIGGRAPH) 22*, 3 (2003), 943–949.

[JKS05] JULIUS D., KRAEVOY V., SHEFFER A.: D-charts: Quasi-developable mesh segmentation. *Computer Graphics Forum (Proc. Eurographics) 24*, 3 (2005), 581–590.

[JLSW02] JU T., LASASSO F., SCHAEFER S., WARREN J.: Dual contouring of hermite data. In *Proc. of ACM SIGGRAPH 02* (2002), pp. 339–346.

[Ju04] JU T.: Robust repair of polygonal models. In *Proc. ACM SIGGRAPH 04* (2004), pp. 888–895.

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

[KBS00] KOBBELT L., BAREUTHER T., SEIDEL H.-P.: Multiresolution shape deformations for meshes with dynamic vertex connectivity. In *Proc. of Eurographics 00* (2000), pp. 249–260.

[KBSS01] KOBBELT L., BOTSCH M., SCHWANECKE U., SEIDEL H.-P.: Feature sensitive surface extraction from volume data. In *Proc. of ACM SIGGRAPH 01* (2001), pp. 57–66.

[KCS98] KOBBELT L., CAMPAGNA S., SEIDEL H.-P.: A general framework for mesh decimation. In *Proc. of Graphics Interface 98* (1998), pp. 43–50.

[KCVS98] KOBBELT L., CAMPAGNA S., VORSATZ J., SEIDEL H.-P.: Interactive multi-resolution modeling on arbitrary meshes. In *Proc. of ACM SIGGRAPH 98* (1998), pp. 105–114.

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

[KG00] KARNI Z., GOTSMAN C.: Spectral compression of mesh geometry. In *Proc. of ACM SIGGRAPH* (2000), pp. 279–286.

[KGG05] KARNI Z., GOTSMAN C., GORTLER S. J.: Free-boundary linear parameterization of 3d meshes in the presence of constraints. In *Shape Modeling International* (2005).

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

[Kli80] KLINCSEK G.: Minimal triangulation of polygonal domains. *Annals of Discrete Mathemathics 9* (1980), 121–123.

[KLS96] KLEIN R., LIEBICH G., STRASSER W.: Mesh reduction with error control. In *Proc. of Visualization 96* (1996), pp. 311–318.

[KLS03] KHODAKOVSKY A., LITKE N., SCHRÖDER P.: Globally smooth parameterizations with low distortion. *ACM Transactions on Graphics (Proc. SIGGRAPH) 22*, 3 (2003), 350–357.

[Kob97] KOBBELT L.: Discrete fairing. In *Proc. on 7th IMA Conference on the Mathematics of Surfaces* (1997), pp. 101–131.

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

[KS04] KRAEVOY V., SHEFFER A.: Cross-parameterization and compatible remeshing of 3D models. *ACM Transactions on Graphics 23*, 3 (2004), 861–869.

[KSS06] KHAREVYCH L., SPRINGBORN B., SCHRÖDER P.: Discrete conformal mappings via circle patterns. *ACM Transactions on Graphics 25*, 2 (2006), to appear.

[KT03] KATZ S., TAL A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics (Proc. SIGGRAPH) 22*, 3 (2003), 954–961.

[KVLS99a] KOBBELT L., VORSATZ J., LABSIK U., SEIDEL H.-P.: A shrink wrapping approach to remeshing polygonal surfaces. In *Proc. of Eurographics 99* (1999), pp. 119–130.

[KVLS99b] KOBBELT L., VORSATZ J., LABSIK U., SEIDEL H.-P.: A shrink wrapping approach to remeshing polygonal surfaces. *Computer Graphics Forum (EG 99 proc.) 18* (1999), 119–130.

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

[LBS05] LANGER T., BELYAEV A., SEIDEL H.-P.: Exact and approximate quadratures for curvature tensor estimation. Extended Abstract, 2005.

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: a high resolution 3D surface construction algorithm. In *Proc. of ACM SIGGRAPH 87* (1987), pp. 163–170.

[Lév01] LÉVY B.: Constrained texture mapping for polygonal meshes. In *Proc. SIGGRAPH* (2001), pp. 417–424.

[Lév03] LÉVY B.: Dual domain extrapolation. *ACM Transactions on Graphics 22*, 3 (2003), 364–369.

[Lév06] LÉVU B.: Laplace-beltrami eigenfuntions — towards an algorithm that "understands" geometry. In *Shape Modeling International* (2006).

[LHSW03] LOSASSO F., HOPPE H., SCHAEFER S., WARREN J.: Smooth geometry images. In *Eurographics Symposium on Geometry Processing* (2003), pp. 138–145.

[Lie03] LIEPA P.: Filling holes in meshes. In *Symposium on Geometry Processing* (2003), pp. 200–205.

[Lin00] LINDSTROM P.: Out-of-core simplification of large polygonal models. In *Proc. of ACM SIGGRAPH 00* (2000), pp. 259–262.

[Liu85] LIU J. W. H.: Modification of the minimum-degree algorithm by multiple elimination. *ACM Trans. Math. Softw. 11*, 2 (1985), 141–153.

[LMH00] LEE A., MORETON H., HOPPE H.: Displaced subdivision surfaces. In *Proc. of ACM SIGGRAPH 00* (2000), pp. 85–94.

[LPRM02a] LÉVY B., PETITJEAN S., RAY N., MAILLOT J.: Least squares conformal maps for automatic texture atlas generation. In *Proc. of ACM SIGGRAPH 02* (2002), pp. 362–371.

[LPRM02b] LÉVY B., PETITJEAN S., RAY N., MAILLOT J.: Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics 21, 3* (2002), 362–371.

[LS76] LIU J. W. H., SHERMAN A. H.: Comparative analysis of the Cuthill-McKee and the reverse Cuthill-McKee ordering algorithms for sparse matrices. *SIAM J. Numerical Analysis 2*, 13 (1976), 198–213.

[LS01] LINDSTROM P., SILVA C.: A memory insensitive technique for large model simplification. In *IEEE Visualization '01* (2001).

[LSC*04] LIPMAN Y., SORKINE O., COHEN-OR D., LEVIN D., RÖSSL C., SEIDEL H.-P.: Differential coordinates for interactive mesh editing. In *Proc. of Shape Modeling International 04* (2004), pp. 181–190.

[LSLC05] LIPMAN Y., SORKINE O., LEVIN D., COHEN-OR D.: Linear rotation-invariant coordinates for meshes. In *Proc. of ACM SIGGRAPH 05* (2005), pp. 479–487.

[LSS*98] LEE A. W. F., SWELDENS W., SCHRÖDER P., COWSAR L., DOBKIN D.: MAPS: Multiresolution adaptive parameterization of surfaces. In *Proc. of ACM SIGGRAPH 98* (1998), pp. 95–104.

[Man88] MANTYLA M.: *An introduction to solid modeling*. Computer Science Press, 1988.

[MDSB03] MEYER M., DESBRUN M., SCHRÖDER P., BARR A. H.: Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and Mathematics III*, Hege H.-C., Polthier K., (Eds.). Springer-Verlag, Heidelberg, 2003, pp. 35–57.

[MF97] MURALI T. M., FUNKHOUSER T. A.: Consistent solid and boundary representations from arbitrary polygonal data. In *Proc. Symposium on Interactive 3D Graphics* (1997), pp. 155–162.

[MK04] MARINOV M., KOBBELT L.: Direct anisotropic quad-dominant remeshing. In *Proc. of Pacific Graphics 04* (2004), pp. 207–216.

[MK05] MARINOV M., KOBBELT L.: Automatic generation of structure preserving multiresolution models. *Computer Graphics Forum (Eurographics proceedings) 24*, 3 (2005), 479–486.

[MK06] MARINOV M., KOBBELT L.: Structure recovery via hybrid variational surface approximation. *Computer Graphics Forum (Eurographics proceedings) to appear* (2006).

[MS92] MORETON H. P., SÉQUIN C. H.: Functional optimization for fair surface design. In *Proc. of ACM SIGGRAPH 92* (1992), pp. 167–176.

[MSS94] MONTANI C., SCATENI R., SCOPIGNO R.: A modified look-up table for implicit disambiguation of marching cubes. *The Visual Computer 10*, 6 (1994), 353–355.

[MYC*01] MORSE B. S., YOO T. S., CHEN D. T., RHEINGANS P., SUBRAMANIAN K. R.: Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *Proc. of Shape Modeling & Applications 01* (2001), pp. 89–98.

[NT03] NOORUDDIN F., TURK G.: Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics 9*, 2 (2003), 191–205.

[OB04] OHTAKE Y., BELYAEV A.: Ridge-valley lines on meshes via implicit surface fitting. In *Proc. SIGGRAPH* (2004), pp. 609–612.

[OBB00] OHTAKE Y., BELYAEV A., BOGAEVSKI I. A.: Polyhedral surface smoothing with simultaneous mesh regularization. In *Proc. of Geometric Modeling and Processing* (2000), pp. 229–237.

[OBS02] OHTAKE Y., BELYAEV A. G., SEIDEL H.-P.: Mesh smoothing by adaptive and anisotropic gaussian filter applied to mesh normals. In *Proc. Vision, Modeling, and Visualization (VMV)* (2002), pp. 203–210.

[OBS04] OHTAKE Y., BELYAEV A., SEIDEL H.-P.: 3D scattered data approximation with adaptive compactly supported radial basis functions. In *Proc. of Shape Modeling International 04* (2004).

[O'R94] O'ROURKE J.: *Computational geometry in C*. Cambridge University Press, Cambridge, 1994.

[PBP02] PRAUTZSCH H., BOEHM W., PALUSZNY M.: *Bézier and B-Spline Techniques*. Springer Verlag, 2002.

[PC04] PEYRÉ G., COHEN L.: Surface segmentation using geodesic centroidal tesselation. In *Proceedings 3DPVT'04* (2004), pp. 995–1002.

[Pet01] PETITJEAN S.: A survey of methods for recovering quadrics in triangle meshes. *ACM Computing Surveys 34*, 2 (2001).

[PFTV92] PRESS W. H., FLANNERY B. P., TEUKOLSKY S. A., VETTERLING W. T.: *Numerical Recipes: The Art of Scientific Computing*, 2nd ed. Cambridge University Press, 1992.

[PG01] PAULY M., GROSS M.: Spectral processing of point-sampled geometry. In *Proc. of ACM SIGGRAPH 01* (2001).

[PGK02] PAULY M., GROSS M., KOBBELT L.: Efficient simplification of point-sampled surfaces. In *Proc. of IEEE Visualization 02* (2002).

[PH03] PRAUN E., HOPPE H.: Spherical parametrization and remeshing. *ACM Transactions on Graphics 22*, 3 (2003), 340–349.

[PKKG03] PAULY M., KEISER R., KOBBELT L., GROSS M.: Shape modeling with point-sampled geometry. In *Proc. of ACM SIGGRAPH 03* (2003), pp. 641–650.

[PKS*01] PAGE D. L., KOSCHAN A., SUN Y., PAIK J., ABIDI A.: Robust crease detection and curvature estimation of piecewise smooth surfaces from triangle mesh approximations using normal voting. In *Proceedings on Computer Vision and Pattern Recongition* (2001).

[PM90] PERONA P., MALIK J.: Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence 12*, 7 (1990), 629–639.

[PMG*05] PAULY M., MITRA N., GIESEN J., GROSS M., GUIBAS L. J.: Example-based 3d scan completion. In *Symposium on Geometry Processing* (2005).

[PP93] PINKALL U., POLTHIER K.: Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics 2*, 1 (1993), 15–36.

[PR05] PODOLAK J., RUSINKIEWICZ S.: Atomic volumes for mesh completion. In *Symposium on Geometry Processing* (2005).

[PSS01] PRAUN E., SWELDENS W., SCHRÖDER P.: Consistent mesh parameterizations. In *Proc. SIGGRAPH* (2001), pp. 179–184.

[PSZ01] PENG J., STRELA V., ZORIN D.: A simple algorithm for surface denoising. In *IEEE Visualization* (2001), pp. 107–112.

[PT97] PIEGL L. A., TILLER W.: *The NURBS Book*, 2nd ed. Springer, 1997.

[QBH*00] QUICKEN M., BRECHBÜHLER C., HUG J., BLATTMAN H., SZÉKELY G.: Parameterization of closed surfaces for parametric surface description. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2000), pp. 354–360.

[RB93] ROSSIGNAC J., BORREL P.: Multi-resolution 3D approximations for rendering complex scenes. In *Modeling in Computer Graphics*, Falcidieno B., Kunii T. L., (Eds.). Springer Verlag, 1993, pp. 455–465.

[RL03] RAY N., LÉVY B.: Hierarchical least squares conformal map. In *Pacific Conference on Computer Graphics and Applications* (2003), pp. 263–270.

[RLL*05] RAY N., LI W. C., LÉVY B., SHEFFER A., ALLIEZ P.: Periodic global parameterization. In *(preprint)* (2005).

[RP05] RENARD Y., POMMIER J.: `Gmm++`: a generic template matrix C++ library. http://www-gmm.insa-toulouse.fr/getfem/gmm_intro, 2005.

[Rus04] RUSINKIEWICZ S.: Estimating curvatures and their derivatives on triangle meshes. In *Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT)* (2004).

[SAG03] SURAZHSKY V., ALLIEZ P., GOTSMAN C.: Isotropic remeshing of surfaces: a local parameterization approach. In *Proc. of 12th International Meshing Roundtable* (2003).

[Sam94] SAMET H.: *The Design and Analysis of Spatial Data Structures*. Addison–Wesley, 1994.

[Sap94] SAPIDIS N. S.: *Designing Fair Curves and Surfaces: Shape Quality in Geometric Modeling and Computer-Aided Design*. SIAM, 1994.

[SAPH04] SCHREINER J., ASIRVATHAM A., PRAUN E., HOPPE H.: Inter-surface mapping. *ACM Transactions on Graphics 23*, 3 (2004), 870–877.

[SCGL02] SORKINE O., COHEN-OR D., GOLDENTHAL R., LISCHINSKI D.: Bounded-distortion piecewise mesh parameterization. In *Proc. IEEE Visualization* (2002), pp. 355–362.

[Sch97] SCHROEDER W.: A topology modifying progressive decimation algorithm. In *IEEE Visualization '97 Conference Proceedings* (1997), pp. 205–212.

[SCL*04] SORKINE O., COHEN-OR D., LIPMAN Y., ALEXA M., RÖSSL C., SEIDEL H.-P.: Laplacian surface editing. In *Proc. of Eurographics symposium on Geometry Processing 04* (2004), pp. 179–188.

[SCT03] SORKINE O., COHEN-OR D., TOLEDO S.: High-pass quantization for mesh encoding. In *Symposium on Geometry Processing* (2003), pp. 42–51.

[SD92] SHOEMAKE K., DUFF T.: Matrix animation and polar decomposition. In *Proc. of Graphics Interface 92* (1992), pp. 258–264.

[SdS00] SHEFFER A., DE STURLER E.: Parameterization of faceted surfaces for meshing using angle based flattening. *Engineering with Computers 17*, 3 (2000), 326–337.

[Set96] SETHIAN J.: A fast marching level set method for monotonically advancing fronts. In *Proc. of the National Academy of Science* (1996), vol. 93, pp. 1591–1595.

[SG01] SHAFFER E., GARLAND M.: Efficient adaptive simplification of massive meshes. In *Proc. of IEEE Visualization* (2001), pp. 127–134.

[SG03a] SHEFFER A., GOTSMAN C.: Matchmaker: constructing constrained texture maps. *ACM Transactions on Graphics 22*, 3 (2003), 326–333.

[SG03b] SURAZHSKY V., GOTSMAN C.: Explicit surface remeshing. In *Proc. of Eurographics/ACM SIGGRAPH symposium on Geometry processing 03* (2003), pp. 20–30.

[SGD03] SHEFFER A., GOTSMAN C., DYN N.: Robust spherical parameterization of triangular meshes. In *Proc. of 4th Israel-Korea Binational Workshop on Geometric Modeling and Computer Graphics* (2003), pp. 94–99.

[She94] SHEWCHUK J. R.: *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Tech. rep., Carnegie Mellon University, 1994.

[She97] SHEWCHUK J. R.: *Delaunay refinement mesh generation*. PhD thesis, Carnegie Mellon University, Pittsburg, 1997.

[She02] SHEWCHUK J. R.: What is a good linear element? Interpolation, conditioning, and quality measures. In *Eleventh International Meshing Roundtable* (2002), pp. 115–126.

[SK00] SCHNEIDER R., KOBBELT L.: Generating fair meshes with G^1 boundary conditions. In *Proc. of Geometric Modeling and Processing 2000 (GMP-00)* (2000), pp. 251–261.

[SK01] SCHNEIDER R., KOBBELT L.: Geometric fairing of irregular meshes for free-form surface design. *Computer Aided Geometric Design 18*, 4 (2001), 359–379.

[SKR02] SZYMCZAK A., KING D., ROSSIGNAC J.: Piecewise regular meshes: Construction and compression. *Graphical Models 64*, 3–4 (2002), 183–198.

[SLMB04] SHEFFER A., LÉVY B., MOGILNITSKY M., BOGOMYAKOV A.: ABF++ : Fast and robust angle based flattening. *ACM Transactions on Graphics* (Apr 2004).

[SOS04] SHEN C., O'BRIEN J. F., SHEWCHUK J. R.: Interpolating and approximating implicit surfaces from polygon soup. In *Proc. of ACM SIGGRAPH 04* (2004), pp. 896–904.

[SP86] SEDERBERG T. W., PARRY S. R.: Free-form deformation of solid geometric models. In *Proc. of ACM SIGGRAPH 86* (1986), pp. 151–159.

[SP04] SUMNER R. W., POPOVIĆ J.: Deformation transfer for triangle meshes. In *Proc. of ACM SIGGRAPH 04* (2004), pp. 399–405.

[SSGH01] SANDER P. V., SNYDER J., GORTLER S. J., HOPPE H.: Texture mapping progressive meshes. In *Computer Graphics (Proc. SIGGRAPH)* (2001), pp. 409–416.

[SWG*03] SANDER P., WOOD Z., GORTLER S., SNYDER J., HOPPE H.: Multi-chart geometry images. In *Eurographics Symposium on Geometry Processing* (2003), pp. 146–155.

[SYGS05] SABA S., YAVNEH I., GOTSMAN C., SHEFFER A.: Practical spherical embedding of manifold triangle meshes. In *Proc. Shape Modeling International* (2005), pp. 258–267.

[SZL92] SCHROEDER W., ZARGE J., LORENSEN W.: Decimation of triangle meshes. In *Proc. of SIGGRAPH 92* (1992), pp. 65–70.

[TACSD06] TONG Y., ALLIEZ P., COHEN-STEINER D., DESBRUN M.: Designing quadrangulations with discrete harmonic forms. In *Symposium on Geometry Processing* (2006).

[Tau95a] TAUBIN G.: Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Proceedings International Conference on Computer Vision* (1995), pp. 902–907.

[Tau95b] TAUBIN G.: A signal processing approach to fair surface design. In *Proc. of ACM SIGGRAPH 95* (1995), pp. 351–358.

[Tau00] TAUBIN G.: Geometric signal processing on polygonal meshes. In *Eurographics 00 State of the Art Report* (2000).

[Tau01] TAUBIN G.: Linear anisotropic mesh filtering. In *IBM Research Report RC2213* (2001).

[TCR03] TOLEDO S., CHEN D., ROTKIN V.: Taucs: A library of sparse linear solvers. http://www.tau.ac.il/∼stoledo/taucs, 2003.

[TG98] TOUMA C., GOTSMAN C.: Triangle mesh compression. In *Proc. of Graphics Interface* (1998), pp. 26–34.

[THCM04] TARINI M., HORMANN K., CIGNONI P., MONTANI C.: PolyCube-Maps. *ACM Transactions on Graphics 23*, 3 (2004), 853–860.

[TL94] TURK G., LEVOY M.: Zippered polygon meshes from range images. In *Proc. of ACM SIGGRAPH 94* (1994), pp. 311–318.

[TM98] TOMASI C., MANDUCHI R.: Bilateral filtering for gray and color images. In *ICCV* (1998), pp. 839–846.

[TRZS04] THEISEL H., RÖSSL C., ZAYER R., SEIDEL H.-P.: Normal based estimation of the curvature tensor for triangular meshes. In *Pacific Conference on Computer Graphics and Applications* (2004), pp. 288–297.

[TSS*04] TEWARI G., SNYDER J., SANDER P. V., GORTLER S. J., HOPPE H.: Signal-specialized parameterization for piecewise linear reconstruction. In *Symposium on Geometry Processing* (2004), pp. 57–66.

[Tur92] TURK G.: Re-tiling polygonal surfaces. In *Proc. of ACM SIGGRAPH 92* (1992), pp. 55–64.

[Tut63] TUTTE W. T.: How to draw a graph. *Proc. London Mathematical Society 13* (1963), 743–768.

[TWBO02] TASDIZEN T., WHITAKER R., BURCHARD P., OSHER S.: Geometric surface smoothing via anisotropic diffusion of normals. In *IEEE Visualization* (2002), pp. 125–132.

[TZG96] TAUBIN G., ZHANG T., GOLUB G.: Optimal surface smoothing as filter design. In *European Conference on Computer Vision (ECCV Vol. 1)* (1996), pp. 283–292.

[VC04] VALETTE S., CHASSERY J.-M.: Approximated centroidal voronoi diagrams for uniform polygonal mesh coarsening. *Computer Graphics Forum (Eurographics proceedings) 23*, 3 (2004), 381–389.

[VMM99] VOLLMER J., MENCL R., , MÖLLER H.: Improved laplacian smoothing of noisy surface meshes. *Computer Graphics Forum (Proc. Eurographics) 18*, 3 (1999), 131–138.

[VRS03] VORSATZ J., RÖSSL C., SEIDEL H.-P.: Dynamic remeshing and applications. In *Proc. of Solid Modeling and Applications* (2003), pp. 167–175.

[WB01] WATANABE K., BELYAEV A.: Detection of salient curvature features on polygonal surfaces. In *Proc. Eurographics* (2001), pp. 385–392.

[Wei98] WEICKERT J.: *Anisotropic Diffusion in Image Processing*. Teubner, 1998.

[WHDS04] WOOD Z., HOPPE H., DESBRUN M., SCHRÖDER P.: Removing excess topology from isosurfaces. *ACM Transactions on Graphics 23*, 2 (2004), 190–208.

[WK03] WU J., KOBBELT L.: Piecewise linear approximation of signed distance fields. In *Proc. of Vision, Modeling, and Visualization 03* (2003), pp. 513–520.

[WK04] WU J., KOBBELT L.: A stream algorithm for the decimation of massive meshes. In *Proc. of Graphics Interface 03* (2004), pp. 185–192.

[WK05] WU J., KOBBELT L.: Structure recovery via hybrid variational surface approximation. *Computer Graphics Forum (Eurographics proceedings) 24*, 3 (2005), 277–284.

[WW92] WELCH W., WITKIN A.: Variational surface modeling. In *Proc. of ACM SIGGRAPH 92* (1992), pp. 157–166.

[WW94] WELCH W., WITKIN A.: Free-form shape design using triangulated surfaces. In *Proc. of ACM SIGGRAPH 94* (1994), pp. 247–256.

[YBP97] YIFAN C., BEIER K.-P., PAPAGEORGIOU D.: Direct highlight line modification on nurbs surfaces. *Computer Aided Geometric Design 14*, 6 (1997), 583–601.

[YBS04] YOSHIZAWA S., BELYAEV A. G., SEIDEL H.-P.: A fast and simple stretch-minimizing mesh parameterization. In *Proc. Shape Modeling International* (2004), pp. 200–208.

[YLW06] YAN D.-M., LIU Y., WANG W.: Quadric surface extraction by variational shape approximation. In *Proc. Geometric Modeling and Processing* (2006).

[YZX*04] YU Y., ZHOU K., XU D., SHI X., BAO H., GUO B., SHUM H.-Y.: Mesh editing with Poisson-based gradient field manipulation. In *Proc. of ACM SIGGRAPH 04* (2004), pp. 644–651.

[ZKK02] ZIGELMAN G., KIMMEL R., KIRYATI N.: Texture mapping using surface flattening via multidimensional scaling. *IEEE Transcations on Visualization and Computer Graphics 8*, 2 (2002), 198–207.

[ZMT05] ZHANG E., MISCHAIKOW K., TURK G.: Feature-based surface parameterization and texture mapping. *ACM Transactions on Graphics 24*, 1 (2005), 1–27.

[ZRKS05] ZAYER R., RÖSSL C., KARNI Z., SEIDEL H.-P.: Harmonic guidance for surface deformation. In *Proc. of Eurographics 05* (2005), pp. 601–609.

[ZRS04a] ZAYER R., RÖSSL C., SEIDEL H.-P.: Efficient iterative solvers for angle based flattening. In *Vision, Modeling, and Visualization (VMV)* (2004), pp. 347–354.

[ZRS04b] ZAYER R., RÖSSL C., SEIDEL H.-P.: Variations of angle based flattening. In *Advances in Multiresolution for Geometric Modelling*, Dodgson N. A., Floater M. S.,, Sabin M. A., (Eds.), Mathematics and Visualization. Springer, Berlin, Heidelberg, 2004, pp. 187–199.

[ZRS05a] ZAYER R., RÖSSL C., SEIDEL H.-P.: Discrete tensorial quasi-harmonic maps. In *Proc. Shape Modeling International* (2005), pp. 276–285.

[ZRS05b] ZAYER R., RÖSSL C., SEIDEL H.-P.: Setting the boundary free: A composite approach to surface parameterization. In *Symposium on Geometry Processing* (2005), pp. 91–100.

[ZRS06] ZAYER R., RÖSSL C., SEIDEL H.-.: Curvilinear spherical parameterization. In *Shape Modeling International (SMI)* (2006), p. to appear.

[ZSD*00] ZORIN D., SCHRÖDER P., DEROSE T., KOBBELT L., LEVIN A., SWELDENS W.: Subdivision for modeling and animation. In *Course notes of ACM SIGGRAPH 00* (2000).

[ZSS97] ZORIN D., SCHRÖDER P., SWELDENS W.: Interactive multiresolution mesh editing. In *Proc. of ACM SIGGRAPH 97* (1997), pp. 259–268.