Master Thesis

September 2009

Curvature-adaptive and Feature-sensitive Isotropic Surface Remeshing

Technische Universität Darmstadt, Germany
Department of Computer Science
The Interactive Graphics System Group

Supervisors: Prof. Dr. Michael Goesele, Thomas Kalbe

Simon Fuhrmann

Contents

1 Introduction

Capturing three dimensional shapes from the real-world is of vital importance in the field of computer graphics. Over the last decades an abundance of methods has been developed to derive surface discretizations from real-world objects. These discretizations are a key component in simulation, visualization, animation and reverse engineering of CAD models, mainly motivated by the computer games and film industry, to digitize objects of cultural interest and to aid fabrication of goods, e.g. for the automobile industry.

While one approach is to directly model (or design) the subject with the help of computers, e.g. in CAD applications, another approach is to scan the original object using three dimensional scanner devices. The latter process requires a whole scanning pipeline to acquire a final data record, but the potential to digitize the geometry of an artifact in a semi-automatic or even completely automated process enables rapid prototyping for modeling or even replace time-consuming modeling completely.

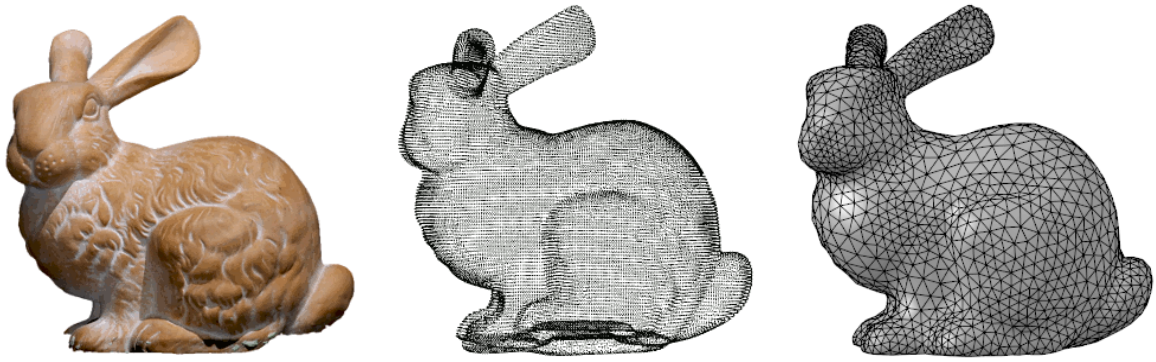


Figure 1: The original Stanford Bunny (left), the point set surface after scanning the object (middle) and a triangulated, reconstructed surface (right).

These scanning techniques generate a discretization of the domain, typically a set of points located on the surface area of the domain, and the resulting record is called a *point set surface*. Depending on the quality and accuracy, the point set may contain thousands or even millions of points. The next step in the scanning pipeline is to find an appropriate continuous surface representation for the point set. There are several ways of describing the surface, ranging from using triangles over quads to general bivariate polynomials as surface descriptors. Finding a proper partition of the domain is one of the main challenges in the pipeline of creating a digital record for rendering, animation or simulation.

One of the most common surface representations are triangle meshes, i.e. triangles are used as primitives to approximately describe the surface. Triangles have many nice properties: They are a special case of bivariate polynomials, namely linear bivariate polynomials, all three points of a triangle are guaranteed to lie on a plane and the

handling of triangles in algorithms is relatively simple. Most important, nowadays graphics hardware is especially optimized for rendering huge numbers of triangles at interactive frame rates.

Although triangular meshes are a quite simple representation of the original surface, finding a "good" partition of the domain is a difficult task. There are several suitable algorithms and methods to extract an initial triangle mesh from the point data, but these raw meshes that emerge from the scanning pipeline have unsatisfactory characteristics. For example, they may have an enormous amount of triangles, triangles with bad shape or inappropriate sampling, which make these meshes bulky, slow to render and it is almost impossible to get reliable simulation results.

This thesis deals with finding a new partition of an existing triangular surface representation, i.e. the task is to *repartition* (or *remesh*) the triangle mesh in order to enhance the discretization. The techniques in this theses do not only apply to large meshes as a result of the scanning process, but also to more general meshes, for example those that are typically generated by CAD applications, with relatively few but lengthy, skinny triangles. Special care has to be taken to not destroy sharp edges (called *features*) in these models to preserve the characteristic appearance and fidelity to the original mesh.

The document at hand is structured in the following sections:

- Section 2 describes some fundamentals for this thesis: The terminology and the mathematical armamentarium that is used herein. The scanning pipeline is shortly reviewed to impart a feeling about the general procedure of capturing real-word data. The section is concluded with a discussion about the meaning of quality and what a high-quality mesh is.

- Section 3 surveys some triangulation techniques that can be used to create triangle meshes from scanner data. Also some remeshing techniques are presented that has been developed and proposed over time.

- Sections 4 forms the heart of this work. The method of isotropic surface remeshing is explained in-depth and a detailed description of the techniques is provided. This includes bringing the mesh to an exact, user-defined vertex budget, and polishing the resulting sampling afterwards. The algorithms are extended using guidance information to create adaptive and feature-sensitive remeshes.

- Section 5 presents several remeshing results that demonstrates the flexibility of the framework. The thesis is concluded with a discussion about open problems and what can be improved in future work.

2 Fundamentals

This sections opens with some definitions and surveys a few important mathematical tools. The scanning pipeline is shortly reviewed and the section closes with a discussion about quality criteria of meshes.

2.1 Definitions and Concepts

In computer graphics, three dimensional models are represented as surfaces. The following definition emphasizes that surfaces are *embedded* in $\mathbb{R}^3$ and only the outer boundary of the three dimensional object is represented, i.e. the object is "hollow".

Definition 2.1 *A* surface $\mathcal{S}$ *is a compact, connected and orientable, two-dimensional manifold embedded in* $\mathbb{R}^3$. *A surface may have a boundary, a surface without boundary is called a* closed surface.

A surface is *compact* if the area of the surface is finite. For example, the area of a sphere is finite, but the area of a plane embedded in $\mathbb{R}^3$ is infinite, thus a plane is not a surface. An orientable, two-dimensional manifold (or just *2-manifold*) can be assigned an inside and an outside and leaving one side for the other is only possible when passing a boundary. The *Möbius Strip* and the *Klein Bottle* are examples for non-orientable 2-manifolds.

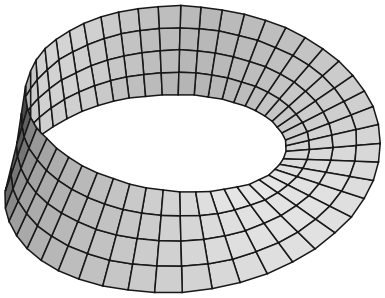


Figure 2: The Möbius Strip

Definition 2.2 *A* point set $\mathcal{P}$ *of a surface* $\mathcal{S}$ *is a set of* n *points* $\mathcal{P} = \{p_1, \ldots, p_n\}$ *with* $p_i = \tilde{p}_i + e_i \in \mathbb{R}^3$ *and* $\tilde{p}_i \in \mathcal{S}$. e_i *is called the* error vector *for* p_i *with error* $\|e_i\|$.

Point set surfaces are a common artifact from the scanning pipeline and consist of points in the vicinity of the real surface $\mathcal{S}$. While optimally $||e_i|| = 0$, scanning devices introduce errors which are accommodated with the error vector.

Definition 2.3 *The* medial axis *of a surface is the set of points with more than one nearest point on the surface. A point set surface* $\mathcal{P}$ *is an* ε-sampling *of a surface* $\mathcal{S}$ *if no point* $s \in \mathcal{S}$ *is farther away from the nearest point* $p \in \mathcal{P}$ *than* ε*-times the distance from* s *to the medial axis.*

Many algorithms use the notion of the ε-sampling to express reconstruction guarantees, i.e. algorithms guarantee that a point set surface $\mathcal{P}$ is correctly reconstructed if $\mathcal{P}$ is at least an ε-sampling of $\mathcal{S}$ with a bounded ε.

Definition 2.4 *A* C^k-continuous surface *is a surface with particular constraints in continuity: A* C^0*-continuous surface is positional continuous without further constraints. A* C^i*-continuous surface is* C^{i-1}*-continuous and its derivative is also* C^{i-1}*-continuous.*

In particular, this means that a C^0-continuous (*positional continuous*) surface may have connections which are not seamless, i.e. may contain sharp edges (or *features*). A C^1-continuous surface has both continuous positions and tangents (*tangential continuous*) and the surface is seamless everywhere. Additionally, a C^2-continuous surface has continuous curvature everywhere.

Definition 2.5 *A piecewise linear surface is called a* simple surface.

This special case of a surface is better suited for discretization, and typical representations are triangle and quad meshes. A simple surface is certainly C^0-continuous everywhere (except at boundaries). The remainder of this thesis will treat triangle meshes only.

Definition 2.6 *A* triangulation $\mathcal{T}$ *of a surface is a simple surface of connected triangles forming an orientable, two-dimensional manifold.*

It is obvious that connected triangles are powerful enough to build up more general shapes, not only simple surfaces. For example the Möbius strip can also be build with a set of suitable triangles or quads. The concepts and algorithms in this thesis are not limited to simple surfaces, but to more general types of triangulations.

Definition 2.7 *A* triangle mesh $\mathcal{M}$ *is a set of triangles, possibly not connected, with or without boundaries and not necessarily an orientable 2-manifold, i.e. in general not a simple surface. The mesh* $\mathcal{M} = (V, F)$ *consists of a list of vertices* V *and a list of triangles* F*, where each triangle* $T \in F$ *connects three vertices* $(v_1, v_2, v_3), v_i \in V$*.*

In practice, triangle meshes are often invalid simple surfaces. Two triangles are *connected* if they share a common edge, i.e. they share two common vertices. Even though triangle meshes may not be orientable, each triangle has an implicit orientation given by the order the vertices are issued. The literature concludes on having counter-clockwise oriented triangles, that is if v_1, v_2 and v_3 are issued counter-clockwise, the normal n of the triangle points upwards (to the direction of the viewer), see Figure 3.

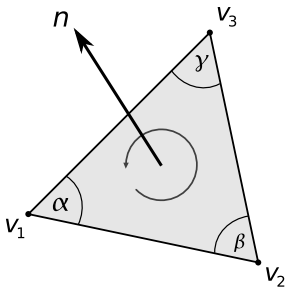


Figure 3: A triangle

Barycentric Coordinates

Every point $v = (x, y) \in \mathbb{R}^2$ has a unique representation regarding a non-degenerate triangle $T = (v_1, v_2, v_3)$ in the form

$$v = b_1 v_1 + b_2 v_2 + b_3 v_3 \quad \text{with} \quad b_1 + b_2 + b_3 = 1$$

Definition 2.8 *The numbers* b_1, b_2, b_3 *are called a* barycentric coordinate *of* v *with respect to a triangle* T*. The numbers sum to unity, i.e.* $\sum b_i = 1$*.*

Barycentric coordinates are much more useful than the usual Cartesian coordinates when working with triangles because they have many nice properties. If all b_i's are positive, the point $v = \sum b_i v_i$ is located inside the triangle. In matrix form, the above equations become $M \cdot b = \tilde{v}$:

$$\begin{pmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 1 \\ x \\ y \end{pmatrix}$$

and the area A_T of T is given by the expression

$$A_T = \frac{1}{2} \det(M)$$

This area is always positive as long as T is not degenerated and the vertices v_i of T are given in counter-clockwise order. Barycentric coordinates are also referred to as *areal coordinates* because of their geometric interpretation. Given $v \in T$, let $T_1 := (v, v_2, v_3)$, $T_2 := (v, v_3, v_1)$ and $T_3 := (v, v_1, v_2)$, see Figure 4. The barycentric coordinates of v relative to T are given by the ratio of the sub-areas A_{T_i} and A_T, i.e.

$$b_i = \frac{A_{T_i}}{A_T} \qquad i = 1, 2, 3$$

Another important observation is that subregions of $\mathbb{R}^2$ relative to T are defined by the signs of the barycentric coordinates, see Figure 5. This is particularly useful for searching the triangle that contains a point v in a planar triangulation: Given a seed triangle T, the barycentric coordinates of v relative to T are calculated. If all b_i's are positive, T contains the point. Otherwise the next triangle in the direction indicated by the signs is picked, and the search continues until the triangle that contains v has been found.

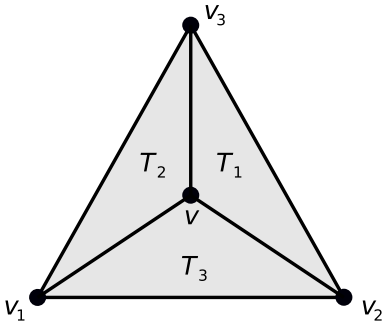


Figure 4: Areal coordinates

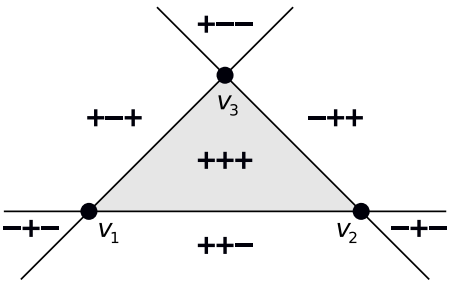


Figure 5: Subregions of $\mathbb{R}^2$

Delaunay triangulation and Voronoi tessellation

Both the concept of the Delaunay triangulation for a set of points $P \in \mathbb{R}^2$ and the corresponding Voronoi tessellation of the region are of particular importance in the field of geometric computing. The Delaunay triangulation maximizes the minimum angle of all triangles in the triangulation, thus creates the "best" triangulation for a fixed set of points and tend to avoid skinny triangles.

Definition 2.9 *The* circumcircle *of a triangle is the circle that contains all three points of the triangle. The radius of this circle is called* circumradius *and the circle is centered at the* circumcenter.

The circumcircle is uniquely defined for every triangle. In case of a degenerate triangle where all points are near a straight line, the circumradius tends to infinity.

Definition 2.10 *A triangulation is a* Delaunay triangulation *if the Delaunay condition is met, that is if no point in P is inside the circumcircle of any triangle in the triangulation.*

Points directly on the circumcircle are explicitly allowed; this happens for example if four vertices are in rectangular position. For a given triangulation the Delaunay criterion can be restored by *flipping* edges of the triangulation (see Figure 6). An efficient method is to flip an edge if the sum of the angles opposite to the edge are greater than $180°$. With this techniques it's not even necessary to explicitly calculate the circumcircles.

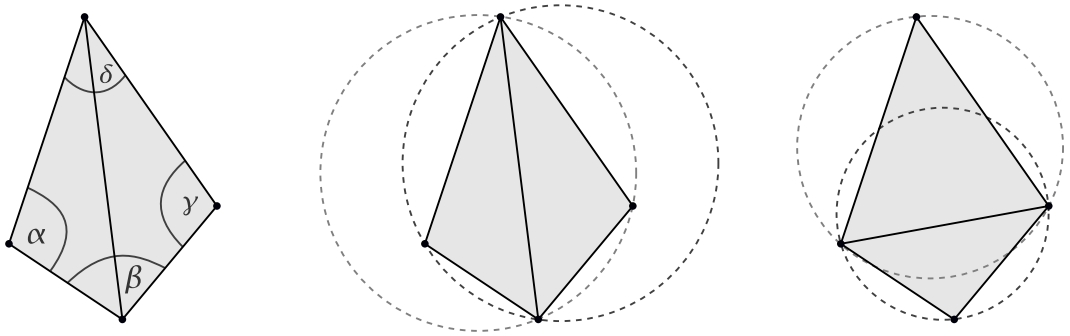


Figure 6: The sum of angles α and γ is greater than $180°$ and the circumcircle criterion is not met. An edge-flip restores the Delaunay criterion and improves the minimum-angle property.

The original definition of the Delaunay criterion is for the two-dimensional space only, but it is also possible to extend the definition to higher dimensions.

Triangulations are a special kind of *decomposition* of the two-dimensional space into triangular regions and a boundary. Another very important decomposition is the Voronoi diagram.

Definition 2.11 *A* Voronoi diagram, *also called a* Voronoi tessellation, Voronoi decomposition *or* Dirichlet tessellation, *is a decomposition of a metric space determined by distances to a discrete set of points in the space. Each point, called a* Voronoi site, *is located inside its* Voronoi cell, *and all positions inside a cell are closer to its Voronoi site than to any other site.*

The Voronoi diagram and the Delaunay triangulation are closely related (the graph of the Delaunay triangulation is dual to the graph of the Voronoi tessellation). The Delaunay triangulation can be derived by connecting the sites with adjacent Voronoi cells. In the other direction, each Voronoi cell can be constructed by connecting the circumcenters of the triangles adjacent to the point. Voronoi cells are always convex polygons; see [Aur91] for an extensive survey of Voronoi diagrams.

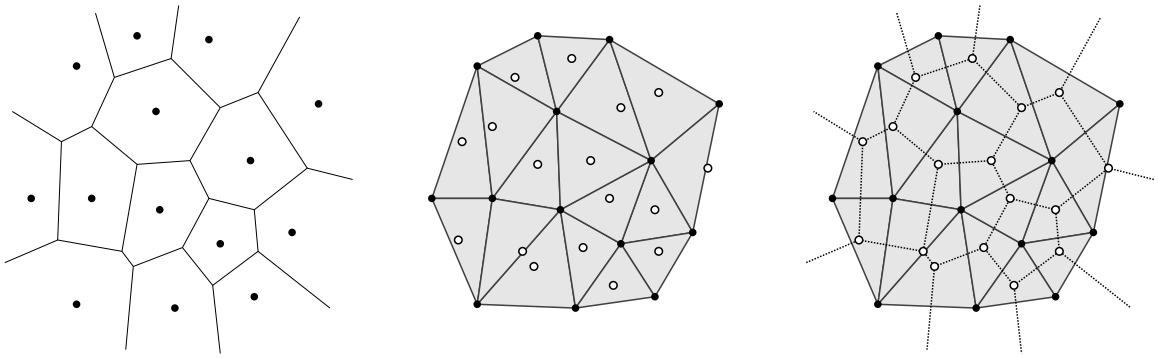


Figure 7: The Voronoi diagram for a set of points (left), a corresponding Delaunay triangulation with the circumcenters of the triangles (middle), and a combination of both (right). Connecting the sites of adjacent cells reveals the Delaunay triangulation. The circumcenters of a triangle fan are vertices of the Voronoi polygon.

2.2 Generation of Triangle Meshes

The generation of triangle meshes can be roughly divided into modelling the three dimensional surface with CAD-like applications and scanning a real-world object using dedicated devices. While the former approach is able to create artificial and artistic surfaces directly in the digital domain, production is time-consuming and subject to professionals. However, generating a triangle mesh is typically simply a matter of exporting the surface. The latter approach is commonly referred to as the *3D model acquisition pipeline* [BR02] and a lot of work has been published in this area. The process of digitizing a real-world shape starts with the scanning device. Conventional methods to record three dimensional data are *laser scanning* [LPC+00], projection of *structured light* [SS03, ZCS02, JBS98] and *stereoscopic techniques* [SS02].

Laser scanning exploits the fact that light travels at constant speed; measuring the time between irradiation and visibility of the reflection yields the distance to the object

(*time-of-flight technique*). The *triangulation method* determines the angle between the incoming laser beam and the reflection using a triangular setup of laser beam, laser point and camera. Laser scanning produces highly accurate point set surfaces, also with color and texture, but suffers from long scanning times.

Structured light makes use of light patterns that are projected on the subject to be recorded. These projected patterns are photographed and analyzed, the distortions of the light patterns give information about distances to the object. This approach is particularly fast because the visible part of the object is recorded with a single photograph. The individual scans are often referred to as 2.5D scans, because depth information is acquired from a single, fixed viewport. Depending on the complexity of the object, several recordings from different viewing directions have to be taken to completely cover the object, which involves registration of the resulting set of 2.5D depth images.

The stereoscopic technique is passive, and no active scanning or light irradiation is performed. Stereoscopic photographs of the object from several viewing directions are taken, and depth information can be derived from the spatially varying images. This technique is inspired and based on the apparatus of the human vision. Similar to structured light, the technique is 2.5D and several sets of stereo data are required to fully reconstruct the object. However, the time to process the images to produce 3D data is quite long and might be a problem for some applications.

A conceptually similar approach is to use Multi-View Stereo (MVS) methods [ES04, GCS06] to derive depth information from usual photographs of the object. These techniques have recently been extended to produce reasonable results on Community Photo Collections for popular public sites [GSC+07].

While laser scanning immediately results in a three dimensional point set, the use of structured light, stereoscopic techniques or MVS produce depth images for each scan as a premature artifact. These depth images need to be registered [CM91] (a transformation between pairs of images to combine all views into a common space). This is often a semi-automatic process, but can be omitted if position and direction of the sensors is known. Although sufficiently dense point sets can directly be visualized [ABCO+01, AK04, WK04], the goal of *scan integration* is to reconstruct the surface geometry from the available data [BV91]. This thesis will focus on triangle meshes for the geometry; they provide the connectivity information essential for many algorithms that operate on the surface, and nowadays graphics hardware is especially optimized for rendering triangles. Scan integration is a difficult problem because the input data, i.e. the point set surface or the depth images, may be contaminated with noise. In case of a point set, the available sampling may not even be sufficient for a proper reconstruction (too high ε-sampling).

There are several approaches to extract a triangulation from a point set or directly from depth images, some of them are reviewed in Section 3. The aim of these techniques is to create a "good" piecewise linear approximation of the surface $\mathcal{S}$. Depending on the context, there are diverse demands on the shape of the elements and the sampling of the surface. These quality criteria are discussed next.

2.3 Quality Criteria of Meshes

The goal of surface reconstruction and triangulation is to create a partitioning of the domain, where the elements or the whole approximation is subordinate to some quality characteristics. The discipline of surface remeshing aims at improving the triangle mesh quality in terms of vertex sampling, regularity and triangle quality. Unfortunately, there is no general quantity to measure the quality of a specific approximation and different applications have different demands. This section gives a brief overview of some practical quality criteria for triangle meshes.

Amount of Triangles

Maybe one of the most obvious quality criteria is the amount of triangles in the mesh. For example a triangle mesh with only several hundred triangles cannot, in general, approximate a complex, three-dimensional shape accurately. A large amount of triangles suggest a higher quality, at least in the sense of high fidelity to the original surface. However, the amount of triangles alone can impossibly make reliable statements about quality. The mesh may have inappropriate *vertex distribution*, i.e. wasting a lot of triangles in nearly planar regions.

Vertex Distribution

The vertex distribution is a very important criterion for most applications. For simulations, a *uniform vertex distribution* is often required, which results in overall similar-sized triangles. On the other hand it might be desirable to represent the surface as accurate as possible with as few as possible triangles. This is achieved by placing more samples in areas of high curvature, i.e. the sampling is *curvature adapted*. For nearly planar regions, only a few triangles are necessary to approximate the surface. Considerably more triangles are needed to represent details on the surface, see Figure 8 for a simple example in 2D.

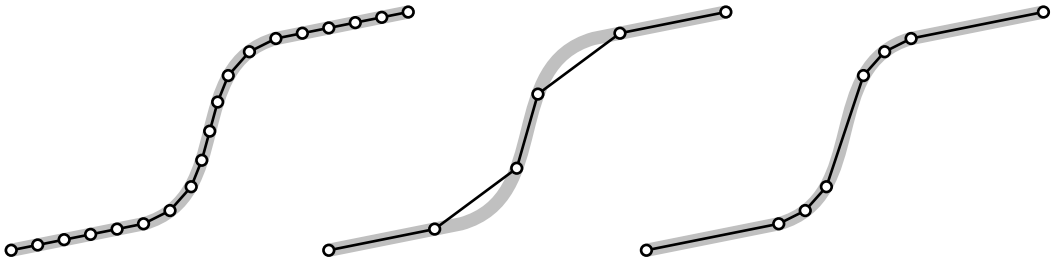


Figure 8: A uniform sampling with too many vertices (left), a uniform sampling with few vertices results in a bad approximation (middle), and a curvature-adapted sampling with both, few vertices and a good approximation (right).

Error of the Approximation

A strong criterion for the quality of a mesh is its fidelity to the original surface. As seen in Figure 8, the adaptive vertex sampling considerably reduces the error with a fixed vertex budget. In general, the deviation between the approximation $\mathcal{T}$ and the surface $\mathcal{S}$ cannot be determined, because the real surface is not known. It is, however, possible to measure the distance between two triangular meshes (for example using *Metro* [CRS98]), or the distance between a triangulation and the corresponding point set surface $\mathcal{P}$. Two common quantities are the mean error e_{mean} and the maximum error e_{max}.

$$e_{mean} \;=\; \frac{1}{n}\sum_i \min_{t\in\mathcal{T}} \|t - p_i\| \qquad\qquad e_{max} \;=\; \max\left\{\min_{t\in\mathcal{T}} \|t - p_i\|\right\}_i$$

The error e_{max} is often referred to as *Hausdorff distance*. Obviously, if all the input points of $\mathcal{P}$ are used to create the triangles, e_{mean} and e_{max} are zero, and the quantities are useless.

Shape of the Triangles

Important types of triangles include

- *equilateral* or *equiangular* triangles, all sides and angles are equal,
- *right triangles*, one angle is exactly $90°$,
- *acute triangles*, all angles are smaller than $90°$,
- *obtuse triangles*, one angle is larger than $90°$ and
- *degenerated triangles*, with at least one very small angle.

A degenerated triangle (or *skinny triangle*) can be classified into two different types, *needles* and *caps* [BK01], see Figure 9. It is crucial to avoid all types of degenerated triangles because they can cause numeric problems and instabilities in algorithms. However, to create a visually pleasing and numerically stable triangulation, the goal is to generate a mesh free from degenerated but nearly equilateral triangles. It is therefore important to formalize a measure for the quality of a triangle. A good quantity is the ratio between the incircle radius r_i and the circumcircle radius r_c.

$$\frac{r_i}{r_c} \;=\; 4\sin\frac{\alpha}{2}\sin\frac{\beta}{2}\sin\frac{\gamma}{2}$$

For an equilateral triangle, this ratio yields 0.5; this is a global maximum for this function. For increasing degeneracy of the triangle the function approaches zero. Similar to the approximation error, a mean and minimum ratio r_{mean} and r_{min} can be calculated.

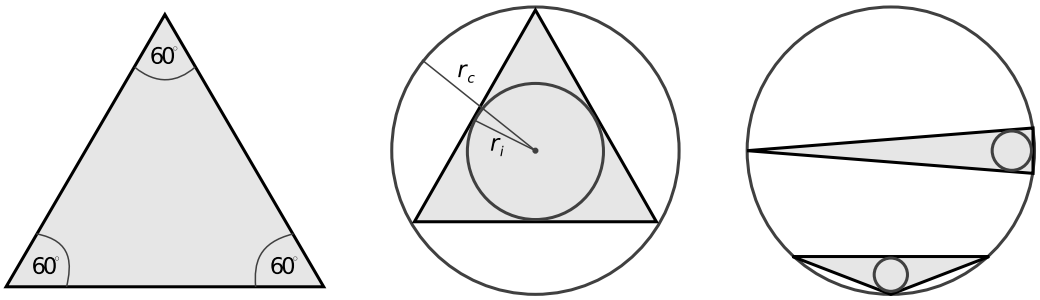


Figure 9: An equilateral triangle (left) has best ratio of inradius r_i and circumradius r_c (middle). Both types of degenerated triangles, caps (bottom right) and needles (top right) have a smaller, thus worse ratio.

Degree of the Vertices

The *degree of a vertex*, or sometimes *valence of a vertex*, is the amount of edges that join that vertex. Deduced from the fact that an equiangular triangle has $60°$ angles, the degree of a vertex should be $\frac{360°}{60°} = 6$. A vertex with a degree of 6 is called a *regular vertex*. In fact, it is possible to triangulate some shapes solely with regular vertices. However, for surfaces embedded in the three dimensional space, this is generally not useful because of distortion caused by the constrains. But certainly, the amount of irregular vertices should be minimized, this is especially important for the stability of Finite Element Method (FEM) simulations.

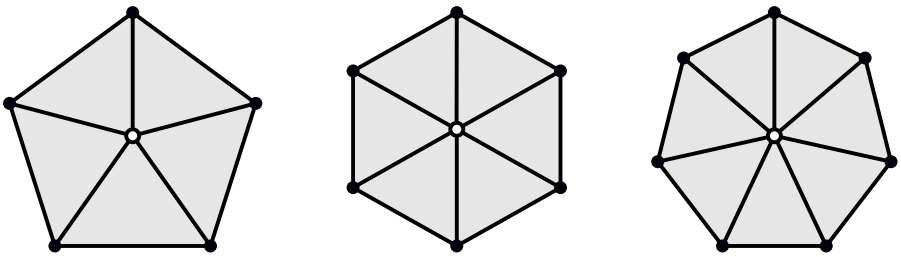


Figure 10: A vertex with degree 5 (left), with degree 6 (middle) and degree 7 (right).

High-Quality Meshes

The fundamentals given in this sections are now concluded with a discussion about what high quality meshes are. Surely, the meaning of high quality is related to the specific scope of application. For example if fidelity is a major issue, the meshes generated by CAD applications with a lot of skinny and degenerated triangles are probably very well suited to accurately approximate the surface. These meshes are also good for the matter of rendering, but simulations or further processing of these meshes is almost impossible without remeshing or CAD repair [SWC00, MW99]. Geometric processing

or numerical computations on the mesh require a fairly regular structure, not only in terms of connectivity but also geometry of the elements.

In this thesis, the notion of quality refers to a combination of the properties mentioned in this section; a mesh is of high quality if several of the above criteria are fulfilled to some extend. It is desirable to have regular vertices, i.e. the connectivity of the mesh should be almost regular, and near equilateral triangles to obtain a regular geometry of the elements. A proper distribution of the vertices on the surface is necessary in order to approximate the surface as good as possible with a fixed amount of vertices, and this is closely related to the error of the approximation. A uniform vertex distribution is often a worthwhile property for simulation and computations on the mesh, while adaptive distributions (typically adapted to the surface curvature) are favorable when reducing the error to the original mesh. Thus the amount of triangles and the vertex distribution indirectly influence the error of the approximation.

The downside of advocating this notion of quality, especially regularity in terms of connectivity and element shape, is that resulting meshes are often not optimal in the amount of elements: The same geometry may be approximated with fewer elements and higher fidelity at the cost of adverse triangles. Quite honestly, another disadvantage is the rendering quality in curved regions. The isotropic placement of the samples prevents the triangles from being aligned in any particular direction, for example along the directions of the principal curvature, which may produce unpleasant shading artifacts as shown in Figure 11. Anisotropic surface discretizations are seriously better suited for the purpose of visualization [ACSD+03].

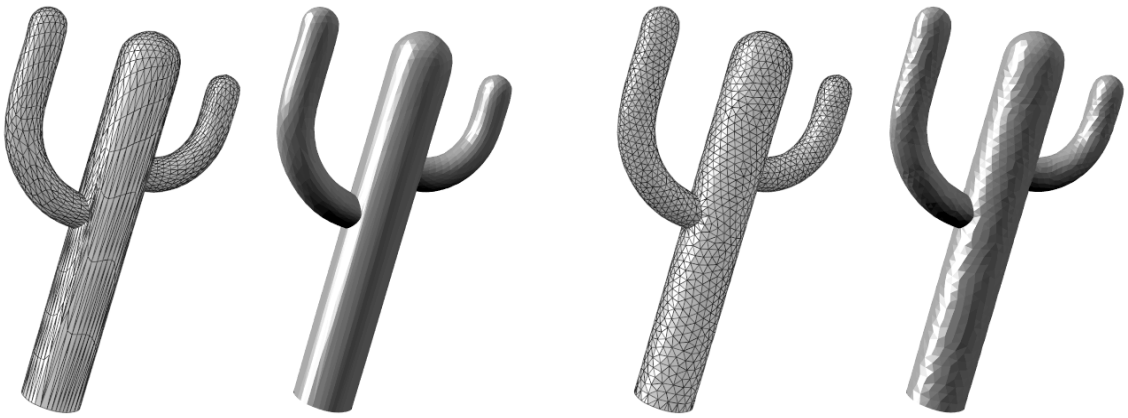


Figure 11: The original triangulation and a shaded rendering with about 3k triangles (left), and the isotropic remesh with about 5k triangles (right). The remesh suffers from shading artifacts caused by the isotropic placement of the samples.

3 Surface Reconstruction and Remeshing

This section deals with related work in the field of surface reconstruction from point sets and depth images, as well as with remeshing of already existing triangular meshes.

Reconstruction of surfaces is also referred to as *scan integration*, i.e. the generation of a piecewise linear surface representation from the scan data. Scan integration is difficult in general, because the data points are typically noisy and may contain outliers. The reconstruction has to compensate for these errors. Literature in this field is vast and this section gives a brief overview of important algorithms. The interested reader is pointed to [BV91, MM97, BR02].

The problem of scan integration can be divided into methods that operate on point set surfaces and directly on depth images for reconstruction. While the former representation typically only provides point positions and sometimes point normals, depth images implicate more information. Accurate surface normals, connectivity and a point set can easily be deduced from the data.

3.1 Delaunay-Based Reconstruction Methods

A certain kind of reconstruction algorithms rely on the Delaunay triangulation or the dual Voronoi tessellation to extract the elements of the triangulation. In the following, a few classical algorithms that directly operate on the point set data are presented. These algorithms typically come with reconstruction guarantees if certain prerequisites are met. Although these guarantees are nice to have, the assumptions made on the input are very tight, and practical examples rarely meet these requirements.

Alpha Shapes

In 1983 Edelsbrunner et al. introduced a mathematical definition of shape in two dimensions. Edelsbrunner and Mücke extended the idea to three dimensions in [EM94], called Alpha Shapes (or α-shapes). They basically provide a mathematically formal and concrete definition of *shape* for a finite set of points $\mathcal{P}$ in space. Their method is a generalization of the convex hull of the point set $\mathcal{P}$ with respect to a parameter α, where the convex hull is computed if α is set to ∞.

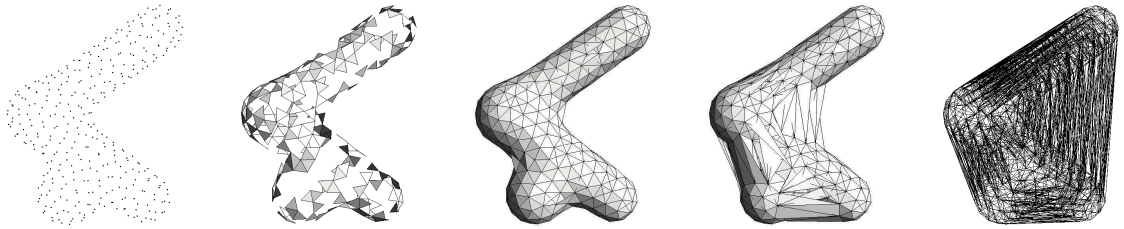


Figure 12: Different α-shapes from zero (left) to infinity (right), taken from [TC98].

Alpha Shapes is one of the oldest surface reconstruction methods based on the Delaunay triangulation. At the intuitive level, [EM94] explains the α-Shapes as follows:

> For $\alpha = \infty$, the α-shape is identical to the convex hull of $\mathcal{P}$ (the point set). However, as α decreases, the α-shape shrinks by gradually developing cavities. Think of $\mathbb{R}^3$ Styrofoam and the points of $\mathcal{P}$ made of more solid material, such as rock. Now imagine a spherical eraser with radius α. It is omnipresent in the sense that it carves out Styrofoam at all positions where it does not enclose any of the sprinkled rocks, that is, points of $\mathcal{P}$. The resulting object will be called the α-hull. To make things more feasible we straighten the surface of the object by substituting straight edges for the circular ones and triangles for the spherical caps. The obtained object is the α-shape of $\mathcal{P}$.

Although α-shapes are an elegant mathematical concept, the algorithm lack practical applicability. Even simple shapes are often not properly reconstructed, triangles may overlap and points that don't belong to each other may be connected. To gain more control of the situation, several additions and improvements have been proposed, not least because α-shapes provide theoretical guarantees for provably correct reconstruction, which other reconstruction methods often lack.

In [BB97] some practical aspects like elimination of dangling and isolated faces, edges and points are discussed, and a regularized version of α-shapes is presented. Other examples include calculation of an optimal α parameter, weighted α-shapes and generalization to higher dimensions, see [Ede92] for details. Conformal Alpha Shapes [CGPZ05] have been established which uses a locally parameterized scale parameter instead of the globally defined α in the original method. This is especially useful for reconstruction of non-uniformly sampled surfaces. Similarly, [TC98] describes a density-scaling technique to adjust α depending on the density of points in a region. Additionally, they modulate the form of the α-ball based on point normals to further improve the method, yielding *anisotropic, density-scaled alpha shapes*.

The Crust

The Crust algorithm [ABK98] is a reconstruction method based on Voronoi filtering by Nina Amenta et al. Similarly to alpha shapes, the algorithm creates a piecewise linear interpolation of the input point set. The authors claim that they developed the first algorithm with provable guarantees, and they present their theoretical results and proofs in a companion paper [AB98].

In particular, they prove that the reconstruction of a *well-sampled* set of points is topologically valid and convergent to the original surface. For the purpose of defining "*well-sampled*" they use the notion of the *local feature size*, which involves the medial axis of the surface and makes the algorithm sensitive to the local geometry. This means that, unlike other algorithms for this problem, the new technique allows highly

non-uniform samplings, i.e. dense samplings in detailed areas and yet sparse samplings in featureless areas. The sample spacing required to correctly reconstruct the surface is proportional to the distance to the medial axis. As shown in [ABK98], the algorithm performs good up to an ε-sampling of 0.5, however, correct reconstruction is only guaranteed for $\varepsilon = 0.06$.

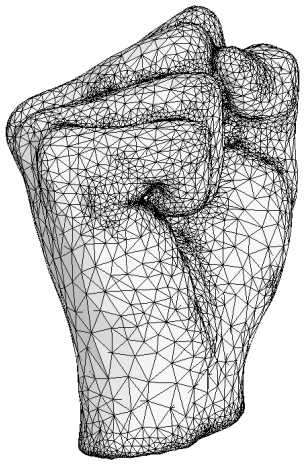


Figure 13: Example of the curst algorithm, taken from [ABK98].

In a first step, the crust algorithm creates a three dimensional Voronoi diagram for all points in $\mathcal{P}$. For each of the sites two poles are calculated, that is the two vertices of the Voronoi cell with maximum distance. A Delaunay triangulation is created from the union of $\mathcal{P}$ and the poles. The final triangulation is extracted by keeping the triangles for which all three vertices are sample points. This triangulation, however, is not necessarily a manifold and may contain overlapping triangles, see Figure 13. The authors claim that this is visually acceptable.

Reconstruction using Co-Cones

In [ACDL00], Amenta et al. present a variation of the Crust algorithm for surface reconstruction. They simplified both, the algorithm and the proofs for the crust, additionally they provide a new proof and show that the modified crust algorithm is homeomorphic to the input surface. In contrast to [ABK98], which involves two passes for calculating the Delaunay triangulation and two passes for post processing, the new algorithm requires only a single Delaunay algorithm for reconstruction.

Similar to the crust, a normal for each sample point is estimated. For an angle θ, which is a user parameter, a *complemented cone* (co-cone) is defined, and a set of Voronoi edges that intersect the co-cone is determined for each sample. The dual triangulation of the calculated Voronoi edges constitute the candidate triangles (which are Delaunay). The final triangulation is extracted with a technique similar to the last step of the Crust algorithm. See [ACDL00] for a more detailed explanation.

The co-cone algorithm as presented in [ACDL00] does not properly reconstruct sharp features and boundaries, and invalid triangulations may be produced in these areas. Several contributions regarding the co-cone algorithm have been proposed, mainly by the co-author of [ACDL00], Tamal K. Dey. He investigated in detecting undersampled areas to approximate the boundaries of the point set in [DG01]. The well-sampled surface is then reconstructed using the co-cone algorithm. A major drawback of Delaunay based methods is the reconstruction time and memory demands. Additionally they cannot properly handle large amounts of data. In [DGH01], Dey et al. presented an extension to co-cone that is able to handle large data sets with millions of points by taking a divide-and-conquer approach that partitions the sample points into smaller clusters. Another typical artifact in reconstructed surfaces is the presence of holes in areas where the sampling condition is not met. In [DG03], the *Tight Co-cone* algorithm is presented that takes an initial mesh, fills up holes and resolves anomalies near sharp edges and corners to produce a water-tight surface. To accommodate for noise in the input point set, Dey proposed a technique for surface reconstruction from noisy samples [DG04]. The technique requires a dense point set of a surface without boundaries. Reasonable samples are interpolated while outliers are deleted.

Conclusion

The presented methods directly operate on the point data and the goal is to create a Delaunay triangulation that contains all points of $\mathcal{P}$. Although these techniques are able to process sparse point data, the algorithms are sensitive to noise, and outliers are directly visible in the resulting surface if no special care is taken (cf. [DG04]). Furthermore, the amount of triangles cannot be controlled and is related to the amount of points in the input data. The shape of the triangles directly depends on the location of the points in space, and unfavorable point distributions cause badly shaped triangles and highly irregular vertices.

3.2 Implicit Reconstruction Methods

As concluded in the previous section, there are several constraints when using the input points directly for surface reconstruction. A solution is to switch to a different surface representation for further processing. Implicit functions have been successfully applied in the past. In an implicit function, the dependent variable is not given *explicitly* in terms of some independent variable like in $y = f(x)$, where x is independent and y depends on x, but rather given as the solution of some equation $f(x, y) = 0$. In this example, all values (x, y) for which $f(x, y) = 0$ are called the *zero-set* $Z(f)$ of f.

Signed Distance Function for Reconstruction

In [HDD+92], Hoppe et al. advocates the use of the Signed Distance Function (SDF) for creating surface approximations from unorganized points. This is fundamentally different from the Delaunay based methods because the surface is not interpolated but approximated, i.e. the output vertices are not part of the input point set. The SDF is a natural choice for an implicit function: The zero-set of the function consists of points directly on the surface, the SDF is negative for all points in the interior of the surface, and positive everywhere else. In contrast, the *Unsigned* Distance Function does not distinguish between the interior and the exterior. Distance functions are typically represented in regular, three dimensional grids, where each grid point (also called *voxel*) contains the shortest distance to the surface. This is of course a discretized representation and the dimension of the grid is crucial. A small grid may miss details of the surface whereas a large grid takes a considerable amount of memory.

Given such a SDF, the resulting surface can easily be extracted, for example using the Marching Cubes algorithm [LC87]. The basic principle of the algorithm is as follows: The algorithm iterates over all *cells* in the grid and classifies each into one of 256 possible configurations. The cells that contain a subset of the zero-set of the SDF, i.e. the cells where at least one voxel has a different sign than the other voxels, are triangulated, see Figure 14. Several extensions to the Marching Cubes algorithm exist, for example feature-sensitive iso-surface extraction [KBSS01].

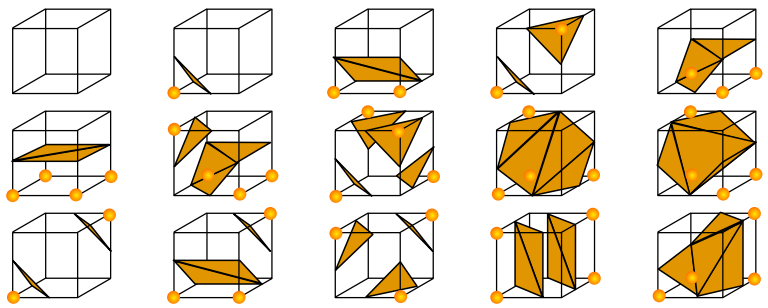


Figure 14: The 15 unique cube configurations for the marching cubes algorithm. All 256 configurations can be obtained by reflections and symmetrical rotations.

One of the main challenges is to create the SDF from the given set of points. Hoppe associates oriented planes for each data point which serve as local linear approximations for the surface. While the tangent planes can easily be constructed, maintaining a consistent orientation of the planes is more difficult. The alignment of the plane is estimated by using a least squares fitting technique (see Appendix A), using the k nearest neighbors, i.e. the k-neighborhood of the data point. The orientation of the plane is chosen in a way such that the planes of *nearby* points have similar orientation, i.e. the scalar product of the plane normals is positive.

In the next step, the SDF is calculated by finding, for each voxel, the tangent plane which is closest to the voxel and determining the distance to that plane. Depending on the plane normal, the distance is multiplied by ± 1 to flag the voxel as either interior or exterior.

Bernstein-Bézier-Patches for Reconstruction

A conceptually different and more complex approach is used by Bajaj et al. in [BBX95], but they also use the signed distance function to create an implicit representation. In the first step, an approximate, initial surface is calculated using α-solids [EM94], and the distance to that surface is used to initialize the SDF. The second step decomposes space into tetrahedra using an incremental approach, starting with a single tetrahedron that contains the whole data set. For tetrahedra traversed by the initial surface, a new surface approximation is calculated using trivariate Bernstein-Bézier polynomials. The approximation error to the data points is calculated and a bad approximation is resolved by refining the affected tetrahedron until an acceptable accuracy is achieved. For this purpose, a *Delaunay tetrahedralization* is used, which is a generalization of the Delaunay triangulation to three dimensions.

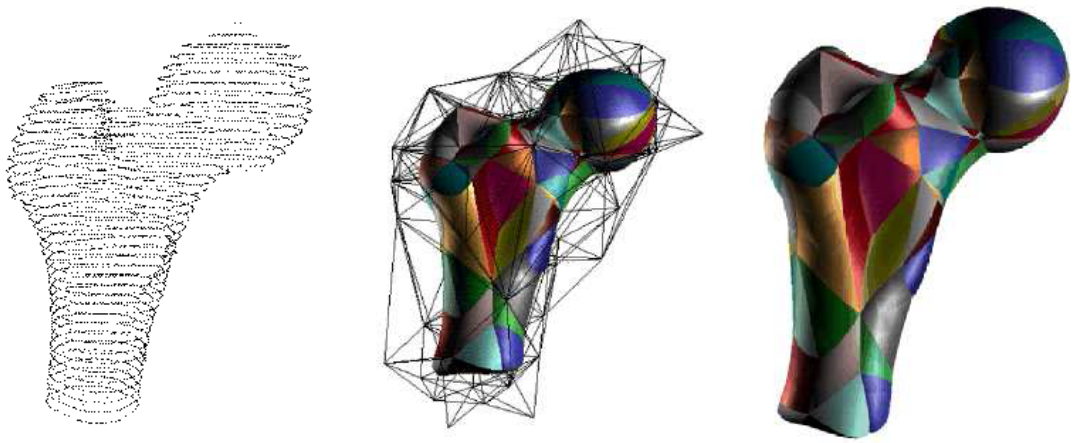


Figure 15: CT scan of a human femur. Point set surface (left), tetrahedral decomposition (middle) and resulting surface with colored patches (right), taken from [BBX95].

After this iteration, a C^0-smooth surface composed of trivariate Bernstein-Bézier patches is obtained. To compute a C^1-smooth surface, a Clough-Tocher subdivision scheme is applied. The method is especially well suited for reconstruction of smooth and curved surfaces. Although the surface is C^1 continuous, seams and bumps may be visible in the resulting surface, see Figure 15.

In a later work Bernardini et al. proposed an extended method [BBCS99] which uses a simplification step on the α-solid reconstruction. Polynomial BB-patches are constructed directly on the triangles of the simplified model. Even reconstruction of sharp features can be realized with this method.

Direct extraction from Range Images

Another volumetric approach is presented by Curless and Levoy in [CL96], which is specific in a way that it directly operates on several registered range images. The method is robust in the presence of outliers, able to fill gaps in the resulting reconstruction and works incremental, i.e. the method can be used in an interactive scanning session to incrementally build the reconstruction.

For each range image i, a signed distance function $d_i(x)$ and a weighting function $w_i(x)$ is computed, and a cumulative SDF $D(x)$ and weighting function $W(x)$ is composed from the functions on the range images. One important point is that $D(x)$ and $W(x)$ are incrementally calculated, i.e. updated for each new range image that is added. Similar to the method of Hoppe, the function is represented in a discrete voxel grid. The weighting function is used to express confidence or certainty of the range data, e.g. the boundaries in the range images typically need down-weighting because of increasing uncertainty. The cumulated SDF is combined with respect to the weighting function in a straightforward way using a simple weighted combination of the range values.

To calculate the SDF for a single range image, the domain is tessellated first by constructing triangles from the image lattice. Triangles with edge lengths that exceed a certain threshold are discarded. These triangles arise from *step discontinuities*, i.e. *cliffs* in the range images. These cliffs are indicators for hidden, uncaptured surface parts. Once the range image has been converted to a triangle mesh, the voxel grid is populated using a simple ray casting technique: A ray is emitted from the sensor through the voxels near the surface that intersects the mesh. The intersection point is used to compute the distances and to estimate the weight for the voxels.

In the next step, holes in the voxel grid are identified and filled by using a classification on the voxels into *empty*, *near surface* and *unseen*. This classification is referred to as *space carving*, by following the lines of sight back from the observed surface and marking the voxels as empty. Finally, the iso-surface is extracted at the zero-crossing of the SDF. The implementation is fast, robust and capable of handling large data sets.

Poisson-based Reconstruction

Recently in 2006, Kazhdan et al. presented a novel approach for reconstruction in [KBH06] by formulating the problem as a spatial Poisson equation, which reduces to solving a well-conditioned, sparse linear system of equations. Unlike other approaches, the method requires a set of *oriented* points, i.e. a surface normal has to be provided for each sample. If normals are not provided in the input data, they can be estimated from the positions of neighboring samples, similar to the method Hoppe uses in [HDD+92]. Poisson reconstruction is a global approach and considers all points at once. Poisson systems are robust even in the presence of noise, thus the method is able to produce very smooth surfaces even for imperfect data. Consequently, the method is less suited for reconstruction of sharp features or incomplete data sets with holes.

Figure 16: Poisson surface reconstruction of the Stanford Dragon model with an octree depth of 6, 8 and 10 respectively, images taken from [KBH06].

The method works by defining a vector field as linear sums of hierarchical functions, a set of high-resolution functions near the surface and coarser resolutions away from it. The Poisson equation is solved and the iso-surface of the resulting indicator function is extracted. Since an accurate representation of the implicit function is only necessary near the surface, an adaptive octree is used to represent the discretized version of the function and to solve the Poisson system. Finer nodes of the octree are associated with higher-frequency functions, thus reconstruction becomes more precise for higher tree depth, see Figure 16. In contrast to a previous method based on the Fast Fourier Transform (FFT) in [Kaz05], which does not use an adaptive grid and has very high memory and runtime demands, the Poisson based method has linear complexity in both, time and space.

The Poisson surface reconstruction has recently been extended with multilevel streaming for Out-of-Core (OOC) surface reconstruction [BKBH07] for large models that don't fit into memory. In this work a streaming framework has been employed for efficient traversal of the octree. The solution of the involved linear system is obtained in a single iteration of a cascadic multigrid solver. Thus evaluation is possible within a single pass over the data.

3.3 Greedy Sample Placement Methods

The methods that are discussed in this sections are well suited for both, reconstruction and remeshing of surfaces. The intention is to place new samples on the surface, i.e. to resample the domain. The placement of the new samples is done in a *greedy* way, that means that one sample is placed after the other and yet placed samples are neither removed nor relocated, they are fixed after insertion. For reconstruction of point set surfaces, samples may be placed on local approximations of the surface, for example using the *Moving Least Squares* (MLS) method [Lev98]. Remeshing is done by placing new samples on the elements of the original mesh.

Farthest Point Sampling

One paradigm for the greedy placement of samples is farthest point sampling (FPS). The central idea is to insert one point at a time as far as possible away from already placed samples. This paradigm is used by Eldar et al. [ELPZ97] for progressive image sampling in two dimensions. The main advantages are the possibility to sample a domain to an exact vertex budget, always retaining uniformity and isotropy while increasing the density of the samples. To find the center of the biggest void, Eldar et al. rely on the Voronoi tessellation of the domain.

The technique of maintaining a Delaunay triangulation while inserting points is called *incremental Delaunay refinement* [Rup95], which is an essential element of FPS. Chew used this technique to create high quality, curvature adapted surface meshes on curved domains with boundaries [Che93]. Miller developed a time-efficient Delaunay refinement algorithm in [Mil04] that runs in sub-quadratic time.

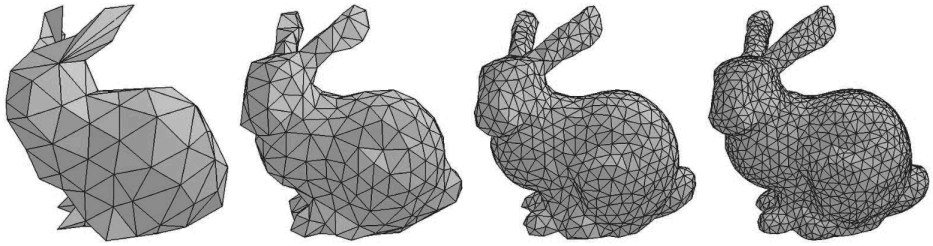


Figure 17: Geodesic farthest point remeshing with 100, 300, 800 and 1500 samples, taken from [PC06].

FPS has been developed for three dimensional surfaces using geodesic distances. The Fast Marching Method [Set99] developed by Sethian offers an efficient way to calculate geodesic distances on surfaces, and it has been extended for the purpose of adaptive, farthest point sampling on triangular meshes, called *FastFPS* [MD03a]. In a subsequent paper [MD03b] the idea is extended to directly operate on implicit surfaces and point sets for surface reconstruction. Similarly to FastFPS, Peyré and Cohen present an isotropic remeshing algorithm using FPS and geodesic Voronoi diagrams

based on the Fast Marching Method, yet not able preserve features in the mesh [PC06], see Figure 17.

Yet another algorithm for provably good sampling of implicit surfaces is given in [BO03]. Points are progressively added while restoring the Delaunay criterion, to produce good uniform or curvature adapted triangulations, even with boundaries. The authors make use of *restricted*, or *constrained Delaunay triangulations* to prove that the algorithm creates triangulations with the same topology, a "good" sampling and bounded aspect ratios of the triangles.

Advancing Front Algorithms

The Advancing Front approach, or sometimes called *Marching Front*, is a relatively young class of algorithms, and it first appeared somewhat around 1997. Development of the new technique was driven to acquire high quality triangle meshes directly from implicit surfaces. Typically, these implicit surfaces are triangulated using the Marching Cubes algorithm, which is known to produce bad, ill-shaped triangles, arbitrary samplings that don't belong to a sensible measure, e.g. the surface curvature, and approximate the surface in a way that needs improvement. The Advancing Front technique, however, is not limited to implicit surfaces. As mentioned by Hartmann in [Har98], the method is rather suited for all kind of representations. Advancing Front algorithms have been extended to operate directly on point set surfaces for reconstruction, or on already existing triangle meshes for remeshing.

The basic principle of the Advancing Front method is as follows: A seed edge is placed on the surface. A new vertex is predicted to forge a perfectly shaped triangle together with the two existing vertices. The predicted vertex is projected on the surface, completing the first triangle. All three edges of the triangle are the *active front*, partitioning the surface into a triangulated region and a yet undeveloped region. The algorithm proceeds with triangle growing, that is, new points are predicted, projected on the surface and new triangles are created, thus expanding the front until the whole surface is triangulated. During triangle growing, two topological events may occur: If a front encounters itself, the front is split into two fronts by building a bridge edge between two vertices on the front. If a front encounters another front, both fronts are merged, again by creating a bridge, see Figure 18.

The first generation of Advancing Front algorithms was developed to triangulate implicit surfaces as an alternative to the Marching Cubes algorithm. Hartmann created the foundation for the Advancing Front family in [Har98]. The algorithm creates nearly equilateral triangles and produces a good, uniform sampling. The size of the triangles is constant in that algorithm, and details in the surface may be lost if the triangle size is chosen too large.

The next generation of Advancing Front algorithms address the problem of constant triangle sizes. Karkanis and Stewart proposed an algorithm [KS01] to triangulate implicit surfaces using a curvature sensitive approach. With this approach the size of the triangles is changed according to the local curvature, creating smaller triangles

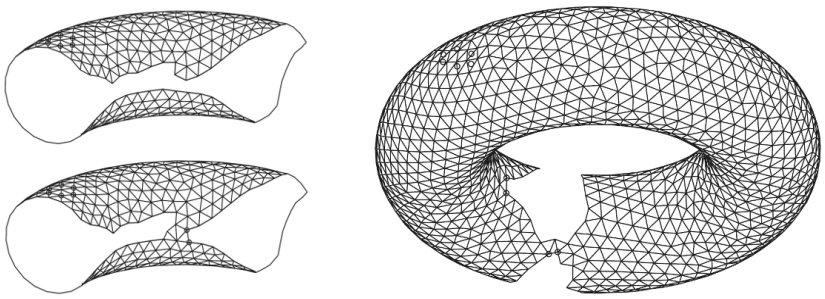
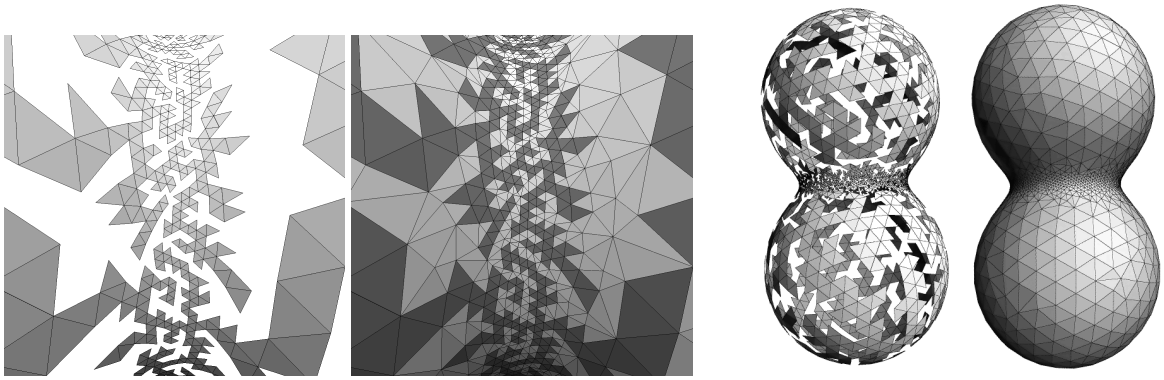


Figure 18: Topological events in Advancing Front: A front encounters itself and is split (left) and a front encounters another front, which are merged (right), taken from [Har98].

in more curved regions. The amount of adaptation is a user parameter that controls the ratio of triangle edge length and the local radius of curvature. The algorithm does not use front splits and merges. Instead, it operates in two phases. In the *growing phase*, the algorithm creates new triangles as long as fronts don't encroach each other. If fronts encroach the triangulation is stopped and gaps emerge. In the *filling phase*, the gaps are bridged using several heuristics. This is the flawed part in the algorithm because filling may introduce bad triangles.



(a) After growing phase (b) After filling phase (c) After growing (d) After filling

Figure 19: Growing and filling phase of the advancing front algorithm in [KS01].

Scheidegger et al. [SFS05] use a guidance field to gradually change the size of the triangles instead of just evaluating the *local* curvature. The algorithm directly operates on point data using Moving Least-Squares (MLS) [Lev98, Kol05] for the underlying surface representation. One of the strengths of MLS is the ability to handle fair amounts of noise in the input data, which makes the technique particularly suited for surface reconstruction from possibly noisy point set data. In a successive work, Schreiner, Scheidegger et al. present an improved algorithm [SSFS06]. In particular, they delay triangle output by keeping a small band of triangles before finalizing them, to improve the triangles with edge flips. The algorithm was also extended to operate on triangle

meshes for direct remeshing. They provide their results in a software package called *Afront*.

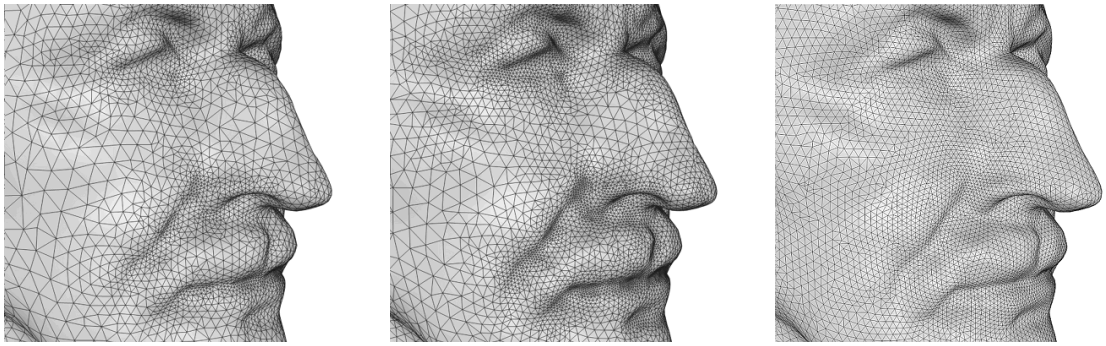


Figure 20: Triangulation results with various parameter settings from [SSFS06].

Although the use of MLS has shown success in triangulating point set surfaces, MLS is computationally expensive and complex, because it involves a non-linear optimization problem that can only be handled numerically. Solving that optimization problem is particularly hard. Even though Scheidegger et al. suggest an initial value for the optimization solver [SFS05], the method does not guarantee the global optimum. The polynomial approximation of the surface may fail due to missing connectivity information, and points beyond the vicinity of the surface are not guaranteed to be correctly projected. This makes advancing front algorithms particularly fragile, because a sole, failed projection likely messes the whole triangulation.

A new projection method for point set surfaces has been proposed in [KFU+09, Uhr07], which does not involve a non-linear or similar optimization problem. Unlike MLS, the technique uses bivariate polynomials of variable degree depending on the points to be approximated. The use of connectivity information prevents the technique from collecting nearby points in space with large Riemannian distance on the surface, and improves the approximation of the surface and the robustness of projections.

Although the new projection improves stability, Advancing Front algorithms have several serious drawbacks in practice. The guarantees of the methods only apply to the "growing phase" where new triangles have bounded ratio depending on the configuration of the guidance field. The worst triangles, however, are created when topological events occur, i.e. while bridging fronts. As already mentioned, the projection can fail, which typically destroys the reconstruction. Several experiments show that Advancing Front algorithms are particularly unstable for complex data sets or simply if the triangle size is chosen too large. Even in spite of failures, the computational complexity of sophisticated Advancing Front algorithms is often not acceptable. Although the amount of triangles can be roughly controlled, neither an exact nor an approximate vertex budget can be specified.

3.4 Relaxation-based Techniques

The relaxation-based methods significantly differ from the previously described techniques. Instead of directly operating on point set surfaces, relaxation-based methods operate on existing triangular meshes, and point sets need to be reconstructed in a pre-processing step. Thus, the techniques address remeshing of surfaces only.

Unlike Advancing Front or Farthest Point Sampling, the method does not create a new triangulation from scratch. Instead, a series of local mesh adaptation steps is performed to improve the sampling and the quality of the mesh elements. In a first step, the mesh complexity is adjusted using simplification or oversampling to achieve an exact vertex budget. In the second step, the vertices are relocated on the surface using a relaxation procedure to obtain a good sampling with high-quality triangles, which is the most delicate part of the algorithm.

Attraction-Repulsion

In an older work, Turk proposed an relaxation-based algorithm for re-tiling polygonal surfaces to automatically create several *Level of Detail* (LoD) representations of a mesh [Tur92]. Turk places a user-defined number of vertices on the input mesh. Since the initial placement of the vertices is more or less random, the vertices are re-arranged using a attraction-repulsion particle relaxation procedure to move a vertex as far as possible away from neighboring vertices. In the next step, a mutual tessellation that contains both, the original and newly placed vertices, is created. The input vertices are carefully replaced with the new vertices. Although this technique produces aesthetic results with a good vertex distribution, the method is poorly suited for models with well-defined corners and sharp edges.

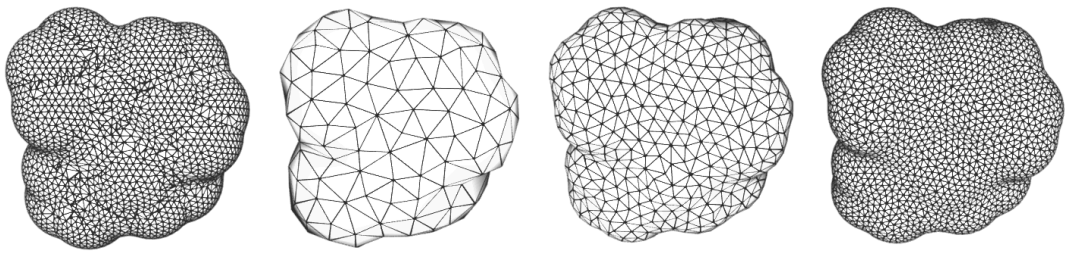


Figure 21: Attraction-Repulsion: Original molecular model (left) and several LoD re-tilings of the same object, taken from [Tur92].

Umbrella Operator

Vorsatz et al. use the umbrella operator [VRS03] as relaxation operator to locally reposition a vertex with respect to its direct neighbors (the 1-ring of the vertex to be repo-

sitioned). In particular they use a *weighted 3D Umbrella operator* $\mathcal{U}$:

$$\mathcal{U}(p_i) = p_i + \frac{1}{\sum_j \omega_{ij}} \sum_{j=0}^{n} \omega_{ij}(p_{ij} - p_i), \quad \omega_{ij} \geq 0$$

This operator basically moves a vertex p_i in the barycenter of the neighboring vertices. If the weights ω_{ij} are chosen to be uniform, i.e. $\omega_{ij} = 1$, a globally uniform distribution of triangles is generated in the simple case, and a locally uniform distribution (i.e. isotropic distribution) is achieved if weights ω_{ij} are chosen proportional to the area of the triangles in the 1-ring. The umbrella operator is flexible in a way that the weights can be adapted to a specific application, e.g. with respect to the principal curvature of the mesh for anisotropic vertex distribution.

Area Equalization

In [SG02], Surazhsky and Gotsman discovered that a (possibly bad) 2D triangulation having triangles with close to equal areas has, surprisingly, a globally uniform, spatial vertex distribution. In their remeshing scheme they alternate between area equalization and Delaunay edge-flips to obtain a close to regular mesh. This iterative process usually does not converge and begins to oscillate, producing different but similar triangulations. The involved optimization problem reduces to solving a system of two linear equations in x and y, and the computational cost is almost negligible.

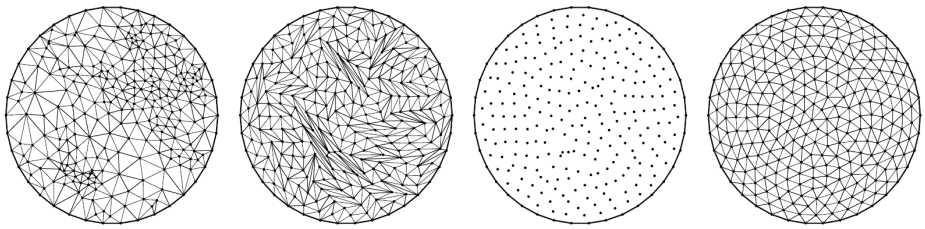


Figure 22: The original triangulation (left). Areas of the triangles are equalized (center-left). Discarding the connectivity reveals the uniform sampling (center-right). The Delaunay triangulation results in a close to regular mesh (right). Images taken from [SAG03].

In [SG03], Surazhsky and Gotsman extend the idea of area based remeshing to work with a density function to create an adaptive vertex sampling. This is done by specifying ratios between the triangle areas depending on the density function.

One of the benefits of using area equalization is the speed of the convergence. Thus, the technique is well suited for distributing the mesh vertices according to the prescribed density function in less time than other relaxation techniques, for example the Lloyd relaxation, which is explained next.

Lloyd Relaxation

The Lloyd relaxation method [Llo82] can be used to construct a centroidal Voronoi tessellation [DFG99] of the domain, i.e. a Voronoi diagram where all sites are located at the centroid of their cells. The Lloyd relaxation is of special interest for this thesis, and is explained in more detail in Section 4. For now, the method is considered to create a good, precise isotropic placement of the samples in a slow, iterative relaxation process.

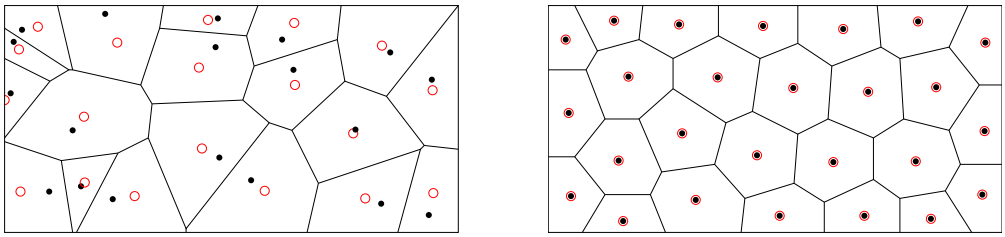


Figure 23: A Voronoi tessellation of random points (left) and a centroidal Voronoi tessellation (right). Dots are the sites of the tessellation, circles are the centroids of the cells. In a centroidal Voronoi tessellation, the sites coincide with the centroids.

Alliez et al. presented a remeshing algorithm that exploits the Lloyd relaxation for isotropic remeshing of surfaces [AdVDI03]. In this work they first distribute the required number of vertices on the mesh using error diffusion, a technique commonly used in image half-toning. The resulting sampling already approximates the density field very well due to a preprocessing step that calibrates the error diffusion. In a second step, the initial sampling is meshed using the Delaunay triangulation and improved by building the weighted centroidal Voronoi tessellation of the points, i.e. by applying Lloyd relaxation on the points.

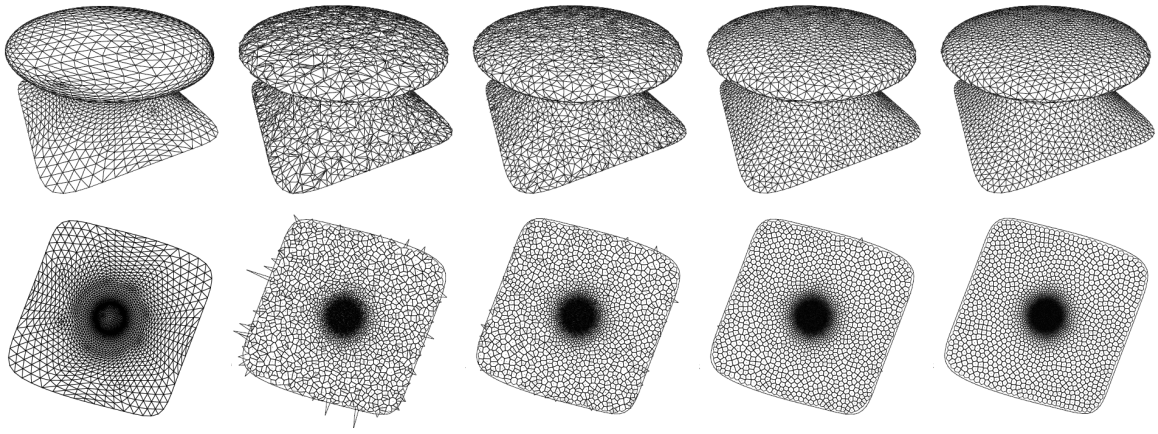


Figure 24: From left to right: Initial mesh, initial sampling, and Lloyd relaxation with 1, 10 and 100 iterations. Bottom row: The corresponding parameterizations, from [AdVDI03].

Since the techniques are only applicable in two dimensions, Alliez et al. parameterize the mesh onto a planar domain, which involves cutting the mesh for closed surfaces with genus > 0. After Lloyd relaxation has been applied, the embeddings are stitched and projected back to the original surface. Although the technique produces great results, the need for global parameterization involves several critical operations:

- Closed or genus > 0 surfaces have to be cut along a cut graph, which is known to be an intricate problem [EHP02].
- Surface cutting introduces a set of artificial boundaries, which generates visually displeasing seams after stitching. The need for techniques to hide the seams [KLS03] complicate the setting further.
- Parameterization of a large domain causes huge distortions. Alliez et al. use stretching factors for all edges to accommodate the distortion. However, numerical issues may arise for large meshes.

Surazhsky et al. present an isotropic remeshing scheme that uses local parameterizations and avoid the complications of global parameterization [SAG03]. The method makes use of local operations only, performed on a copy of the mesh while referring to the original mesh as geometric reference. In the first stage, the algorithm generates the required amount of vertices by simplification or refinement. In the second stage, an initial vertex distribution is produced using an area-based relaxation method [SG03]. The third stage polishes the sampling by constructing a weighted centroidal Voronoi tessellation using Lloyd relaxation.

These techniques are of great significance for this thesis. The centroidal Voronoi tessellation promises precise sampling of the vertices and produces excellent, high-quality triangulations. The next section deals with the details of remeshing using this technique.

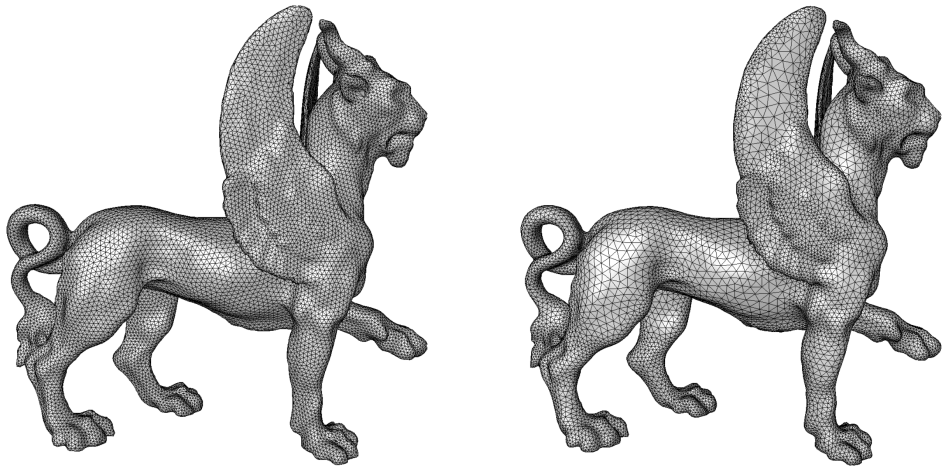


Figure 25: An uniform remesh (left) and a curvature adapted remesh (right), from [SAG03].

4 Isotropic Surface Remeshing

Isotropic remeshing of surfaces aims at sampling the domain to obtain a *tiling*, *tessellation* or *partitioning* of the surface, where the shape of each tile is not biased in a particular direction (*isotropy*), i.e. each tile should be as compact, or round, as possible. Each tile is associated with exactly one sample, and in the uniform case each tile should occupy the same amount of area on the surface. Ideally, each tile of the tessellation should be a disc. However, a disc cannot produce a gapless partition of the domain. Like equilateral triangles and squares, regular hexagons fit together without any gaps and additionally provide optimal compactness in a lattice, see Figure 26.

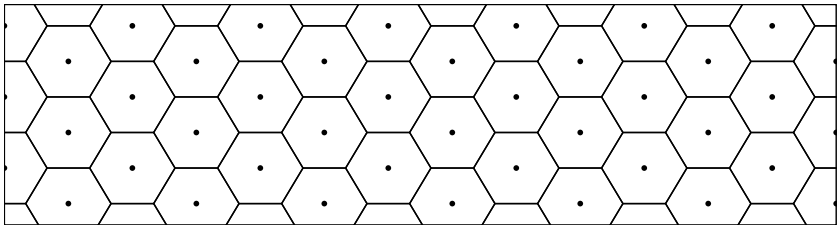


Figure 26: Isotropic, hexagonal tiling of the domain with uniformly distributed samples.

To obtain an adaptive distribution of the vertices in the non-uniform case, e.g. for curvature adapted remeshing, sampling of the domain should be controlled according to a prescribed density function defined on the surface. Partitioning of the density function should be done in a way that the area associated with each sample inherits the *equal-mass enclosing property* [SAG03], that is each tile should contain the same amount of density, or mass, see Figure 27.

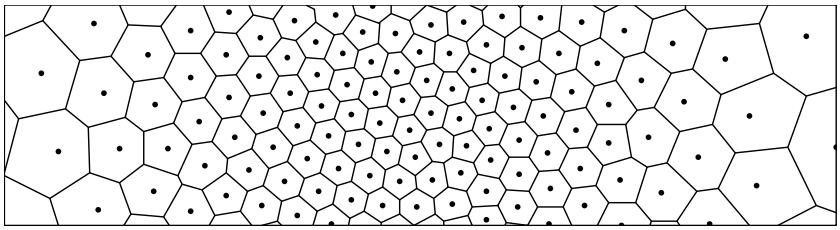


Figure 27: Adapted, non-uniform tiling of the domain with density function $e^{-x^2-y^2}$.

To produce a precise placement of the samples and to obtain the equal-mass enclosing property, it is advisable to switch to a notation where each sample can be associated with an area. Working on the basis of triangles can hardly comply with this requirement, because each tile of a triangulation is associated with three samples instead of one. The method of choice for this thesis is to exploit the nature of the *centroidal Voronoi tessellation* (CVT) of the domain. In a CVT, the position of each sample coincides with the centroid of its Voronoi cell, see figures 23 and 26. With respect to a prescribed density function, the aim is to construct the *weighted, centroidal Voronoi*

tessellation (WCVT), see Figure 27, where the simple CVT is just the uniform case of the WCVT. The WCVT has several nice properties that are directly related to isotropic remeshing:

- Each polygonal tile of the Voronoi tessellation is associated with exactly one sample. Thus area and mass can be associated with each sample.
- In the uniform case, the CVT produces a hexagonal lattice with optimal compactness of the cells, i.e. the tessellation has optimal isotropy.
- In the non-uniform case, the WCVT leads to a tradeoff between compactness and partition of the density function, which allows for adapted tessellations.
- Since a Voronoi diagram is the dual to some Delaunay triangulation, the WCVT implicitly gives a triangulation of the domain. Similar to the Voronoi cells itself, the triangulation comes with optimal compactness, i.e. equilateral triangles, in the uniform case.

The centroidal Voronoi tessellation can be computed in an iterative process using the Lloyd algorithm. The Lloyd relaxation method [Llo82] is a simple, deterministic, fixed-point iteration [DFG99]. Given n sites, the centroidal Voronoi tessellation is obtained by the following three steps:

1. Create the Voronoi tessellation of the n sites,
2. Move the n sites to the centroids of their cells,
3. Repeat the iteration until convergence is achieved.

The Lloyd relaxation repeatedly moves the sites into the centroids of their cells, see Figure 28. To make the Lloyd relaxation consistent with a prescribed density function defined over the domain, each site is relocated to the *center of mass* instead of the geometric centroid of the Voronoi cell. For a uniform density function, the center of mass and the geometric centroid coincide.

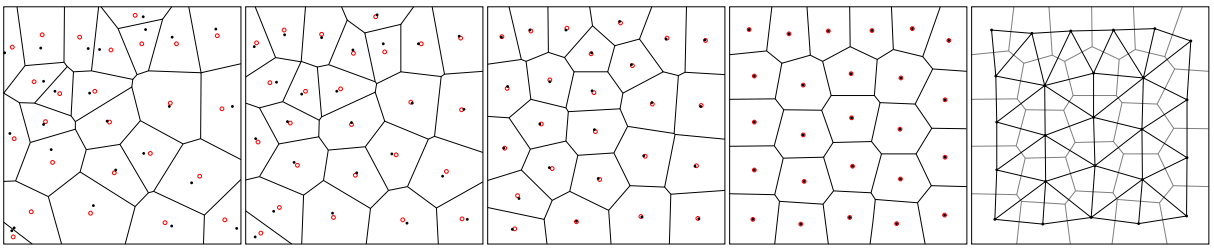


Figure 28: A set of random points and the Voronoi diagram (left), the diagram after 1, 5 and 100 Lloyd iterations, and the dual triangulation after relaxation (right). The centroid of each cell is drawn as red circle. After convergence, sites and centroids of the cells coincide.

4.1 Overview

This section gives an overview of the complete remeshing process. The whole process is build on several algorithms each being an individual step in the creation of the final mesh. The stages of the algorithm are arranged in a pipeline: Starting from an initial model, the pipeline is traversed progressively building the final mesh.

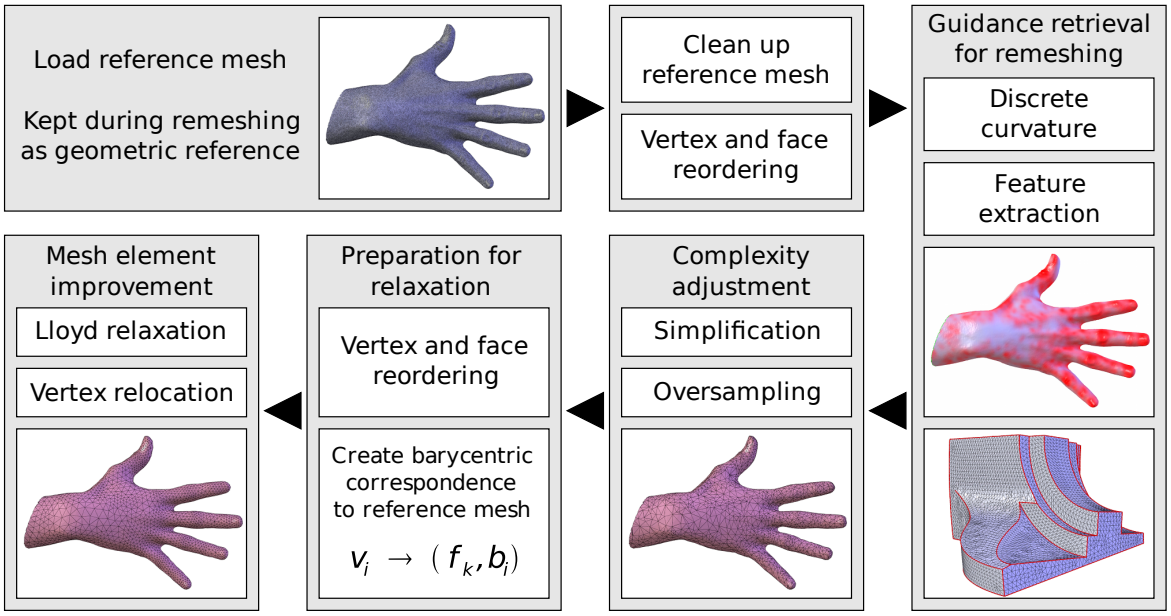


Figure 29: The remeshing pipeline used in this work.

Reference mesh: The first stage in the pipeline is responsible for providing the input mesh. The mesh is cleaned, and the vertices and faces are reordered to improve vertex cache efficiency. The details are presented in Section 4.2.

Guidance retrieval: In the next stage, guidance information is retrieved from the input mesh. The discrete curvature is estimated and a density field is employed to create adaptive remeshes with a denser sampling in curved regions of the surface. Feature edges are extracted to preserve the characteristic appearance of models with sharp edges. The techniques are introduced in Section 4.3.

Complexity adjustment: For the following stages the reference mesh is copied, and all operations are performed on this copy, which is called the *evolving mesh*. The reference mesh is untouched and serves as geometric reference for the rest of the process. To sample the mesh to an exact, user-defined vertex budget, the complexity of the evolving mesh is adapted, which is explained in Section 4.4. Note that throughout this thesis the reference mesh is shown in blue whereas the evolving mesh is shown in red.

Preparation: Since the complexity adjustment changes the mesh, vertex cache efficiency may be lost and an efficient ordering is restored. A barycentric reference for each vertex of the evolving mesh is prepared, pointing to a unique location on the reference mesh. See Section 4.5.1.

Relaxation: The last step of the pipeline polishes the sampling using Lloyd relaxation. During that iterative process, the positions of the vertices are slightly changed in each iteration, always referring to the geometric reference, see Section 4.5.3. Relocation of a vertex is probably the most difficult operation which involves flattening, or *parameterizing*, the surface to two dimensions. Several parameterization techniques exist, and a brief overview is given in Section 4.6.

4.2 Processing the Input Mesh

Processing of the input mesh is the very first stage of the remeshing pipeline. As already mentioned, practical meshes are rarely valid simple surfaces, i.e. they are typically no connected, orientable, two-dimensional manifolds. Instead, a triangle mesh may consist of several unconnected surface parts, may not be closed and have boundaries. Even worse, a triangle mesh can even have invalid, zero area triangles (that is, all three vertices of a triangles lie on a straight line), duplicated and unreferenced vertices. To ensure stability of the subsequent processing steps, the mesh has to be repaired.

4.2.1 Cleaning the Input Mesh

In this step, unreferenced and duplicated vertices are removed from the mesh. The elimination of zero-area faces is essential for calculations that rely on the triangle areas. In particular, flattening the mesh into two dimensions requires solving a linear system of equations: To create a well-conditioned linear system, zero-area faces and duplicated vertices must not be present, otherwise the involved linear system solver will not converge. Ideally, even very close vertices near floating point precision should be eliminated for a well-conditioned system.

An efficient way to detect invalid triangles is to exploit connectivity information. If a triangle references a single vertex two or three times, the triangle is invalid and removed. If a vertex and a connected, neighboring vertex are very close to each other (with respect to some ε bound), the short edge is collapsed, effectively removing two triangles from the mesh (one triangle if the short edge is a boundary edge).

A special kind of degenerated triangle, namely a *cap*, is not easily removed by collapsing a short edge. Although these triangles can also cause zero-area faces in the extreme case, stability of the linear system solver is not directly affected, because the vertices have a reasonable distance to each other. These triangles are not removed during mesh repair, because they are particularly hard to eliminate. This problem is addressed in Section 4.4.2.

4.2.2 Face and Vertex Reordering

For a given triangle mesh, the faces and vertices are typically in no particular order. This order may depend on the software that created the model but it was most likely not created with any specific optimization in mind. At the intuitive level, the order in which faces and vertices are issued does not matter. However, as each triangle is processed for rendering, referenced vertices are processed by a vertex shader in an operation that can be computationally expensive.

Applications whose rendering cost are dominated by per-vertex operations said to be *vertex-bound*. Modern GPUs attempt to avoid unnecessary computations and costly memory accesses with a vertex cache that holds a small number of vertices. To optimize rendering, the aim is to access vertices in an order that reduces the *average cache miss ratio* (ACMR). To do that, the order in which *triangles* are issued is important. For example, if two adjacent triangles are rendered one after another, two vertices of the second triangle are already cached by the GPU and can be used without memory access.

The first practical approach was to use triangle strips and a 2-entry cache. However, the problem of finding a single strip representation is hard and not always possible. Hoppe noticed in [Hop99] that a transparent FIFO cache is very well suited to achieve nearly optimal ACMR ratios if the vertex locality is maximized. Recently Sander et al. proposed a new method [SNB07] for reordering the triangles of a mesh that reduces ACMR by maximizing vertex locality, which results in a more efficient rendering, see Figure 30. Their method produces similar results compared to those algorithms in previous work [Hop99, BG01, YL06, LY06], but is orders of magnitude faster to compute and can even be executed interactively, which justifies the rigorous use of the method for both, the input mesh and the evolving mesh.

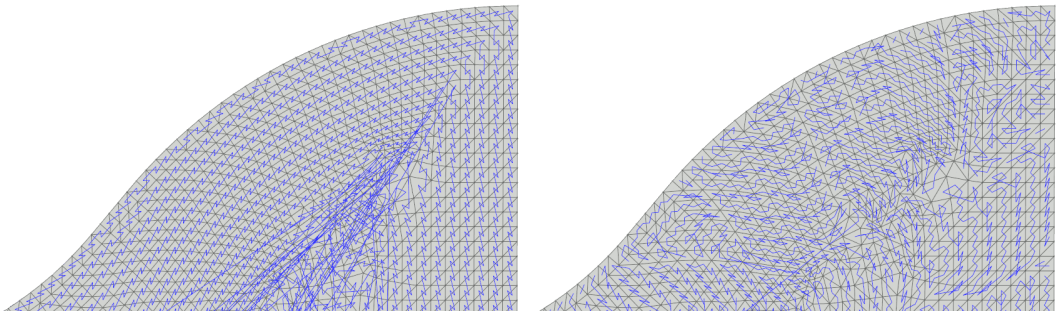


Figure 30: The original face ordering using traditional strips where possible (left), and the optimized, cache-aware face ordering using the method from [SNB07] (right).

Since the original version of the algorithm in [SNB07] only reorders the *triangles* for vertex locality, a modification of the work has been employed for this thesis to rearrange the *vertices* of the mesh. To do this, the cache efficient face ordering is traversed and vertices are issued in order of their first appearance. The benefit of this modifi-

cation lies in the fact that relocation of vertices makes heavy use of caching. For each vertex relocation, a small region of the mesh needs to be flattened to determine the new position of the vertex. This patch-wise parameterization acts on a per-vertex basis, i.e. during Lloyd relaxation, each iteration walks over all vertices of the evolving mesh and creates a patch for the region of the vertex. If this patch is cached, it can be reused for nearby relocations, which results in a significant performance boost.

4.3 Guided Isotropic Remeshing

The default behavior of the Lloyd relaxation does not cope with features or regions of increased sampling in the mesh: vertex clusters gradually drift apart and the algorithm converges to a completely uniform sampling of the domain. This is often not desired.

Since every vertex of the sampling is moved in each iteration of the relaxation process, the position of vertices along feature creases is changed and the characteristic appearance of the model is quickly destroyed if no special care is taken. For this reason, a subset of the edges in the mesh are tagged as *feature edges*. All vertices along the feature crease are constrained during relocation and feature edges may not be flipped. The extraction of features is described in Section 4.3.1.

In order to better approximate the reference surface with a given vertex budget, a denser sampling in more curved areas should be employed. This is achieved with a density field defined over the reference mesh, which describes the relative amount of vertices to be used over the surface. The construction of such a density field is explained in Section 4.3.2.

4.3.1 Recognizing Features

The extraction of feature creases on the mesh is of crucial importance if the surface is not C^1 continuous everywhere and contains tangential discontinuities. These discontinuities are significant for the visual fidelity of the model and must not be destroyed. Since most algorithms assume a smooth surface for their operation, these algorithms need to be adapted in order to account for the features.

This thesis uses a very simple scheme to detect creases on the mesh: Intuitively, the dihedral angle between two adjacent faces of the mesh can be used to measure flatness of the surface. Practically, the angle between the normals of the faces is used for feature detection. This works for both, ridges and valleys, at the same time. If the angle exceeds a certain threshold, the edge between the faces is tagged as feature edge. For a given threshold α, a feature between faces f_1 and f_2 with normals n_1 and n_2 is detected if

$$n_1 \cdot n_2 \quad \leq \quad \|n_1\| \cdot \|n_2\| \cdot \cos\alpha$$

Since n_1 and n_2 are normals with unit length, the normalization step is typically not necessary. Of course, this simple scheme has serious drawbacks in the general case. In

large meshes, feature edges may be detected even if the surface does not have salient features, and sharp angles may be the result of noise. Although the method extracts a reasonable set of features for most CAD-meshes, more robust detection methods may be used for special requirements. For example, [WB01] exploits an area degenerating effect near singularities of the focal surface for feature detection, [YBS05] estimates curvature tensors and derivatives via local polynomial fitting, and recently [ZGM09] uses contextual information of neighboring points for robust detection of perceptually salient features.

During the reconstruction process of models, e.g. using the marching cubes algorithm for volumetric data, the interpolating triangle mesh may chamfer sharp features and introduces aliasing effects. In these cases the detection of features produces unsatisfactory results, and enhancing the situation is referred to as *feature restoration*. In [AFRS03] an edge sharpening algorithm is presented which significantly reduces the aliasing effect by identifying the chamfered triangles and subdividing them to create proper sharp edges.

4.3.2 Density Field Calculation

To generate a non-uniform sampling of the domain with an increased amount of vertices in areas of higher curvature, a density field is employed. Related applications, for example Advancing Front algorithms, construct a similarly defined sizing field, which provides the optimal triangle edge length for any point on the mesh. However, since the remeshing scheme herein works on a given vertex budget, it is better to speak of a *density field*, which is only implicitly related to the sizing of the triangles.

To construct the density field, the curvature of the mesh has to be estimated. Note that the concept of the density field is not necessarily related to the mesh curvature. In the more general setting, the density can be adapted with respect to arbitrary properties of the mesh. From a theoretical point of view, meshes do not have curvature at all, because they are piecewise linear, but having the (unknown) surface in mind that the mesh tries to approximate, curvature can be derived with various techniques. Several sophisticated approaches exist to extract geometric properties from a mesh [MSR07, Rus04], but for the purpose of estimating the discrete curvature for each vertex only, more simplistic approaches provide yet good results.

In particular, a fast and easy method to calculate a per-vertex discretization of the curvature is given in [DHKL01]. The measures they use are based on the approach of replacing every edge with a small cylinder that joints the adjacent faces tangentially. The cylinders are then smoothly blended at the mesh vertices, thus the resulting surface is an approximation of the unknown smooth surface. Well-known theorems from differential geometry can now be applied to derive formulas for the *Gaussian curvature* K and the *absolute mean curvature* $|H|$ at the mesh vertices. These formulas are discussed now.

Curvature estimation: Let v_i be the neighboring vertices of a vertex v for which the *Gaussian curvature* K and the *absolute mean curvature* $|H|$ are to be computed. Let e_i be the edge vectors $e_i = v_i - v$ that point away from the center vertex and α_i the angles between edges e_i and e_{i+1} of triangle T_i. The dihedral angle β_i at edge e_i is the angle between adjacent triangle normals n_{i-1} and n_i, see Figure 31.

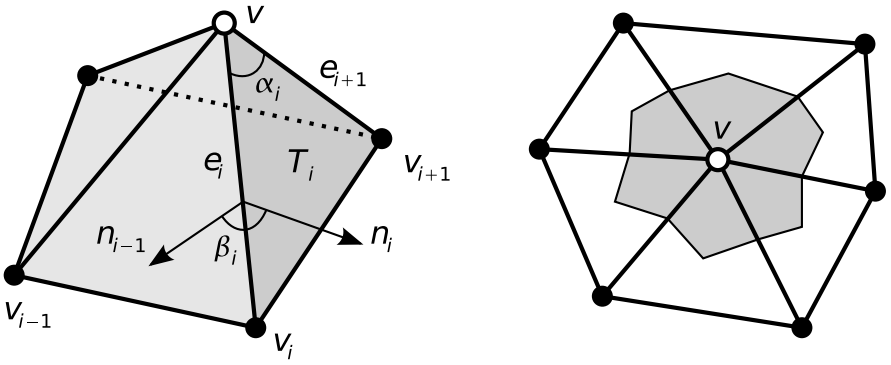


Figure 31: Notation used for calculation of the discrete curvature (left). The Barycentric area S^B of vertex v is one third of adjacent areas (right).

The *integral* Gaussian curvature $\bar{K}$ and the *integral* absolute mean curvature $|\bar{H}|$ with respect to area S attributed to v is defined as:

$$\bar{K} \;=\; \int_S K \;=\; 2\pi - \sum_{i=1}^{n} \alpha_i \qquad \text{and} \qquad |\bar{H}| \;=\; \int_S |H| \;=\; \frac{1}{4}\sum_{i=1}^{n} \|e_i\|\,|\beta_i|$$

The area S associated with v can be calculated in various ways, most important are those definitions which sum up the the total area of the triangle mesh. The Barycentric area $S = S^B$ is very common in literature and easy to compute, which is one third of the area of adjacent triangles. Normalizing by S yields the per-vertex curvature values:

$$K = \frac{\bar{K}}{S} \qquad \text{and} \qquad |H| = \frac{|\bar{H}|}{S} \qquad \text{with} \qquad S = S^B = \frac{1}{3}\sum_{i=1}^{n} T_i$$

The resulting curvature value C_v for vertex v is calculated using a weighting factor λ:

$$\bar{C}_v \;=\; \lambda \cdot |K_v| + (1-\lambda)\cdot |H_v| \qquad \text{with } 0 \le \lambda \le 1$$

Clamping: Since the area S_v associated with v can be arbitrary small, the curvature values are typically not bounded; depending on the size of the input triangles and the amount of noise in the mesh, $\bar{C}_v$ is typically between zero and a few thousands. In order to remove outliers and to eliminate very high density values in some regions,

it is useful to clamp the values into a user-defined range of valid values $[C_{min}, C_{max}]$, where typically $C_{min} = 0$.

$$C_v = \max\left(C_{min}, \min(C_{max}, \bar{C}_v)\right)$$

For smooth and low-noise models, small ranges like $[0, 100]$ yield good results, while ranges like $[0, 1000]$ are better suited for medium-sized scanned objects (e.g. the Stanford Bunny). Large models contaminated with noise may require larger ranges to extract reasonable curvature values. Since the simple method is local and does not take contextual information into consideration, a large amount of noise can lead to arbitrary density values more or less unrelated to the perceived mesh curvature. The method fails in these rare cases and more sophisticated approaches are required.

Scaling and contrast: In order to control the relative amount of samples for low- and high-curvature areas, the range $[C_{min}, C_{max}]$ is scaled to $[0.5, 1.5]$ and a contrast function is applied to the values. This finally yields the density value D_v for the mesh vertices.

$$D_v \quad = \quad \left(\frac{C_v - C_{min}}{C_{max} - C_{min}} + 0.5\right)^{\gamma}$$

This contrast function is defined by a user-specified exponent γ that controls the degree of adaptiveness. A γ-value of 0 means that all density values become 1 and the distribution of the vertices is uniform. A γ-value of 1 has no effect, leaving the range at $[0.5, 1.5]$. An exponent of $\gamma = 10$ for example leads to a range approximately $[0.001, 60]$, which allows for huge varieties in sampling density.

Smoothing: To acquire a density function with smooth transitions between high and low density regions, and to eliminate noise in the curvature values, Laplacian smoothing (which is closely related to the Umbrella operator) is applied by iteratively averaging neighboring density values. A single smoothing iteration for a density value D_v is described by the following formula:

$$D_v^{new} \quad = \quad D_v^{old} \; + \; \lambda \sum_{i=1}^{n} \left(\frac{D_{v_i}^{old} - D_v^{old}}{n}\right) \qquad \text{with } 0 < \lambda \leq 1$$

The smoothing can be controlled by the amount of iterations and the choice of λ. λ relates to the influence of neighboring density values, thus controls the intensity of smoothing, whereas the amount of iterations regulates the area of influence. A high amount of iterations create smoother transitions between high and low density regions. In each iteration, the information from a vertex is propagated to the neighbors only, and a lot of iterations are required for a smooth transition of the density values on dense meshes.

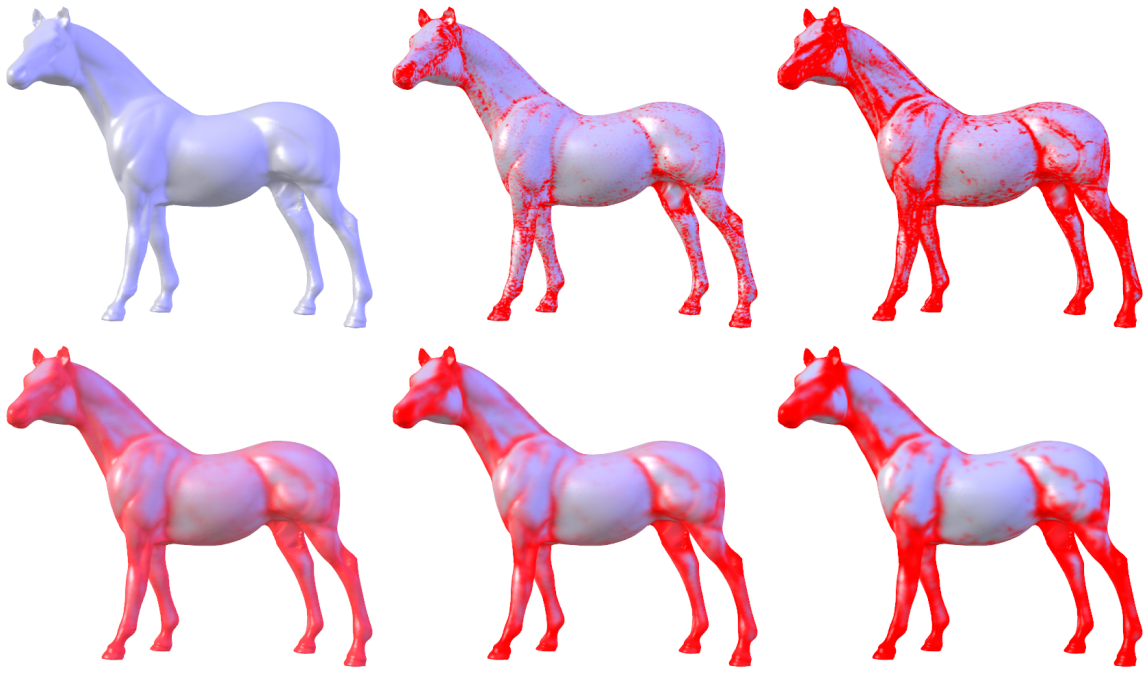


Figure 32: The horse model. Top row: absolute Gaussian curvature $|K|$ (middle) and absolute mean curvature $|H|$ (right). Bottom row: 30 iterations of smoothing with contrast exponent $\gamma = 0.5$ (left), $\gamma = 1.0$ (middle) and $\gamma = 5.0$ (right).

4.4 Adjusting Mesh Complexity

The process of adapting the complexity of the mesh aims at bringing the sampling to an exact, user-defined vertex budget.

Several techniques to achieve an exact vertex amount have been proposed in literature. Alliez et al. use a global error diffusion over the faces and feature edges of the input mesh [AdVDI03]. This technique achieves a great initial sampling, however, meshing of the new samples is difficult and requires flattening the whole domain. This parameterization process is difficult in itself, and it requires cutting and stitching meshes with genus > 0. Farthest point sampling is also capable of providing an initial sampling, but requires to project the samples on the original surface. This is prone to errors and unstable for complex models, and handling of boundaries requires special treatment.

This thesis follows the approach of Surazhsky et al. [SAG03]. Depending on the amount of vertices in the reference mesh, the adaptation performs mesh simplification or refinement to obtain an initial sampling. In case the given vertex budget matches the amount of vertices in the reference mesh, the evolving mesh is simply initialized with a copy of the reference mesh, and the adaptation process is skipped.

A mesh simplification scheme is applied if the vertex amount is to be decreased. The simplification operates on the evolving mesh, which is initialized with a copy of

the reference mesh. As of now, the reference mesh will not be changed and stays untouched during the remaining process. If the vertex amount is to be increased, a mesh oversampling is performed instead. However, the oversampling cannot be applied to the evolving mesh but is performed directly on the reference mesh for various reasons. This is kind of an exception to the remeshing pipeline, and the following sections explain the issue in more detail.

4.4.1 Mesh Simplification

The topic of mesh simplification has been extensively studied, and a wide range of techniques are available in literature, see [CMS98] for a survey. Many polygon simplification methods reduce the number of triangles in a mesh by performing a sequence of *edge collapses*. Given a suitable edge and the corresponding two vertices of the edge, the contraction removes the edge and collapses the two vertices into a single, new vertex. In the general case, each contraction reduces the amount of vertices in the mesh by one, and the amount of faces by two. As a special case, the contraction of a boundary edge removes only one face. For example Hoppe uses edge contractions to create meshes suited for progressive transmission and display [Hop96].

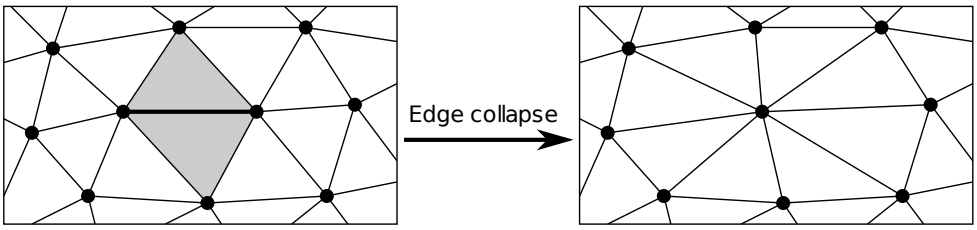


Figure 33: Edge collapse: The edge between two vertices is collapsed. Each collapse deletes one vertex and two faces at a time, placing the remaining vertex at a new position.

Another algorithm based on contractions has been proposed by Garland and Heckbert in [GH97]. They allow contraction of arbitrary vertex pairs, not just physical edges in the model and use quadric error metrics to facilitate better approximations. They extended their surface simplification to preserve a wide range of per-vertex attributes, such as color and texture information [GH98].

The algorithms mainly differ in the way new vertices are placed. A very simple scheme is placing the new vertex in the barycenter of the old vertices. However, there are better methods to find a new position that potentially minimize the error and result in better simplifications, e.g. [GH97] determine the best position based on the assumption of a quadric error metric.

The actual error between the simplified surface and the input surface, namely the Hausdorff distance, is often measured using the Metro tool [CRS98]. For most algorithms the goal is to minimize that global error. For this thesis, however, the global error of the simplification and the placement of the samples is not important, because

the sampling is going to be improved by Lloyd relaxation later on. It is suggestive to stick to a much simpler and more robust kind of mesh simplification, namely vertex decimation algorithms (e.g. [SZL92, SL96, AD01]). These algorithms iteratively determine a vertex for removal, typically based on some error metric. The vertex and all adjacent triangles are removed, and the resulting hole is then re-triangulated. For this thesis, the excellent decimation algorithm from Schroeder et al. [SZL92] has been employed and extended to deal with global features and the density field defined over the reference mesh.

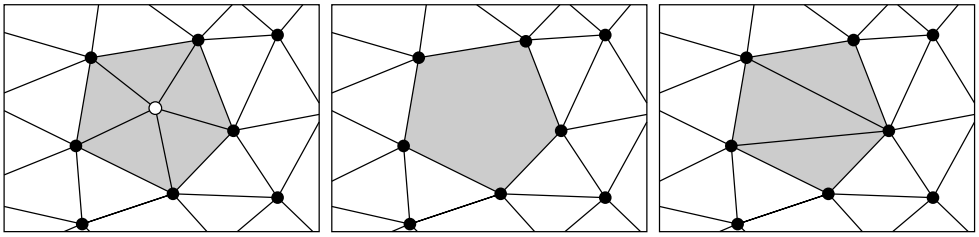


Figure 34: Vertex decimation: A vertex is deleted and the resulting hole is re-triangulated.

The first step of the decimation algorithm classifies all vertices of the mesh into one of the following types: A *simple vertex* is surrounded by a complete cycle of triangles. A *boundary vertex* is surrounded by a semi-complete cycle of triangles and lies on a boundary of the triangle mesh. All other vertices are classified as *complex vertices*, these are non-manifold cases. A simple vertex can further be classified into an *interior edge vertex*, that is, a vertex with exactly two adjacent feature edges, and a *corner vertex*, with one, three or more adjacent features edges. These feature edges are either defined by local geometry, if the dihedral angle between two adjacent triangles is greater than a user-defined threshold, or by global feature edges which may be extracted in a pre-process using more sophisticated algorithms.

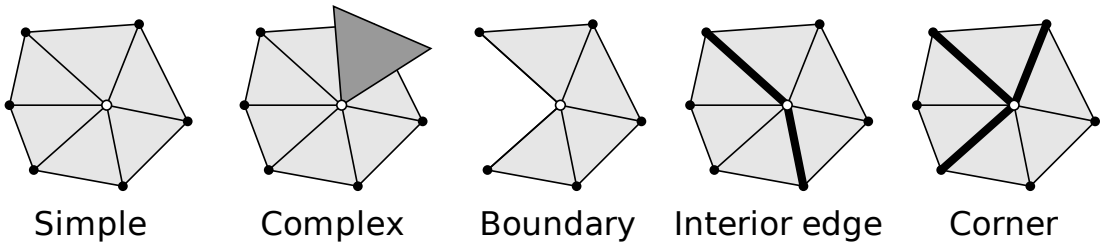


Figure 35: Vertex classification: The five different vertex types.

Complex vertices and corner vertices are never removed from the mesh. Only simple, boundary and inner edge vertices become candidates for removal. To decide if a vertex is actually to be removed, a decimation criterion is evaluated. Depending on the vertex type, different criteria are used. Simple vertices always use the *distance-to-plane* criterion: An average plane for the vertex and all adjacent vertices is created us-

ing a least squares fitting technique, see Appendix A. If the distance from the average plane to the vertex is below a certain threshold, the vertex is removed. Boundary and interior edge vertices use the *distance-to-edge* criterion: The algorithm determines the distance to the line defined by the two vertices creating the new boundary or feature edge after removal. If that distance is below the threshold, the vertex is removed.

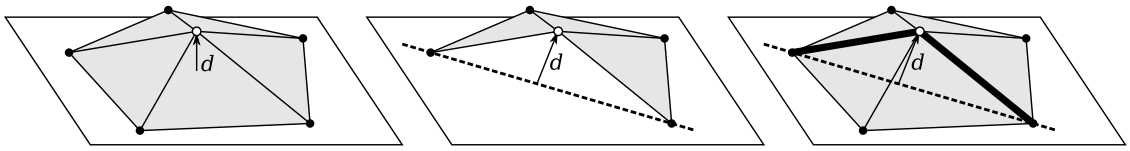


Figure 36: Distance to plane criterion for simple vertices (left), and distance to edge criterion for boundary vertices (middle) and inner edge vertices (right).

These criteria ensure the fidelity to the original mesh. When using solely these criteria it is possible that in nearly planar regions only a few, very large triangles are created. This is not desirable for the purpose of isotropic surface remeshing and a new check is introduced: In addition to the fidelity checks, a vertex must also pass a criterion that is based on some area associated with each vertex. This efficiently prevents the creation of inappropriately large triangles. In particular, the use of the *barycentric area* of a vertex seems to be a reasonable and easy-to-compute choice; the barycentric area of a vertex is one third of the summed area of adjacent triangles, this is the same area that is used to compute the discrete curvature of the mesh (see Section 4.3 for details). In presence of a density function defined over the original mesh, the area checks are extended to mass checks. The *approx. barycentric mass* at a vertex is computed as one third of the summed masses of adjacent triangles. Although this is just an approximation of the real *barycentric mass* at the vertex, the method works well in practice, is very efficient and the approximated barycentric masses sum up to the total mass of the mesh.

If a vertex passed the fidelity and area checks, it is allowed to be removed. Deleting the vertex and all its adjacent triangles leaves a hole in the mesh, which must be re-triangulated, effectively reducing the mesh by one vertex and two faces for each removal. The triangulation needs to be carefully created to avoid intersecting and degenerate triangles. In case the new triangulation cannot be created, the vertex is not removed. To fill the resulting hole in the mesh, the loop of vertices forming the hole is identified and a recursive loop-splitting procedure is employed. An initial static split is performed if an inner-edge vertex is to be removed, to keep the features intact.

Starting with the initial loop(s), each loop is divided into two loops using a *split line* defined by two non-neighboring vertices of the loop. If a loop has only three vertices left, a triangle is created and the recursion stops. To avoid intersections, a loop split is evaluated using a *split plane*. This split plane contains the split line and is orthogonal to the average plane of the vertices (as used for the distance-to-plane fidelity checks). To determine whether the split creates two proper non-overlapping sub-loops, the

split plane is used for half-space comparison. If all vertices of a sub-loop are on the same side of the split plane, the split is valid. Since there are typically more than one valid splits for a given vertex loop, the best possible split line is determined. All possible split lines are evaluated and the split with maximum quality ratio is selected. The quality ratio is calculated by taking the minimum distance of all vertices to the split plane divided by the length of the split line. This ratio can be constrained to prevent degenerated triangles from being created.

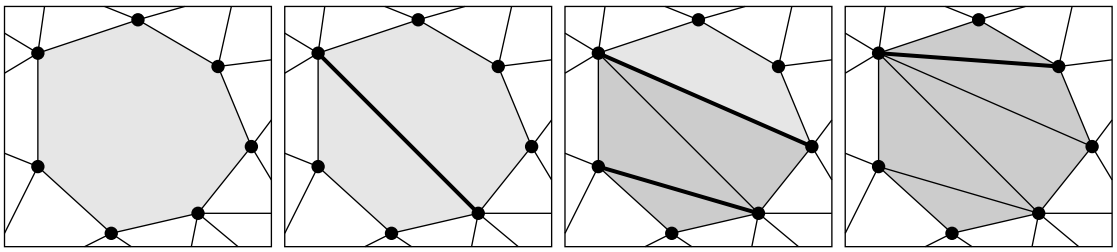


Figure 37: Recursive loop splitting: The best split line for each loop is found, the loop is split and recursively re-triangulated.

Special topological operations can occur during the simplification process, for example tunnels may be joined or duplicated triangles can be created. These cases are treated using a classification on the vertex loop that is about to be re-triangulated. If the vertex loop is *complex*, that is, if two non-neighboring vertices of the loop are connected with an edge after the adjacent triangles are removed, deletion of the vertex is not permitted.

4.4.2 Mesh Oversampling

The idea behind mesh oversampling is to successively refine the mesh until the required vertex budget has been achieved. One major obstacle is the degeneration of triangle quality if refinement is done in a naive way, for example by placing a new vertex at the barycenter of a triangle. This causes all angles in the triangle to be halved, thus triangles quickly degenerate. Another issue is the choice of the new vertex position: If the vertex is placed directly on the triangle to be refined, the original shape of the model is always preserved, which might be good for actual piecewise linear surfaces, but the increased amount of vertices could approximate a *smooth* surface in a much better way.

One way to improve the positions of new vertices is to place them onto a local approximation of a smooth surface, for example a surface defined by PN triangles as proposed by Vlachos et al. in [VPBM01]. However, the problem of degenerating triangles is not acceptable. One observation is that there is no proper way of locally injecting samples without harming the triangle quality. Instead, oversampling should be better performed as a global process. Global insertion, as opposed to local inser-

tion, also affects adjacent triangles, for example if new vertices are placed on the edges of the mesh. Although it is not possible to directly sample the mesh to an exact vertex budget, well-known subdivision techniques can be applied to create smooth surfaces without harming the triangle quality, see [Uml05] for a survey.

In contrast to simplification, the operations are directly applied to the original mesh for various reasons. Most important is that the original model may not be suited to grow reasonably round surface patches, which are required for vertex relocation later on, when applying the relaxation. This is especially true for CAD models that contain many large or degenerated triangles. Another reason is that it is not possible to achieve an exact vertex budget with the global techniques described in this section. An adequate solution is to simplify the oversampled mesh to achieve a precise amount of samples, and use the oversampled mesh as reference during remeshing.

Mesh subdivision: Subdivision of meshes, or *subdivision surfaces*, dates back to the late 70s with the introduction of the Catmull-Clark [CC78] and the Doo-Sabin [Doo78, DS78] subdivision schemes. In later work, Charles Loop introduced the Loop subdivision [Loo87], an easy and efficient algorithm to create smooth surface approximations on triangular meshes. These early methods were mostly not able to deal with sharp features, resulting in an overall smooth surface even at sharp feature edges. Hoppe extended the Loop subdivision scheme to deal with piecewise smooth surfaces in [HDD+94]. Due to the flexibility and simplicity, the Loop subdivision has been chosen for this thesis.

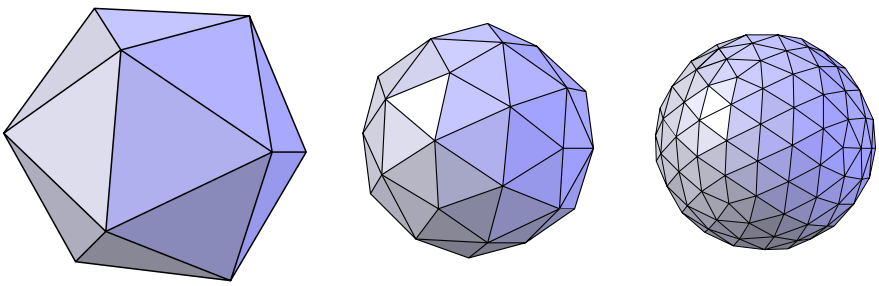


Figure 38: Loop subdivision of an icosahedron (left), after one iteration (middle) and after two iterations (right).

The early subdivision schemes were approximating, i.e. the original vertices, or control points, are not part of the limit surface but original vertex positions are adjusted with respect to smoothness conditions. This typically causes shrinking of the mesh within some bound. However, for denser meshes the shrinking is usually not a problem. In interpolating schemes, like the Butterfly subdivision scheme [DLG90], the original vertices are not allowed to move. This causes the mesh to enlarge and it has been observed that the resulting limit surfaces tend to be not as smooth as with approximating schemes.

Loop subdivision: The Loop subdivision scheme iteratively splits each triangle into four smaller triangles, called the *refinement*, which increases the amount of triangles by factor four, see Figure 39. For this reason, typically only few iterations are applied. The refinement is applied in two phases. In the first phase, new vertices are inserted on the edges of the mesh using the weighted average of neighboring vertices (*control points*). In the second phase, the control points (original vertices) are moved to obtain a smooth surface, i.e. to achieve continuity of the tangent planes.

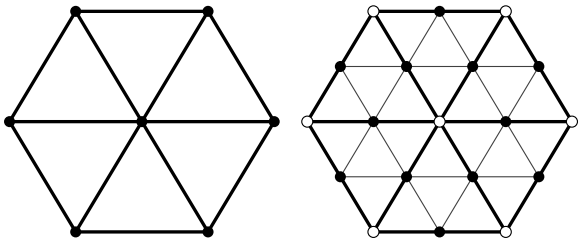


Figure 39: Loop subdivision refines a triangle into four sub-triangles.

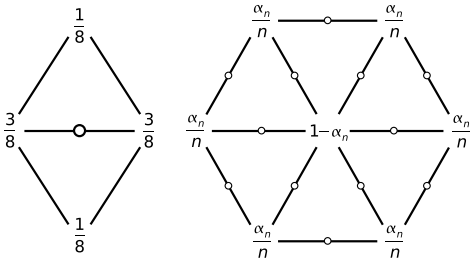


Figure 40: Position masks for edges and original vertices.

The positioning scheme of the vertices can be visualized in diagrams called *subdivision masks*, *stencils* or *templates*, see Figure 40. The mask is centered over a vertex and the new position is calculated with an affine combination of neighboring vertices and the vertex itself. First, all edges are refined using the position mask for edge vertices: The position of the new vertex x^{new} with neighboring vertices $x_1, \ldots, x_4$, where x_2 and x_3 are the vertices adjacent to the edge, is calculated as:

$$x^{new} = \frac{1}{8}(x_1 + 3x_2 + 3x_3 + x_4)$$

Then, all remaining control vertices are moved using the position mask for original vertices. The new position x_0^{new} of the vertex x_0^{old} with neighboring control vertices x_i, $1 \leq i \leq n$ is calculated as:

$$\begin{aligned} x_0^{new} &= (1-\alpha_n) \cdot x_0^{old} + \frac{\alpha_n}{n} \sum x_i \\ \alpha_n &= \frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n}\right)^2 \end{aligned}$$

Piecewise smooth subdivision: The fitting of a smooth surfaces often produces unpleasant results if the original surface has sharp edges. In [HDD+94] complementary rules and masks have been developed to deal with piecewise smooth models and commonly occurring features the authors call *creases*, *corners* and *darts*. The method requires a subset of all edges to be tagged as sharp; this subset should also include boundary edges. The extraction of those features has been discussed in Section 4.3.1.

Vertices are classified according to the amount of incident feature edges e. *Smooth vertices* have no incident features, i.e. $e = 0$. A *dart vertex* has $e = 1$, a *crease vertex* has $e = 2$ and a *corner vertex* has $e > 2$. A crease vertex is either *regular* if the vertex has valence 6 and exactly two smooth edges at each side of the crease, or *non-regular* in all other cases. A crease vertex on the mesh boundary is regular if it has valence 4.

The new positions for smooth and dart vertices are calculated using the smooth vertex mask from Figure 40. Crease vertices are only moved along the crease, and corner vertices are never moved, see Figure 41. Finding the right mask for edge refinement is more complicated and depends on the type of incident vertices. If at least one of the vertices is smooth or a dart, the smooth edge subdivision mask is applied. If a regular crease vertex meets a non-regular crease vertex or a corner, non-regular crease edge subdivision is performed, where the regular crease vertex gets more weight. In all other cases, regular crease edge subdivision is performed.

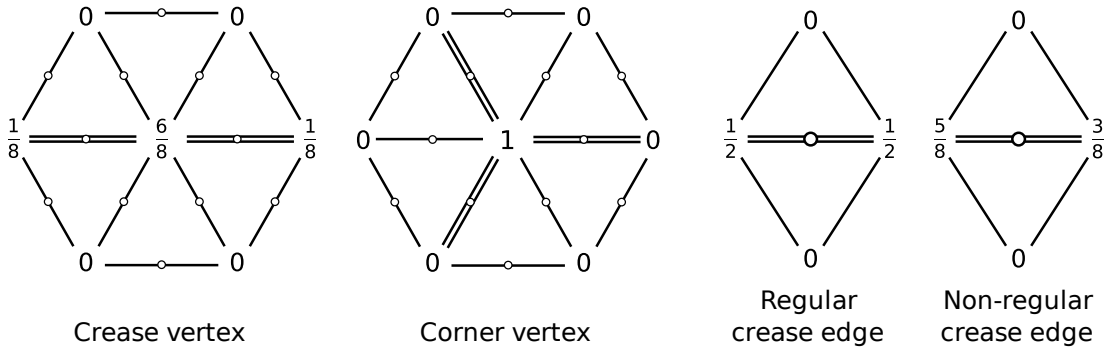


Figure 41: Loop masks for non-smooth vertices (left) and feature edges (right).

Given a mesh and a set of tagged feature edges, the modified Loop subdivision algorithm together with the new subdivision masks is able to create piecewise linear smooth surfaces, see Figure 42 for an example.

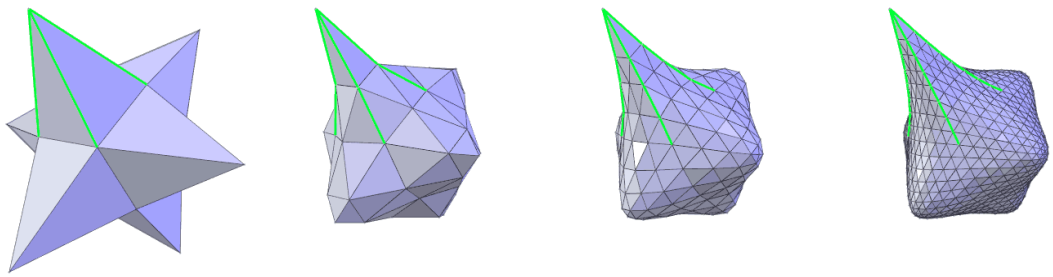


Figure 42: Feature-sensitive Loop subdivision of a star with tagged feature edges.

Although regular subdivision of surfaces is a powerful tool to create smooth and almost regular meshes, subdivision produces best results only if the triangles have acceptable quality. The worst angle in each triangle is also present in all the sub-triangles

after subdivision. For this reason, subdivision is rarely suited for meshes with degenerated faces and numerical stability becomes a serious issue.

CAD meshes: Oversampling of meshes that are typically generated by tessellation units of CAD applications is more difficult because a simple mesh subdivision scheme is not applicable due to the large amount of heavily degenerated triangles. The tessellation goal of a CAD application is to create a mesh with as few as possible triangles for a given error bound, which is fundamentally different from the generation of high-quality meshes. Consequently, the intuitive intention is to insert new samples, especially in regions where only few samples are present, to create a properly sampled domain and to apply the relaxation procedure afterwards. This is achieved by removing the degenerated triangles while inserting new samples.

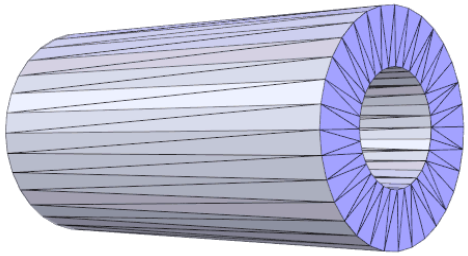


Figure 43: A cylinder exported from a CAD application: Degenerated triangles are a very common artifact.

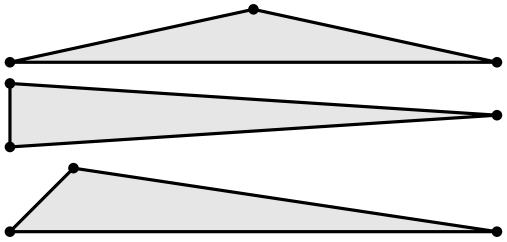


Figure 44: Types of degenerated triangles: A cap (top), a needle (middle) and a cap-like needle (bottom).

A closer analysis of these meshes reveals two different types of degenerated triangles, *caps* and *needles*, see Figure 44. While needles can be removed by collapsing the short edge, caps are much harder eliminate, because none of the relatively long edges can be collapsed without creating new degeneracies on adjacent triangles. One solution for this problem is to subdivide the long edge in a global operation to obtain two needles. This global operation needs to ensure that no new caps are created.

Botch and Kobbelt propose a scheme [BK01] to eliminate degenerated faces based on a mesh slicing approach. For each cap, an appropriate split plane is constructed to eliminate the cap, and the whole mesh is sliced with that plane. In the first pass, the plane is used to partition the space in two half-spaces, each vertex is classified as being above, below or directly on the plane with respect to some ε. In the second pass, all triangles that intersect with the plane are subdivided into two or three triangles, depending on whether one or two edges intersect with the plane. After all caps have been removed, the mesh is simplified by collapsing the short edges of the remaining needles, thus eliminating the degenerated triangles.

Mesh slicing is a robust technique because it operates on vertices only and not on the faces. To ensure that no additional caps are created, the slicing operation snaps

vertices to the plane if the distance is smaller than some ε. Additionally, if the cutting plane and the intersected edge are nearly parallel, the edge is snapped to the plane.

Although the mesh slicing approach is capable of inserting new samples directly on the long edges of caps, the results are not fully satisfactory. For example, CAD meshes or parts of them may only contain needles, and there is no need to eliminate any problematic cap. New samples may still be inserted by slicing the mesh with a set of axis aligned planes, but this produces even worse needles, especially if a needle is sliced several times with a dense set of cutting planes, see Figure 45.

The scheme of Botch and Kobbelt has been improved for this thesis. Instead of immediately creating the new triangles as a triangle is intersected, a lazy triangulation scheme is used to prevent the staggering creation of new needles. In the first pass the mesh is sliced with all planes and intersection points are remembered. Subsequent slicing operations are also snapped to these new points. After slicing is complete, each triangle is re-triangulated using the points on the edges. Re-triangulation is performed with the recursive loop-splitting procedure as explained in Section 4.4.1. The downside of this approach is the quadratic complexity of the re-triangulation scheme, and triangles with a lot of intersection points consume a considerable amount of processing time.

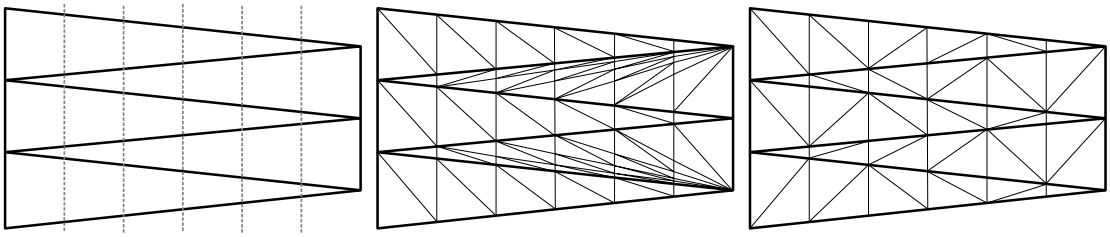


Figure 45: Mesh slicing procedure of an initial mesh using five slicing planes (left) with immediate triangulation (middle) and with lazy triangulation (right).

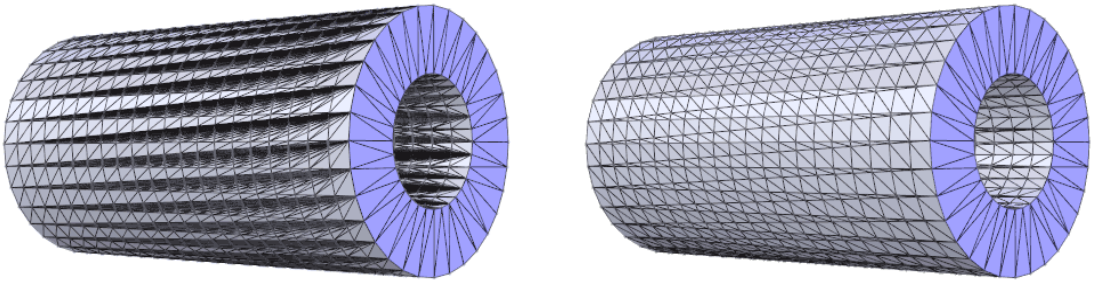


Figure 46: Slicing of a cylinder model using 20 slicing planes along the x-axis with immediate triangulation (left) and lazy triangulation (right).

4.5 Polishing the Sampling

After the mesh complexity has been adapted, the positions of the vertices are polished to obtain a precise, isotropic sample placement. Polishing the sampling is done by iteratively relocating every vertex in the triangulation, this process is called vertex relaxation. Several relaxation schemes has been proposed in literature, for example several smoothing operators [BHP06], angle-based smoothing and area-equalization [SG03]. Lloyd relaxation is used as central concept for isotropic surface remeshing in this thesis. However, other relaxation methods can easily be implemented within the provided relaxation framework.

4.5.1 Providing Vertex References

At this point in the remeshing pipeline, the *reference mesh* and the *evolving mesh* are both loaded. The task is to perform the relaxation procedure on the evolving mesh, iteratively relocating every vertex on the surface of the reference mesh. The evolving mesh vertices carry information that uniquely defines their position on the reference surface. Such a vertex position v can be specified with a pair (f, b) consisting of a face f of the reference mesh and a barycentric coordinate $b = (b_1, b_2, b_3)$ of v with respect to f. With that notation every point on the reference surface can be expressed. Given such a reference (f, b) for v, the coordinates of v can be evaluated by taking the vertices $f = (v_1, v_2, v_3)$ from the reference mesh and calculating the affine combination $v = b_1v_1 + b_2v_2 + b_3v_3$ with respect to the barycentric coordinate b.

During the relaxation process, the task of relocating a vertex v boils down to finding a new barycentric coordinate with respect to the face on the reference mesh that contains v. The resulting position of v is then easily calculated using the above method. Before the relaxation process can be started for the first time, the vertex references need to be initialized. This should be done as soon as the evolving mesh is created, either caused by copying the reference mesh or as the result of the simplification process. Thanks to the fact that the simplification only deletes and never moves a vertex, every vertex of the evolving mesh coincides with a vertex of the reference mesh, and the vertex references are easily calculated by picking any adjacent face on the reference mesh for f and the barycentric coordinate that corresponds to the vertex, i.e. $b = (1, 0, 0)$, $b = (0, 1, 0)$ or $b = (0, 0, 1)$.

4.5.2 Initial Sample Distribution

One tough challenge in remeshing algorithms that use local operations only is to produce an initial distribution of the samples with respect to the density function. For example, if the input mesh has an overall uniform vertex distribution but should be adaptively remeshed with the same vertex budget, vertices must be redistributed to match the density function. Although the Lloyd relaxation is theoretically able to per-

form this task, convergence is particularly slow in these cases: A vast number of iterations are required on the supposition that numerical accuracy is sufficient.

The most important tool to deal with this problem is to make heavy use of simplification. Since the simplification algorithm makes decimation decisions based on triangle areas or masses, it can be utilized to create meshes that already appropriately sample the surface according to the density function. Even if the vertex budget should not be changed, this fact can still be exploited by regular subdivision of the input mesh and subsequent simplification. However, subdivision is often not an acceptable solution for huge input meshes.

Another possibility to create an initial sampling is to make use of *area equalization* [SG02]. The authors discovered that equalizing the areas of triangles in an iterative process results in a fair, globally uniform vertex distribution while requiring very few iterations. Equalization is done by moving a vertex to a new position within the vertex' umbrella in a way that the area of adjacent triangles are as close as possible to each other. Thus area equalization is just a different relaxation procedure that is easily implemented in this remeshing framework. In [SG03] the authors extended the technique to deal with a density field by equalizing the approximate masses of triangles. To be more precise, they equalize the ratios of the areas with respect to an approximate density value.

For the given umbrella polygon $P = (p_1, \ldots, p_n)$, $p_i = (x_i, y_i)$, where A is the area of P, the aim is to find a new position $v = (x, y)$ for the center vertex such that:

$$(x,y) \;=\; \arg\min_{(x,y)} \sum_{i=1}^{n} \left(A_i(x,y) - \mu_i A\right)^2 \qquad \text{with} \qquad A_i(x,y) \;=\; \begin{vmatrix} x_i & y_i & 1 \\ x_{i+1} & y_{i+1} & 1 \\ x & y & 1 \end{vmatrix}$$

In the uniform case, μ_i is set to $1/n$, which equalizes the areas of the triangles. To extend the method, $\bar{\mu}_i$ is set to the density value between vertices p_i and p_{i+1}. The μ_i's are then computed to be normalized ratios of the areas, i.e. $\sum \mu_i = 1$. The involved optimization problem reduces to solving a linear system $Mv = b$ with two equations and two unknowns and has a unique solution. The linear system is given by

$$\begin{pmatrix} \sum_{i=1}^{n} \bar{y}_i^2 & \sum_{i=1}^{n} \bar{x}_i \bar{y}_i \\ \sum_{i=1}^{n} \bar{y}_i \bar{x}_i & \sum_{i=1}^{n} \bar{x}_i^2 \end{pmatrix} v \;=\; \begin{pmatrix} -\sum_{i=1}^{n} \bar{y}_i (x_i y_{i+1} - x_{i+1} y_i - \mu_i A) \\ -\sum_{i=1}^{n} \bar{x}_i (x_i y_{i+1} - x_{i+1} y_i - \mu_i A) \end{pmatrix}$$

where $\bar{x}_i = x_{i+1} - x_i$ and $\bar{y}_i = y_i - y_{i+1}$. The linear system can now be solved by inverting the matrix. See Appendix D on how to derive this linear system.

4.5.3 Lloyd Relaxation

The Lloyd relaxation method is a deterministic fixed-point iteration to create a centroidal Voronoi tessellation for a set of points in 2D, that consists of three steps:

1. Create the Voronoi tessellation of the n sites,
2. Move the n sites to the centroids of their cells,
3. Repeat the iteration until convergence is achieved.

The aim is to extend the original method to create the *weighted* centroidal Voronoi tessellation with respect to the prescribed density function directly on the evolving mesh while referring to the reference mesh as geometric reference. It is not needed to explicitly work with the Voronoi tessellation but rather with the dual Delaunay triangulation. Therefore the Delaunay property is maintained by flipping edges in 3D after each iteration of the relaxation procedure. The Voronoi cells of the vertices are then calculated in a parametric domain and each vertex is moved to the centroid of its Voronoi cell. The steps are now described in detail.

Restoring the Delaunay criterion: The Delaunay triangulation of the mesh is restored with a few iterations over all *triangle pairs* that share a common edge. The pairs are inspected in each iteration and if the sum of the angles opposite to the common edge is greater than $180°$, the edge is flipped. Flipping edges may invalidate the Delaunay criterion of adjacent triangles, thus the process is iterated until no more edges can be flipped. Usually very few iterations are required to completely restore the Delaunay triangulation.

Computing the centroid: Every relaxation step relocates a single vertex to the centroid of its Voronoi cell. Instead of constructing the Voronoi diagram for all points of the evolving mesh, a single Voronoi cell is constructed for each vertex of the mesh. To calculate the cell, the umbrella of the vertex is mapped to a planar domain first. The Voronoi cell is then created in 2D.

For a vertex v, adjacent triangles $T_1, \ldots, T_n$ and adjacent vertices $v_1, \ldots, v_n$, where n is the degree of v, the umbrella of v is flattened using the following procedure: The vertex v is mapped to the origin. Vertices v_i are mapped in a way that edge lengths $\|e_i\| = \|v - v_i\|$ as well as the relative angles between the edges are preserved. Relative angles means that each angle α_i between e_i and e_{i+1} is multiplied with ratio $\frac{2\pi}{\sum \alpha_i}$ to sum up to 2π in the mapping. The method is an approximation to the *geodesic polar map* and is also used by Floater for the shape-preserving parameterization [Flo97].

After the umbrella has been mapped to 2D, the Voronoi cell is computed. Since the triangulation inherits the Delaunay property, the vertices of the Voronoi polygon coincide with the circumcenters of the triangles T_i, and the Voronoi cell is constructed by connecting the circumcenters of T_i. The algorithm needs to be more complex for samples in the vicinity of features.

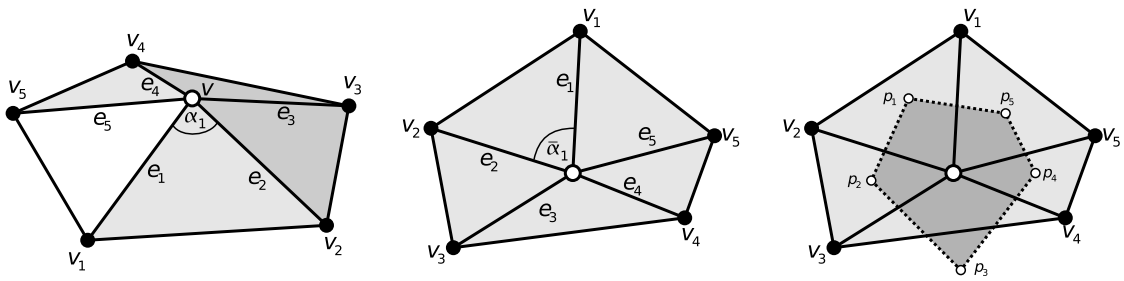


Figure 47: Center vertex and the umbrella in 3D (left) are mapped to the plane using a shape-preserving technique (middle). The Voronoi cell is computed in the parametric domain by connecting the triangle circumcenters (right).

Preserving features: To make the Lloyd relaxation consistent with features, the cells are modified if they overlap with feature edges. Overlapping cells are clipped with these edges, which effectively disconnects two smooth regions of the mesh separated by a crease, see Figure 48. A clipped Voronoi cell leads to a different centroid farther away from the features, thus the samples are repulsed by creases. This leads to a plausible distribution of the samples in the vicinity of these edges and samples on either side of the crease do not influence each other.

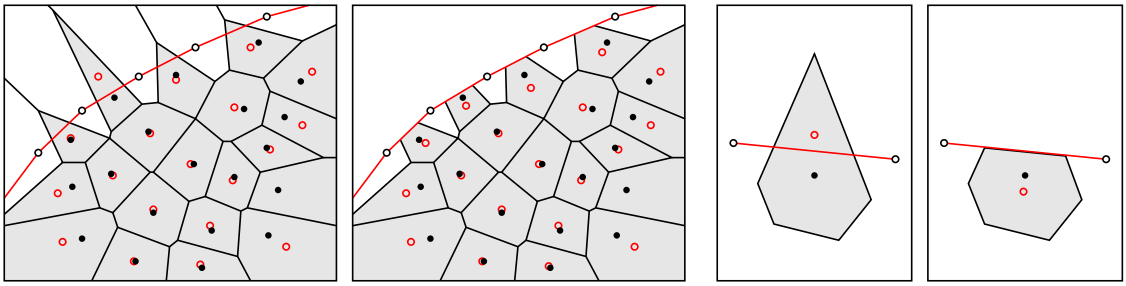


Figure 48: Clipping of Voronoi cells makes the Lloyd relaxation consistent with features.

Clipping appears only for cells of sites that are very close to feature edges. After the first few iterations of the relaxation process, when the major improvement on the sampling is already accomplished and close samples have been repulsed, clipping rarely occurs.

To calculate the centroid of the possibly clipped Voronoi cell, one has to distinguish between the uniform case, i.e. where no density function is given, and the adaptive case, where the centroid is calculated as the center of mass.

Centroid with uniform density: The calculation of the centroid with unit density is just a special case of the more general setting with a given density function, where all density values are equal. However, a separate handing is useful for performance

reasons. For a polygon P with vertices $p_i = (x_i, y_i)$, the area A of P is given by:

$$A = \frac{1}{2}\sum_{i=1}^{n}(x_i y_{i+1} - x_{i+1} y_i)$$

Finally, the centroid $c = (c_x, c_y)$ of p is calculated as:

$$c_x = \frac{1}{6A}\sum_{i=1}^{n}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$
$$c_y = \frac{1}{6A}\sum_{i=1}^{n}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

As one can see, the second factor in the sum is equal to the summands of the area formula, which allows for very efficient calculation of A and c in a single pass over the polygon vertices.

Center of mass: Calculation of the Voronoi cell's centroid, or better called *center of mass*, with respect to a prescribed density function is computationally a bit more expensive. In the first step, the density values of the umbrella vertices v_i and center vertex v are acquired. Since the density function is only defined at the reference mesh vertices, the density values for v_i and v are each evaluated with a linear interpolation over the face of the reference mesh that contains the sample. This is easily done by using the three density values of the reference mesh together with the barycentric coordinate stored in the vertex reference.

In the second step, the density values d_i at the Voronoi polygon vertices p_i are calculated by finding the triangle that contains p_i in the parametric domain. If the triangle is found, d_i is again evaluated with a linear interpolation of the density over the triangles in the mapping. If p_i lies outside the umbrella, p_i is located between two consecutive edges e_k and e_{k+1}. In that case, the density is linearly interpolated between vertices v_k and v_{k+1} with respect to the distance from p_i to e_k and e_{k+1}.

At this point, the Voronoi polygon P, density values d_i at the polygon vertices p_i and the density at the center vertex v are known. To proceed, the polygon P is partitioned into individual triangles $P_i = (v, p_i, p_{i+1})$. The mass m_i and the center of mass c_i for each triangle P_i is calculated using using a Bernstein-Bézier method for bivariate polynomials [LS07], see Appendix C. The center of mass c for P is then computed by weighting the individual c_i with the mass of P_i.

$$c = \frac{1}{\sum_{i=1}^{n} m_i}\sum_{i=1}^{n} m_i c_i$$

Back to the surface: After the centroid c has been calculated, the triangle T that contains c in the parametric domain is determined and the barycentric coordinate b of c

with respect to T is computed. Having triangle T and the barycentric coordinate b of the centroid, a new vertex position located on the evolving mesh could be calculated. It is, however, required to relocate the vertex on the surface of the *reference mesh*, otherwise fidelity to the original surface is quickly lost due to error accumulation. Several authors propose to project the new vertex position back to the original surface, but this involves a computationally expensive and complex operation that may even lead to wrong projections. Instead, vertex relocation is done with a recent parameterization technique that has been proposed by Surazhsky et al. [SG03]. This technique is now explained.

4.5.4 Vertex Relocation

The new position of the vertex to be relocated is given in terms of a barycentric coordinate b_e within a specific face f_e of the evolving mesh. The aim is to acquire a barycentric coordinate b_r within a face f_r of the reference mesh, i.e. bring the position back to the original surface. Recall that for each vertex of f_e a reference to the original mesh is maintained during the algorithm.

The idea is now to parameterize only a small part of the original mesh to a planar domain that is required to relocate the vertex. After a suitable region has been flattened, three positions in 2D are calculated using the references to the original surface from the vertices $f_e = (v_1, v_2, v_3)$. These positions together with the barycentric coordinate b_e yield the position of the new vertex in the planar domain. The triangle f_r that contains the new vertex is looked up and the barycentric coordinate b_r with respect to that face is computed. The pair (f_r, b_r) is stored as new vertex reference, and finally the new vertex position is obtained in 3D. This procedure is now explained in detail.

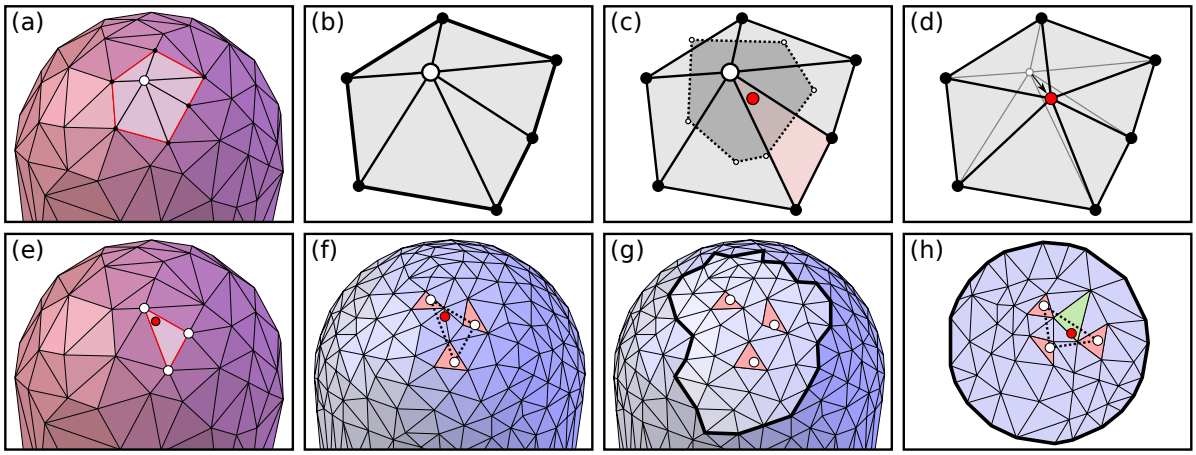


Figure 49: Vertex relocation: (a) The vertex to be relocated. (b) Umbrella mapped to a planar domain. (c) Voronoi cell and centroid. (d) The new umbrella. (e) Face containing the centroid on the evolving mesh. (f) Faces and centroid on the reference mesh. (g) The region to be parameterized. (h) The patch with the face containing the centroid on the reference mesh.

Patch construction: A technique to construct a small patch required to relocate a vertex is now discussed. The patch is created on the basis of the pair (f_e, b_e) that gives a unique position on the evolving mesh. Let v_1, v_2, v_3 be the vertices of f_e and $(f_1, b_1), (f_2, b_2), (f_3, b_3)$ the corresponding vertex references of v_1, v_2 and v_3. The goal is to create a patch on the reference mesh that contains the faces f_1, f_2 and f_3. For every face f_i, a circular neighborhood $\mathcal{N}_i$ of faces is collected using a breadth-first growing approach until the other two faces f_{i+1} and f_{i+2} has been visited (all face indices are modulo 3). The union of $\mathcal{N}_1, \mathcal{N}_2$ and $\mathcal{N}_3$ is then taken as surface patch P_{3D}.

The constructed patch is small enough to have negligible distortion for vertex relocation, but is larger than necessary: That allows reusing the patch for nearby relocations. This caching approach is explained in Section 4.5.5.

Several problematic situations can arise during the creation of a patch. For example, if a patch is growing on a cylinder-like region with small diameter, the patch can twine around the cylinder and connect to itself, creating a domain that is not isomorphic to a disc and thus cannot be flattened to 2D. Another issue is that large triangles on the evolving mesh can generate very large patches on the reference mesh with a larger distortion. If cached, these patches may harm placement precision.

Construction of the patches generally relies on well-shaped triangles. The triangles are collected in a breadth-first search over the faces; if the faces are degenerated or considerably vary in size, the resulting patch P_{3D} may not be as round as intended. In the best case, this causes more distortion in the resulting planar domain. In the worst case, the patch can possibly contain holes after merging the individual neighborhoods $\mathcal{N}_i$ due to the deformed circular regions. The patch cannot be flattened then because it contains several boundaries. One improvement of this problem is to cut *ear triangles* in each circular neighborhood $\mathcal{N}_i$. Ear triangles are on the boundary and have only one adjacent triangle.

In all erroneous cases, the embedding cannot be created and relocation of the vertex is aborted. It is, however, most likely that these vertices are relocated in a subsequent iteration because nearby patches that have successfully been created and cached may contain the region required to relocate the vertex.

Patch flattening: The constructed patch P_{3D} is now flattened to a planar domain by parameterizing the 3D patch, yielding a planar patch P_{2D}. To start, the boundary vertices of the patch are mapped onto the unit circle in the plane, where the straight distances between the vertices on the circle are proportional to the distances in the 3D patch. The flattened patch boundary vertices are then used as seed for parameterization of the remaining vertices. To do this, each interior vertex is expressed as a combination of neighboring vertices using Mean Value Coordinates developed by Floater in [Flo03]. Mean Value Coordinates are very efficient to compute, create weights that smoothly depend on the neighbors and always produce a valid embedding that is free from fold-overs. The issue of surface parameterization is elaborated in Section 4.6.

Finding the new reference: Given a new vertex position (f_e, b_e) on the evolving mesh, the three vertex references $(f_1, b_1), (f_2, b_2), (f_3, b_3)$ of f_e and a planar patch that contains the faces f_1, f_2 and f_3, the algorithm proceeds as follows: Three vertex positions $\bar{v}_1, \bar{v}_2, \bar{v}_3$ in 2D are calculated using the barycentric coordinates b_i together with the corresponding faces f_i in the planar domain. These positions correlate with the vertex positions of f_e on the evolving mesh. To get the new vertex position $\bar{v}$ in the planar domain that corresponds with the centroid, the positions $\bar{v}_i$ are used together with the barycentric coordinate b_e.

The new position $\bar{v}$ is located within some yet unknown face of P_{2D}, which needs to be traced. A face search based on barycentric coordinates is performed: The search starts at an arbitrary face f_x and analyzes the barycentric coordinate b_x of $\bar{v}$ with respect to f_x. If at least one component of b_x is negative, the search advances in the direction that corresponds to the smallest negative component of b_x, see Figure 5. Otherwise, if all components are positive, f_x contains $\bar{v}$. This completes the vertex relocation: The new vertex reference (f_r, b_r) is given by (f_x, b_x) and the position in 3D is computed.

4.5.5 Increasing Efficiency

Efficiency is seriously a matter since every vertex of the evolving mesh is relocated in each iteration of the relaxation. Typically several Lloyd iterations are required in order to obtain a precise, isotropic sampling of the domain. A naive approach is finally compelled to compute a surface patch for each vertex relocation, which is an expensive operation, but the actual algorithm performs with almost no memory.

The efficiency of the relaxation is drastically improved by employing a caching system that keeps patches as long as memory consumption is acceptable. One possible and simple caching system is to use a FIFO that keeps already created patches. New patches are only created if the FIFO lacks an existing patch that contains the three faces required for relocation. The performance gain is further improved by pushing cache hits to the front of the FIFO. If combined with a cache-aware vertex ordering as described in Section 4.2.2, this strategy performs very well even with a small amount of available cache memory. To limit the amount of memory, the elements with earliest access time are popped from the FIFO.

The relaxation procedure is also generally able to operate in parallel. In each iteration, the vertices are divided into disjoint sets which are separately processed in threads. The situation is, however, demanding in combination with caching. Experiments show that the FIFO approach is not suited because every cache hit also involves a cache write to update the FIFO, and thread exclusions prevent proper parallelism.

A different caching scheme is to build a data structure that maintains a list of patches for each face of the reference mesh. A newly created patch extends the lists of all faces contained in the patch. The lookup operation is very efficient with this technique: The lists corresponding to the three given faces are inspected and the first patch that exists in all lists is taken. Since every face contains only a small number of patches

in average, there is no need to push cache hits to the front of the lists; this obviates a write access for each cache hit and allows for parallel operation. Removal of old patches is, however, not as easy as with the FIFO approach. A timer for each patch is maintained and every cache hit increases the global time and updates the patch timer. If the memory limit is reached, a fixed percentage of the cached patches with earliest access time are removed in a single pass over the data structure. Although the scheme is CPU efficient, the data structure alone requires a considerable amount of memory, especially for large reference meshes.

4.6 Parameterization Techniques

Surface parameterization is defined as mapping a mesh embedded in 3D onto a suitable target domain. The problem does not state any restriction regarding the mesh topology or the target domain. Some authors propose to map meshes to their natural domain, e.g. to use a spherical parameterization [CGS03] for closed, genus-0 meshes, which are topologically equivalent to a sphere. However, in this thesis, planar parameterizations are of particular interest, and the problem of surface parameterization is then formulated as follows: Given a mesh, a piecewise linear mapping between the mesh and some planar domain needs to be computed. This planar domain typically is a simple shape, like disk or a rectangle.

Surface parameterization has a lot of applications in the computer graphics such as morphing, mesh completion, texture and normal mapping, mesh editing and remeshing. Several excellent surveys and tutorials cover the different applications, aspects and techniques of surface parameterization, for example [SPR06, FH05, FH02] to cite a few.

Introduction: Planar parameterization techniques focus on minimizing a specific distortion to reduce the error under a certain error metric. Maps that minimize the *angular distortion*, or *shear*, are called *conformal maps*, and the goal is to keep the angles in the parameterization as close as possible to the original angles. Of course, since the angles around a vertex typically don't sum up to 2π, a distortion is almost unavoidable.

Maps that minimize *areal distortion* are called *authalic*. Though authalic parameterizations are achievable, they are not very useful because they allow for high angular and isometric distortions, and should better be used in combination with other conditions.

Another commonly used parameterization goal is the minimization of distance stretching across the mesh using a *stretch metric*, which results in an *isometric map*. A popular technique has been developed by Sander et al. in [SSGH01] for the purpose of texture-mapping progressive meshes. Similar to angle preservation, stretch preserving parameterizations are generally not possible but exist for developable surfaces, i.e. surfaces with zero Gaussian curvature.

Besides distortion, there are other factors that should be considered when choosing a parameterization method for a specific application. For example, a free boundary may considerably reduce the distortion compared to the fixed boundary that is used in many applications. However, computing the free boundary as part of the problem is often much slower to solve and more complicated to implement. Another serious issue is the robustness of the method: Most applications require a bijective mapping, i.e. no flipped triangles and self-intersecting boundaries in the parameter domain, and some techniques don't guarantee bijection. One might also pay attention to the computational complexity; most fixed-boundary parameterizations merely require solving a linear system of equations where the free-boundary setting often involves a non-linear optimization problem.

For the purpose of surface remeshing, any simple, fast and robust technique that minimizes shear is of particular interest for this thesis.

Mathematical formulation: The early papers which address parameterization for computer graphics applications were mostly interested in planar mappings with disk-like topologies. Research dates back to a method presented by W. T. Tutte for the embedding of planar graphs with the title "How to draw a graph" [Tut63]. With his formulations he presented a general framework which is also used by recent methods and directly applies to meshes. Tutte uses a two-stage procedure: First, the vertices of the mesh boundary are mapped to a convex region in 2D, for example a disk or a rectangular region. In the second step, the rest of the vertices are mapped to 2D by solving a linear system of equations.

Let $v_1, \ldots, v_n$ be the unknown interior vertices and $v_{n+1}, \ldots, v_{n+m}$ the boundary vertices in 2D. The 2D positions of the m boundary vertices are calculated in the first stage, for example by placing the vertices on the unit circle where the straight distances between neighboring vertices on the circle are proportional to the distances on the mesh. In the second stage, the following equations for all interior vertices $i \in \{1, \ldots, n\}$ with umbrella neighborhood N_i solve for the new positions $v_1, \ldots, v_n$ by using the fixed vertices $v_{n+1}, \ldots, v_{n+m}$ as boundary conditions $\bar{v}_i$ for the linear system:

$$v_i - \sum_{j \in N_i, j \leq n} \lambda_{ij} v_j = \bar{v}_i \qquad \text{with} \qquad \bar{v}_i = \sum_{j \in N_i, j > n} \lambda_{ij} v_j$$

The coefficients λ_{ij} are normalized weights that depend on the weights w_{ij}:

$$\lambda_{ij} = \frac{w_{ij}}{\sum_{k \in N_i} w_{ik}} \qquad \text{thus} \qquad \sum_{j=1}^{n} \lambda_{ij} = \sum_{j \in N_i} \lambda_{ij} = 1$$

The weights w_{ij} are defined by the particular parameterization method that is used, for example, Tutte employs unit weights $w_{ij} = 1$ if $j \in N_i$, or in words, if (i, j) is an edge in the mesh. If (i, j) is not an edge in the mesh, $w_{ij} = 0$. The linear system of equations $Av = \bar{v}$ can now be obtained by separation of the variables for $v_i = (x_i, y_i)$ and the fixed

right-hand side $\bar{v}_i = (\bar{x}_i, \bar{y}_i)$ and solving the linear system individually for the x and y component, i.e. by solving $Ax = \bar{x}$ and $Ay = \bar{y}$. The system has the following form:

$$\begin{pmatrix} 1 & \cdots & -\lambda_{1n} \\ \vdots & \ddots & \vdots \\ -\lambda_{n1} & \cdots & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \bar{x}_1 \\ \vdots \\ \bar{x}_n \end{pmatrix}$$

It has been proven that if the weights w_{ij} are positive and the resulting matrix A is symmetric, the mapping is bijective [Tut63]. The weights used by Tutte therefore clearly result in a bijective mapping, but they don't preserve any properties of the mesh.

Conformal parameterization: The choice of the weights is critical to achieve a precise vertex placement. Since most remeshing techniques are very sensitive to shear but can tolerate a fair amount of areal distortion and stretch, a conformal mapping is the technique of choice. A noteworthy publication is the *harmonic* or *cotangent mapping* from Eck et al. [EDD+95], who define the weights as follows:

$$w_{ij} = \frac{\cot \alpha_{ij} + \cot \beta_{ij}}{2}$$

However, if the mesh contains obtuse triangles, the weights can be negative and the parameterization contains flipped triangles: The mapping is not bijective. To accommodate this drawback, Floater introduced the *shape-preserving coordinates* [Flo97], which always produce positive, symmetric weights and guarantee a bijective mapping, but they have derivate discontinuities. Later, Floater proposed the *mean-value coordinates* [Flo03], which don't have discontinuities and additionally, they are much easier to compute:

$$w_{ij} = \frac{\tan \gamma_{ij}/2 + \tan \delta_{ij}/2}{\|v_i - v_j\|}$$

The resulting matrix A for the mean-value weights is not symmetric ($w_{ij} \neq w_{ji}$), nevertheless, Floater proved that the coordinates always create a valid embedding.

Solving the linear system: To parameterize the 3D patch to the planar domain, the linear systems $Ax = \bar{x}$ and $Ay = \bar{y}$ need to be solved. The naive approach of storing and solving the linear system results in a gigantic amount of required memory and processing time. For example, if a patch with 1000 inner vertices is to be parameterized, the involved linear system has dimension 1000 and requires about $1000^2 \cdot 4$ bytes storage for the matrix with single floating point precision. Note that practical remeshing examples of large meshes easily generate patches with several thousand vertices.

A closer look at matrix A reveals the sparsity: The diagonal element A_{ii} of row i is 1, and the other elements $A_{ij}, i \neq j$ correspond to the normalized weights λ_{ij}. Only

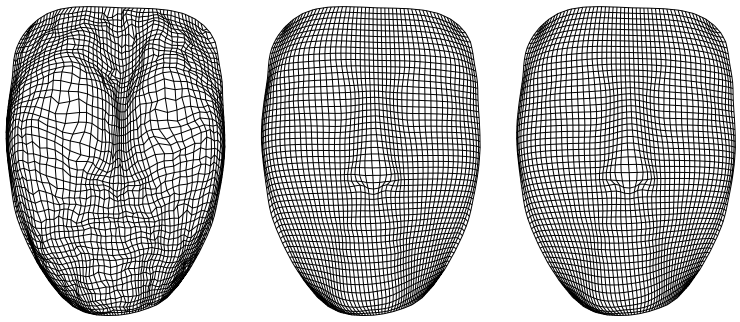


Figure 50: A remesh using different weights to parameterize the mesh. The method from Tutte [Tut63] (left), shape-preserving coordinates [Flo97] (middle), and mean-value coordinates [Flo03] (right), taken from [Flo03].

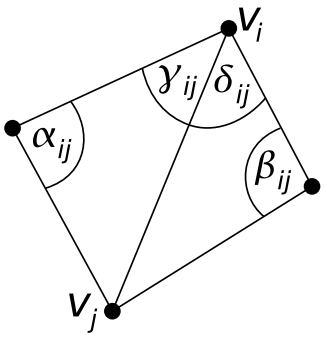


Figure 51: Terminology used for the harmonic coordinates and the mean-value coordinates.

few of these elements A_{ij} with $j \in N_i$ are non-zero, namely those elements in the neighborhood of v_i. For a typical mesh, the average valence of each vertex is 6, and that results in an average of 7 non-zero entries per row. A matrix where the majority of elements are zero is called a *sparse matrix*. Sparse matrices can efficiently be stored and the involved linear system can be handled with solvers that exploit the sparse structure.

Literature distinguishes between different types of linear system solvers. *Dense direct solvers* are the most generic type of solvers: They operate on all kind of matrices and provide accurate results. However, for large linear systems these kind of solvers are inappropriate because they don't utilize special properties of the matrix, e.g. sparsely populated matrices. *Iterative solvers* are able to exploit the sparsity of the matrix; they are simple to implement, fast and therefore de facto standard for solving sparse systems. At some point, however, the speed of convergence towards the solution decreases. *Multigrid solvers* are an extension to iterative solvers to accelerate the convergence by employing a hierarchical solving scheme. But these solvers are expensive to implement and require a fair amount of pre-computation time. *Sparse direct solvers* provide direct accurate results but have high complexity. Nevertheless, as shown in [BBK05], sparse direct solvers can even outperform iterative solvers if proper pre-processing is applied.

Sparse iterative solvers [BBC+94] are of special interest for this thesis because they are widely available in mature linear algebra packages. They compute a sequence $x^{(0)}, x^{(1)}, \ldots$ of approximations towards to solution x^* of the linear system. Simple examples of such solvers are the Jacobi and Gauss-Seidel methods. A much faster convergence is achieved with the method of Conjugate Gradients (CG), which is suited for symmetric positive-definite (spd) matrices. Unfortunately, mean-value coordinates produce a non-symmetric matrix ($w_{ij} \neq w_{ji}$). Instead, an extension named Bi-Conjugate Gradients (BiCG), the stabilized version with improved guarantees (BiCG-STAB) or the generalized minimal residual method (GMRES, [SS86]) can be used.

Another important point is that the linear system needs to be solved twice for different right-hand sides, but the matrix is constant. This additional information can be exploited by investigating pre-computation time in pre-conditioning [Ben02] the matrix as a way of improving the convergence of iterative methods. Simple but powerful pre-conditioners can be obtained by incomplete factorization methods of form $M = \bar{L}\bar{U}$, where $\bar{L}$ and $\bar{U}$ are the incomplete (or *approximate*) triangular LU factors. These incomplete factors are obtained by Gaussian elimination and suitable fill-in discarding strategies, which are surveyed in [Ben02]. These strategies typically differ in the way fill-in positions $\mathcal{S}$ are found, where $\mathcal{S} \subseteq n \times n$ is a subset of all positions in the matrix, usually including the diagonal elements (i, i). Given that subset, a fill-in is only allowed for those positions that are contained in $\mathcal{S}$. For example, the simple no-fill ILU factorization is obtained by choosing $\mathcal{S}$ to coincide with the non-zero positions of the sparse matrix A. Similarly, the same concept applies to the incomplete Cholesky factorization (IC) for spd matrices.

The use of a pre-conditioner significantly reduces the processing time to solve the linear system even for a single right hand side due to increased convergence. For multiple right hand sides, the gain is substantial.

5 Results, Contributions and Issues

A pipeline for relaxation based surface remeshing has been established, in particular by making use of Lloyd relaxation as key concept, to equalize the areas or masses of Voronoi cells. The involved algorithms can be controlled using several parameters: The sampling of the resulting meshes may either be uniform or adaptive with respect to a density function. The flexible design of the density function allows to create smooth transitions between uniform and curvature-adapted samplings. Preservation of feature edges on the mesh makes the algorithms suited for a broad range of models from different application domains.

The remeshing results are demonstrated with several popular meshes. All timings are taken from interactive remeshing sessions performed on an AMD Athlon 64 X2 Dual Core processor with 2600 MHz and 2 GB RAM. A comprehensive table with a statistical breakdown is given at the end of this section.

The Stanford Bunny

The Bunny from the Stanford Scanning Repository is one of the most popular models in computer graphics, and it is obligatory to present results with this model. The Bunny is particularly well-behaved as the mesh is a closed, valid 2-manifold of genus 0 without self-intersections and the like.

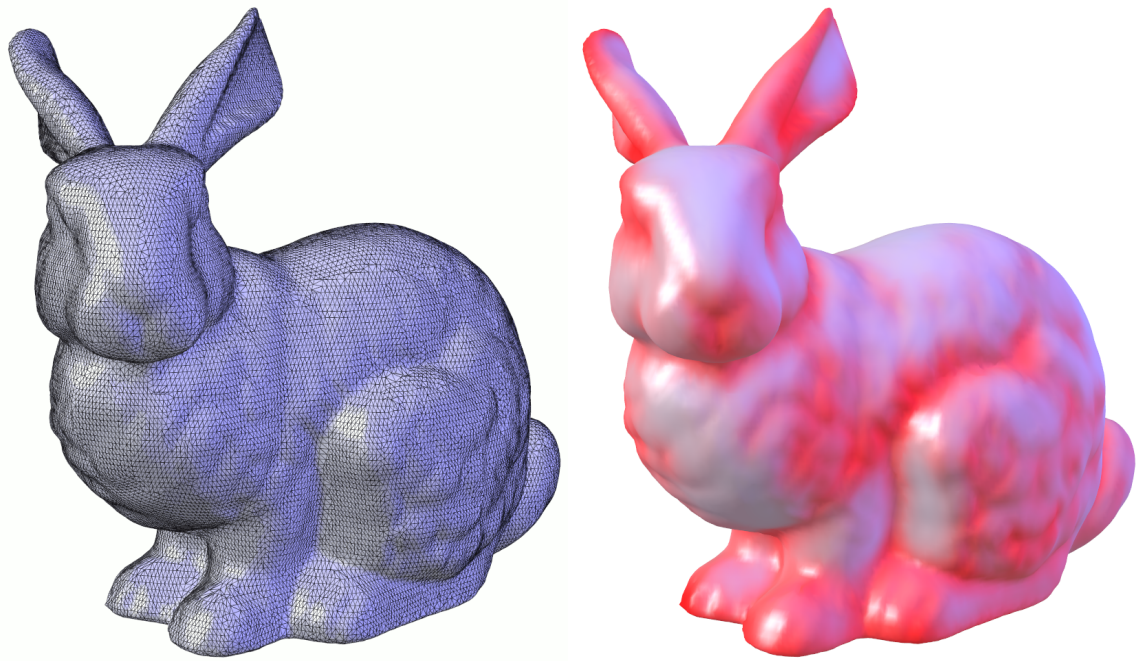


Figure 52: The original Stanford Bunny with about 35k vertices and 70k faces (left) and the discrete curvature estimation on the mesh (right).

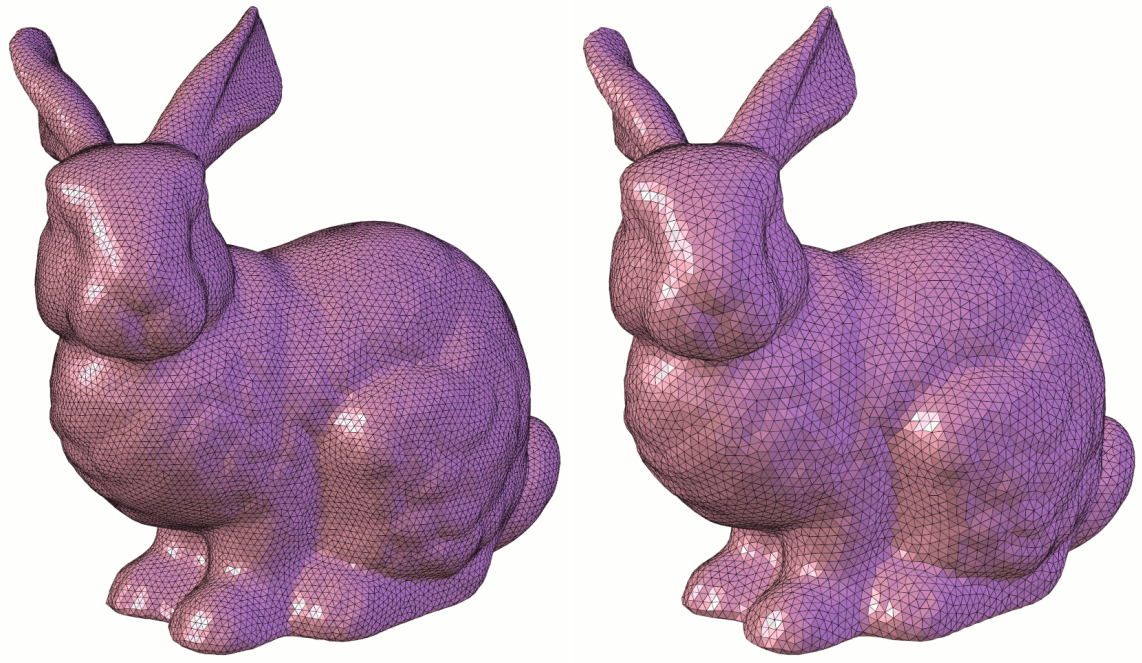


Figure 53: Uniform remesh of the Bunny with 20k vertices (left) and 10k vertices (right).

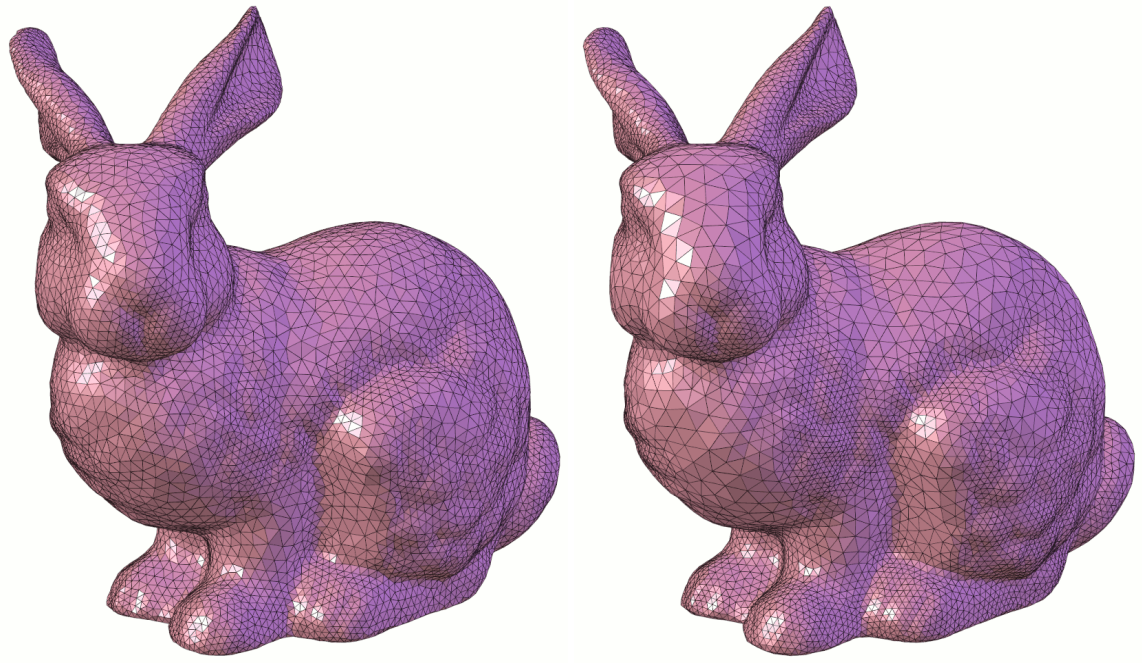


Figure 54: A curvature-adapted remesh using 10k vertices with different parameters for the density field. A greater contrast exponent results in a denser sampling of curved regions.

Marching-Cubes Camel

Surfaces extracted from volume data using the Marching Cubes algorithm are quite common and possess a characteristic sampling in the vicinity of the volumetric grid that leads to badly shaped triangles.

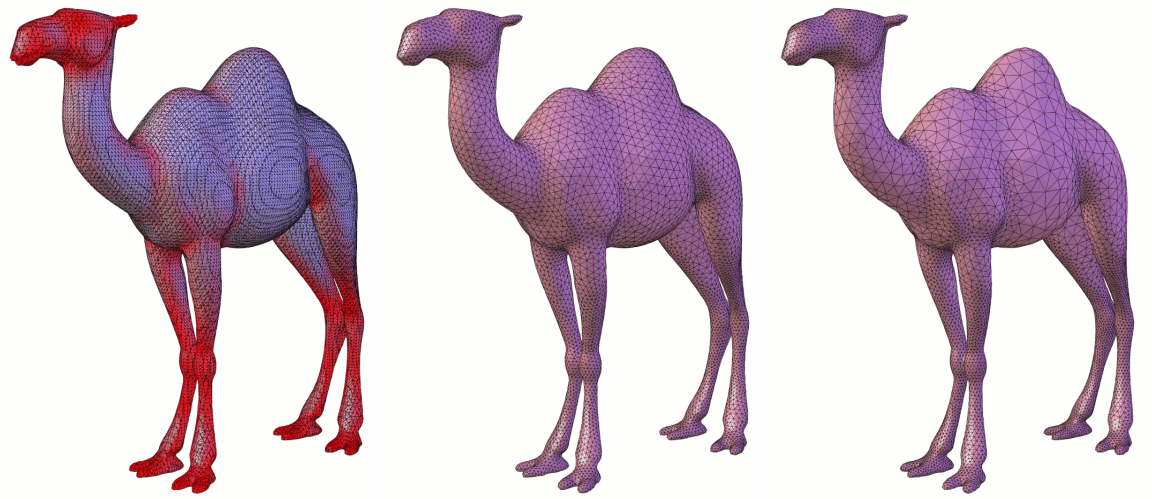


Figure 55: A Marching Cubes extraction of a camel surface with about 35k vertices (left) and curvature-adapted remeshes with 10k vertices and different density functions.

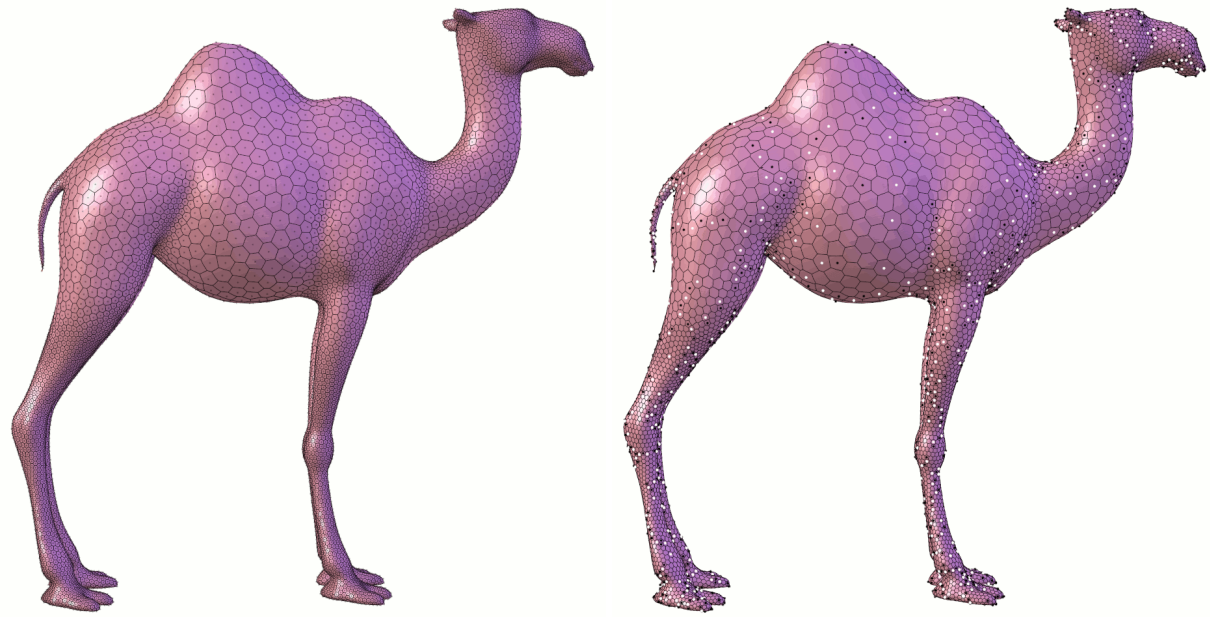


Figure 56: The Voronoi cells on the mesh (left) and indicators for irregular vertices (right). Black dot denotes a vertex valence < 6 and a white dots a valence > 6.

The Stanford Dragon

The method is also able to handle large models in a reasonable amount of time due to optimizations like patch caching and parallelization. Although the original Dragon reconstruction (with about 435k vertices) is in a bad state, the algorithm performs without problems.

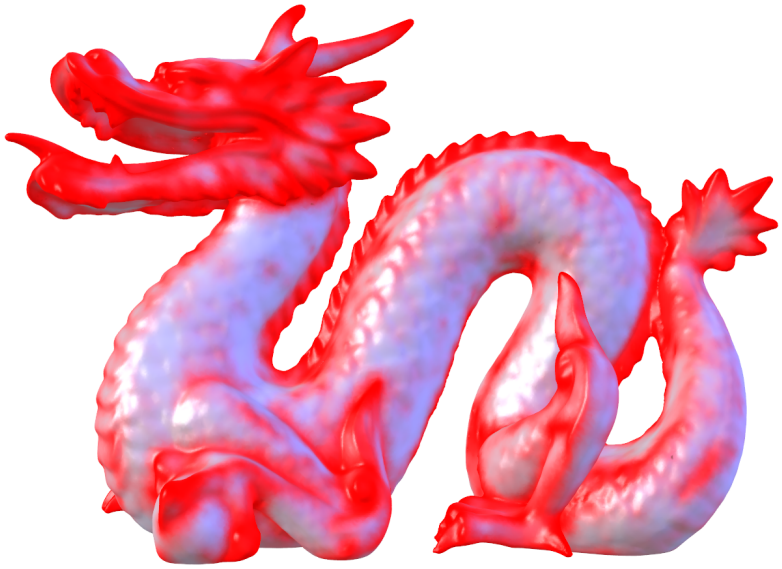


Figure 57: The Stanford Dragon model with the density field.

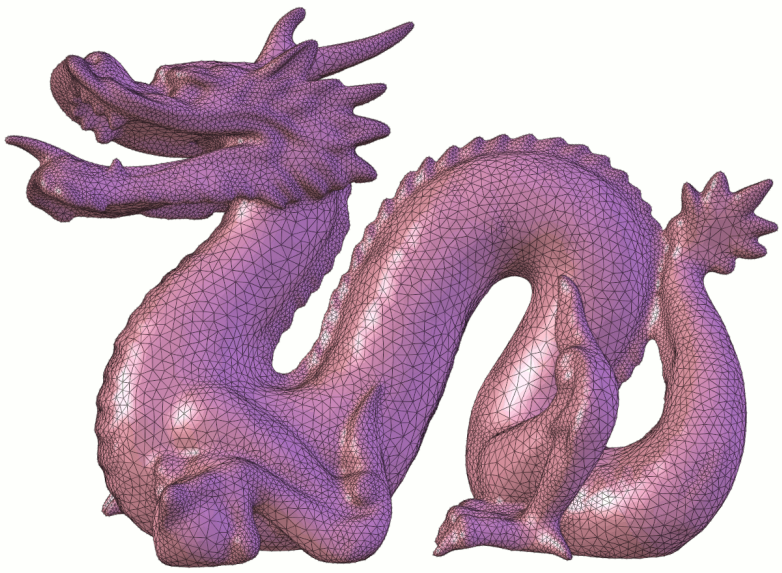


Figure 58: The remeshed Stanford Dragon with 30k vertices.

The Stanford Lucy Statue

The down-sampled version of the Lucy statue with approx. 525k input triangles has been used as reference mesh. The remesh consists of 30k adaptively sampled vertices.

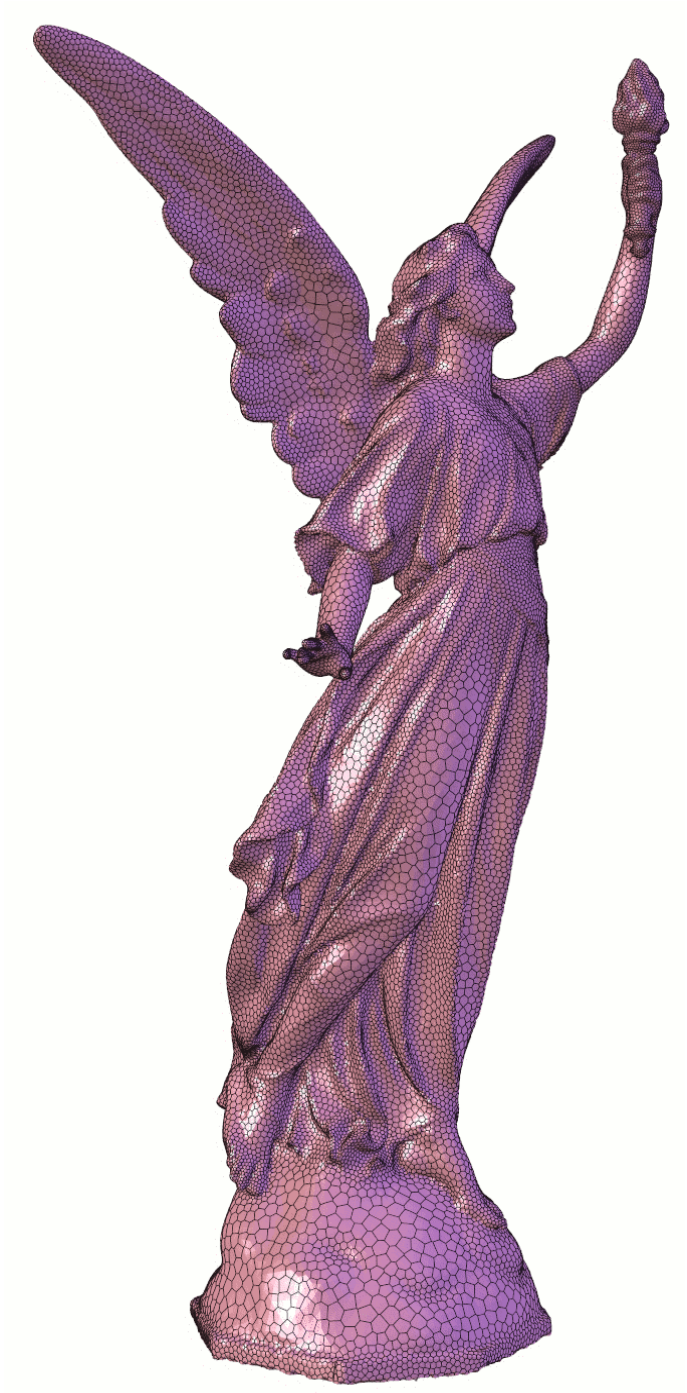


Figure 59: The remeshed Stanford Lucy with the Voronoi diagram on the mesh.

The Egea Model

The initial sparse model with about 8k vertices is first refined by regular subdivision to 132k vertices. The exact vertex budget is then produced by simplification of the refined domain.

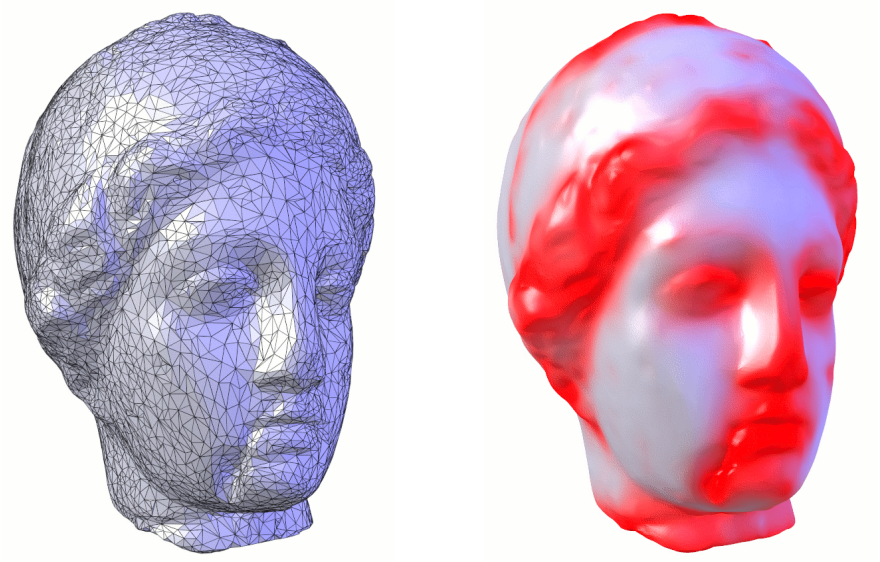


Figure 60: The Egea model and the density field over the mesh.

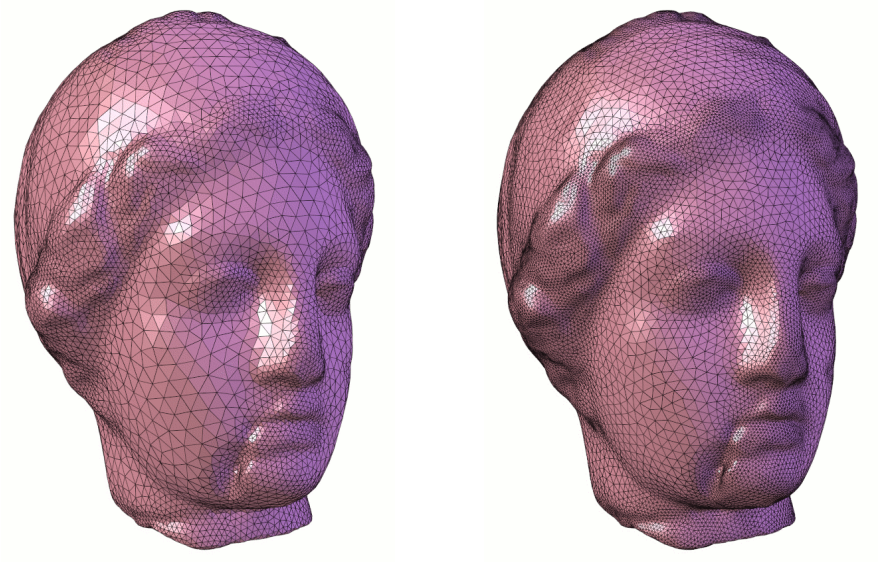


Figure 61: The remeshed Egea model acquired by subdivision, subsequent simplification and Lloyd relaxation. The resulting meshes have 10k and 25k vertices.

Simple Box Model

The initial mesh is oversampled using the mesh slicing procedure with lazy triangulation. To get a nice sampling on the features, the grid is first sliced with 10 planes for each axis, then with 21 planes. As an implementation detail, using the sequence n, $2n+1$, $4n+3$, and so on, guarantees a uniform edge division.

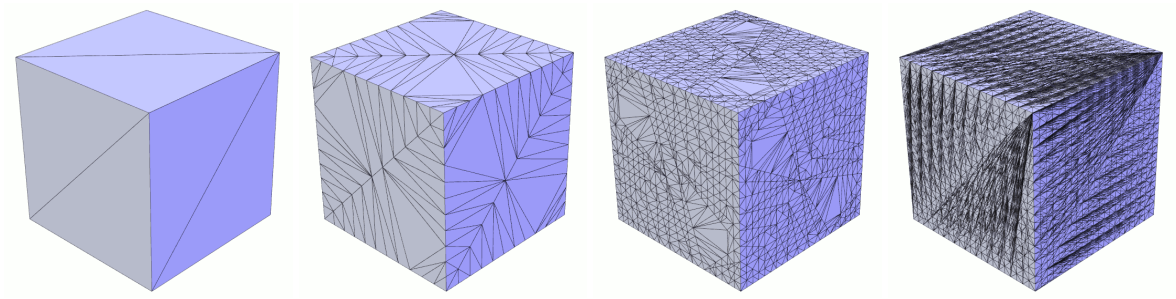


Figure 62: A simple box mesh is refined using mesh slicing with lazy triangulation. The right image shows the result of the same slicing procedure without lazy triangulation.

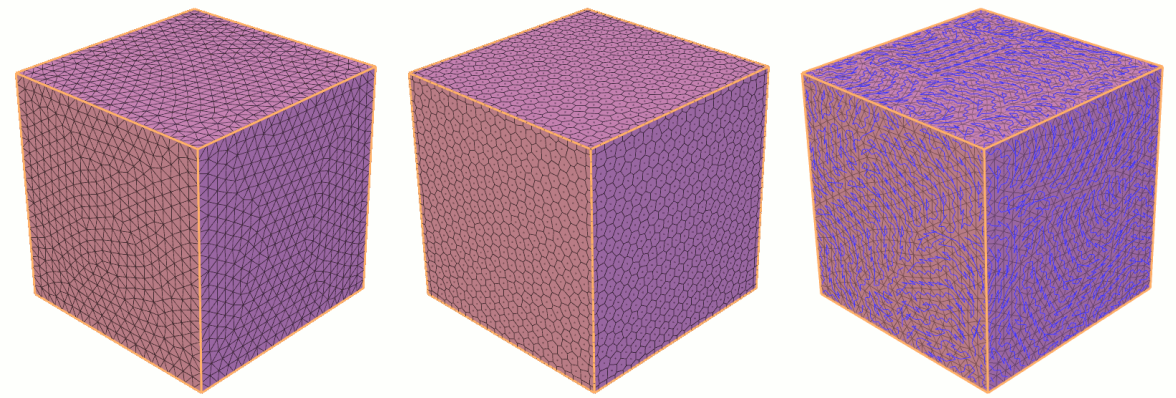


Figure 63: From left to right: The mesh after polishing the sampling, the Voronoi cells on the mesh, and the cache-efficient triangle ordering after remeshing.

The Fandisk Model

The Fandisk is a popular CAD mesh often used for benchmarking purposes. Unlike typical CAD meshes, the sampling of the model is already in very good state. In the first example the vertex positions are simply optimized while feature lines are preserved. The second example demonstrates the piecewise linear surface subdivision technique to create an oversampled version of the original mesh.

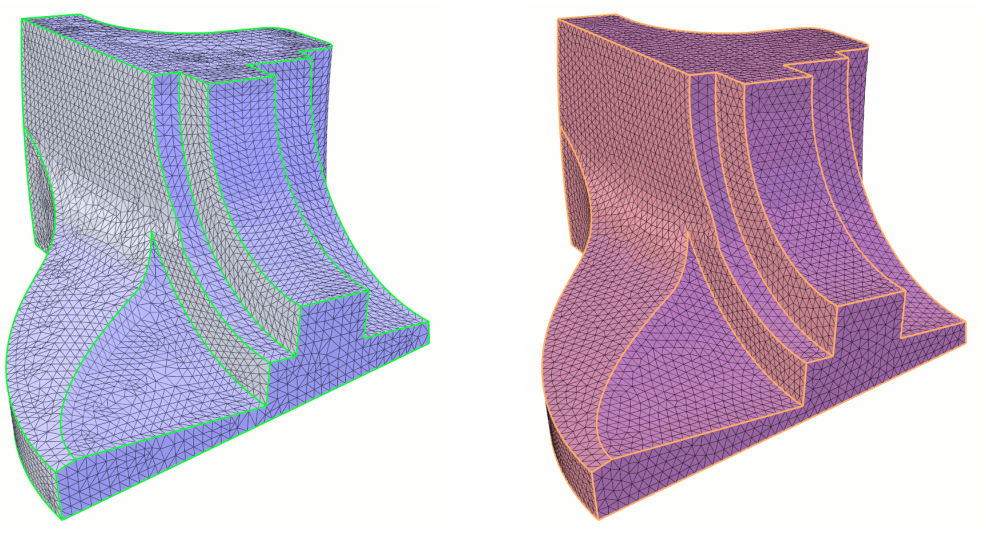


Figure 64: The vertices of the original Fandisk model (left) are polished without changing the vertex budget to obtain a better sampling (right).

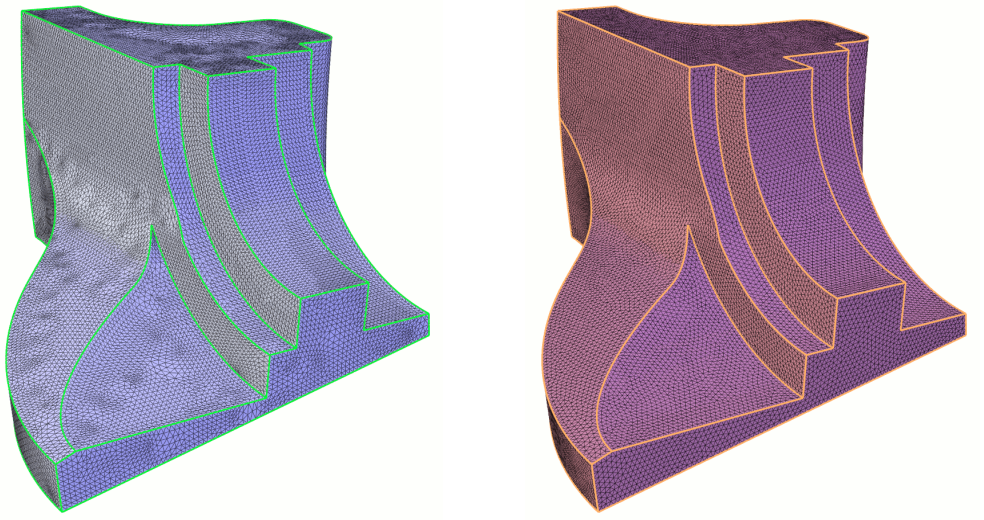


Figure 65: A piecewise smooth subdivision iteration of the original mesh (left) and the resulting isotropic sample placement after Lloyd relaxation (right).

The Joint CAD Model

To show the applicability of the remeshing framework to CAD models, a typical mesh with degenerated faces has been processed. The mesh is sliced with several planes, yielding an oversampled version with 26k vertices, ready for further processing. Subsequent simplification removes the worst triangles and results in a fair mesh.

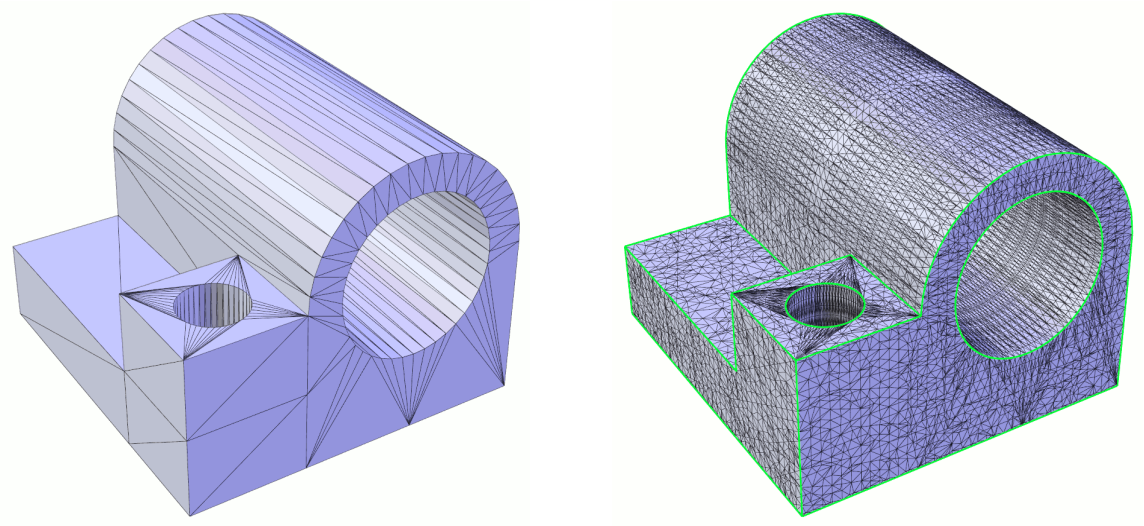


Figure 66: The original Joint CAD mesh (left) and the oversampled version (right).

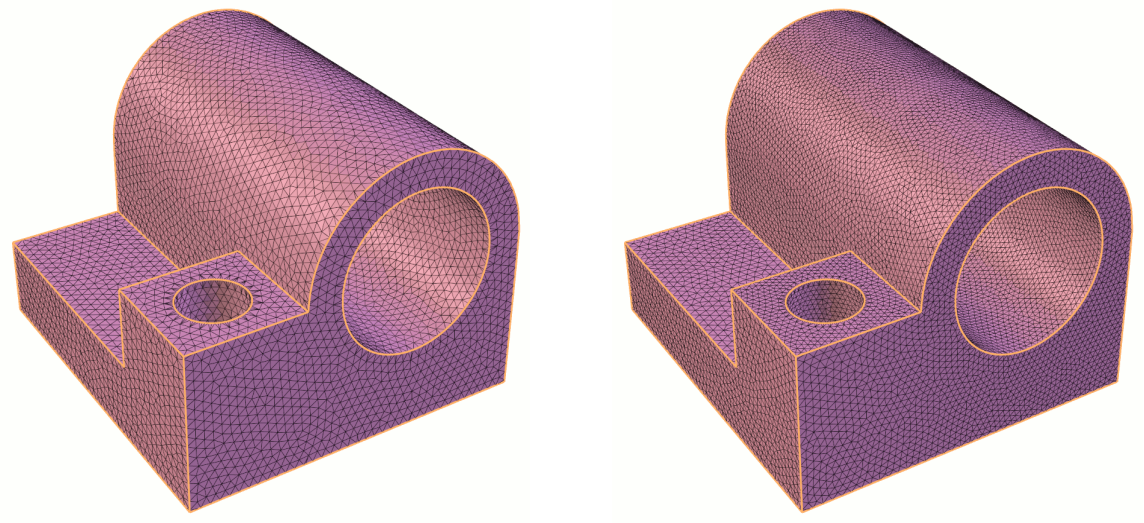


Figure 67: The remesh after applying Lloyd relaxation towards a uniform distribution of samples on the simplified domain with 10k vertices (left) and 20k vertices (right).

Laurent Saboret's Hand

This example shows the improvement of area based remeshing in comparison with Lloyd relaxation alone. An overall uniform simplification is to be remeshed with respect to a density function. Lloyd relaxation alone does not manage to re-distribute the vertices according to the density function in a moderate amount of iterations. A combination of area based remeshing and Lloyd relaxation results in a much better sampling of the surface.

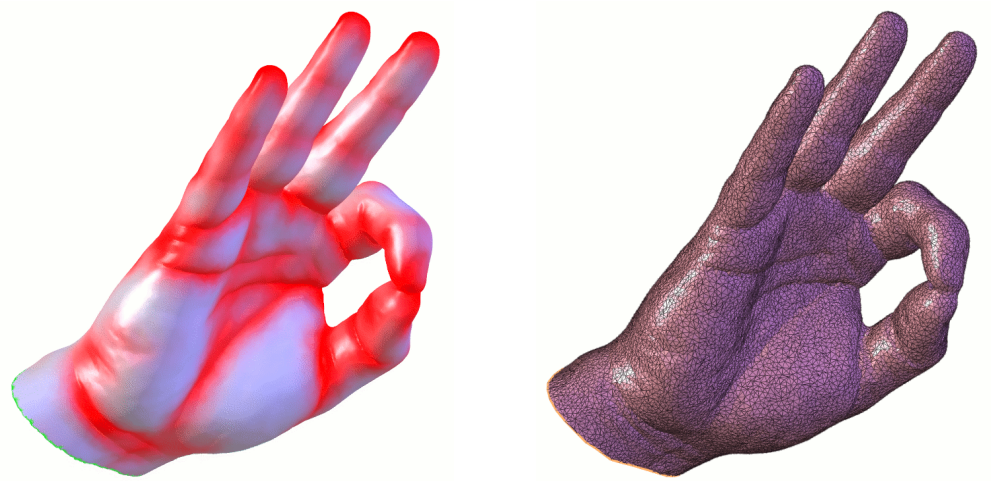


Figure 68: The density field over Laurent's Hand (left) and the uniform simplification to 20k vertices (right).

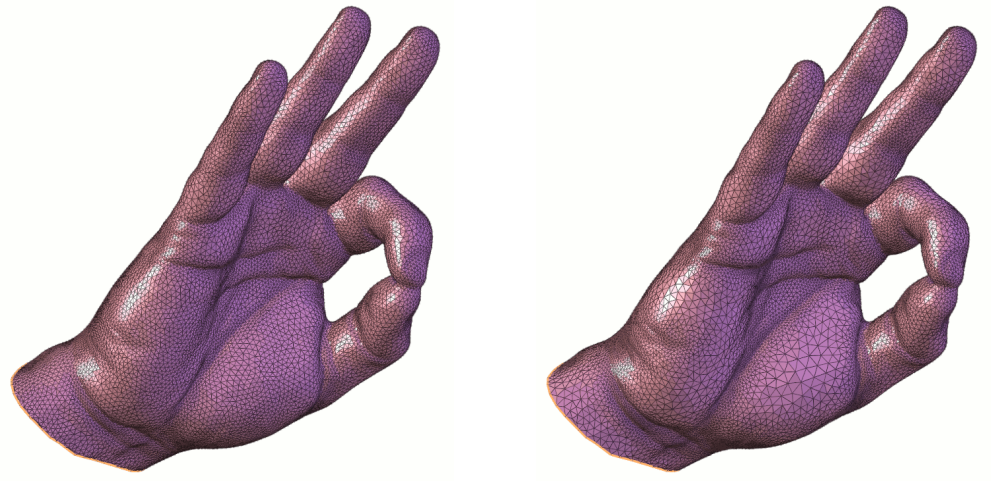


Figure 69: Lloyd relaxation with 50 iterations does not re-distribute the vertices properly (left). Area based remeshing with 20 iterations and subsequent Lloyd relaxation with 20 iterations results in a much better sampling (right).

Statistical Analysis

The following table shows some statistics of the remeshing results. The model and the number of vertices is stated in the first two columns. The time to achieve an exact vertex budget is given under "Simpl.", which is the execution time of the simplification algorithm in seconds. The "Lloyd Relax." column lists the performed iterations and the required time in seconds. The percentage of irregular vertices in the mesh is given in the "Irreg." column. The "Avg. $\angle$" is the average of the minimum angle in each triangle. The error was measured using Metro [CRS98], and it is the maximum Hausdorff distance between the reference mesh and the remesh.

Model	Vertices	Simpl.	Lloyd Relax.		Irreg.	Avg. $\angle$	Error
		(sec)	(iter)	(sec)	(%)	(deg)	(10^{-3})
Bunny (original)	35,286	–	–	–	25.1	37.0	–
Bunny (uniform)	20,000	2.6	50	9.3	19.1	53.2	4.0
Bunny (uniform)	10,000	3.3	50	6.0	24.6	52.7	4.5
Bunny (adapted)	10,000	3.4	50	7.2	28.3	51.1	4.3
Bunny (adapted)	10,000	3.2	50	7.3	29.0	51.1	4.2
Camel (original)	34,546	–	–	–	52.9	32.2	–
Camel (adapted)	10,000	2.7	50	7.5	26.2	52.0	6.9
Camel (adapted)	10,000	1.6	50	8.3	25.6	52.1	7.5
Dragon (original)	435,545	–	–	–	67.5	31.8	–
Dragon (adapted)	30,000	30	50	110	29.2	50.2	7.4
Lucy (original)	262,909	–	–	–	63.7	36.7	–
Lucy (adapted)	30,000	20	50	43	29.4	51.7	2.9
Egea (original)	8,268	–	–	–	74.9	34.7	–
Egea (adapted)	10,000	6.2	100	22	22.0	52.6	3.9
Egea (adapted)	25,000	5.1	100	33	20.8	53.0	2.6
Fandisk (original)	6,475	–	–	–	19.8	43.5	–
Fandisk (uniform)	6,475	–	100	6.4	15.0	52.7	1.4
Fandisk (refined)	25,894	–	–	–	5.0	45.0	–
Fandisk (uniform)	25,894	–	100	21	13.2	53.2	0.7
Joint (original)	221	–	–	–	72.9	9.35	–
Joint (sliced)	25,808	–	–	–	71.2	18.0	–
Joint (uniform)	10,000	2.3	200	19	21.2	52.6	1.2
Joint (uniform)	20,000	1.6	200	30	19.1	53.3	1.2

Table 1: Statistical analysis and comparison of the remeshing results.

Contributions

In this work a remeshing pipeline has been employed that qualifies as generic framework for relaxation based remeshing methods. The technique performs all relocations directly on the surface of the reference mesh, without relying on potentially unstable and computationally expensive projections back to the original surface. The method operates on surfaces of arbitrary genus due to the use of small, overlapping patches, and does not suffer from the drawbacks of global parameterization like cutting and stiching. All operations refer to local geometry only, which makes the treatment particularly fast and stable even for imperfect models.

To make the framework applicable to a wide range of meshes, the involved algorithms are sensitive to piecewise smooth surfaces by taking account of a set of tagged feature edges, which are preserved during the process. Using a flexible design of the density function defined over the original mesh allows for continuous transitions between uniform and heavily adaptive samplings with respect to the surface curvature. The involved algorithms accept a variety of parameters for different remeshing scenarios to make the framework suitable for versatile applications with different demands, especially for those applications that benefit from high regularity in terms of connectivity and shape of the elements.

The practice of transitioning mesh oversampling directly to the surface of the reference mesh enables to treat not only large meshes as the result of a scanning process, but also most CAD models which are particularly hard to remesh. The key ingredient is an improved mesh slicing algorithm with lazy triangulation, to apply oversampling directly to meshes with degenerated triangles and to obtain pleasant intermediate results.

Custom-tailored algorithms facilitate the collaboration between individual stages of the pipeline. For example, the density guided simplification algorithm strives for optimal vertex distribution to support the convergence speed of the Lloyd relaxation.

The algorithms are fast, exploiting multi-threaded execution and a cache aware vertex ordering to stay abreast of recent changes in computer hardware. The timings of the experimental software package outperform most remeshing algorithms while producing quality results on par with the latest developments in the area of surface remeshing.

The algorithms for this thesis have been implemented in a C++ framework and packaged in a geometric library to provide an inter-operable toolset across operating systems. To give the scientific community a tool for isotropic remeshing at hand, a GUI has been created, tying the algorithms together, to provide an exemplary interface to the library. The GUI and the library are both licensed under the GPL and freely available from the Internet.

Although the algorithms perform well for the majority of meshes, there are still several issues open for improvement.

Issues and Open Problems

In this implementation, the relaxation procedure is not performed on vertices that are part of feature creases, thus an initial bad sampling along the crease is not relaxed and remains during the algorithm. To deal with that problem, one has to include the feature creases at the patch level and constrain the relocation. Due to the fact that feature vertices and edges are inherently fixed during the relaxation, the sampling cannot be controlled beyond these lines. The Lloyd relaxation as well as the area-based relaxation are therefore not able to move vertices across these boundaries.

Another serious drawback is the relocation of vertices on top of very thin but perceptually salient crest lines or pinnacles, for example at the Bunny ear or the Camel tail. Initially, these vertices define the characteristic appearance of the crest. If these vertices are relocated orthogonal to the crest line, the ridges are quickly destroyed if no special care is taken. This behavior is often a problem for uniform remeshes if the triangle size is too large for heavily curved regions. Note that defining these regions as special features would effectively prevent these problems but this is generally not a good idea: In principle the surface *is* smooth in these regions and tagging the edges would inhibit the development of a proper sampling. In the current state of the algorithm, no concept for constraining these relocations has been developed. Although it would be possible to forbid certain relocations if the error is rapidly growing, slow and subliminal error accumulation is still possible.

A similar problem may occur during the algorithm when restoring the Delaunay property. Flipping of edges initially oriented along a ridge may notably harm fidelity since edge flips are performed without considering the approximation error of the edge to the reference mesh before and after the flip. Constraining these edge flips is also problematic because the algorithm essentially relies on a valid Delaunay triangulation to construct the Voronoi cells for relocation. Applying fidelity improving edge flips as post-process after remeshing did not turn out to be a success. In fact these edge flips harmed the pleasant appearance of the triangulation and introduced some degenerated triangles.

In case of problematic situations the user needs to experiment with the remeshing parameters to obtain the best results for the specific need. However, during the most remeshing sessions the described issues were not critical and the algorithm performed well for the majority of the meshes.

A Orthogonal distance regression plane

The intention is to calculate a plane that best fits a set of points in three dimensions. There are different kinds of best-fit planes in three dimensions; one strategy minimizes the maximum distance from the points to the plane, another minimizes the sum of squared distances to the plane. The latter plane is called least squares plane and there are also different kinds of. The distances can be measured along one axis only, this is called a regression plane and is basically a generalization of the regression plane for two dimension. However, the most general solution without any specific application in mind is to use yet another regression plane witch uses the orthogonal distances and minimizes the perpendicular distance to the plane. The following calculations are based on the problem of creating an orthogonal distance regression plane.

Derivation

Starting with the distance from a point to a plane, we wish to find a, b, c and d such that we minimize

$$f(a,b,c,d) \quad = \quad \sum_i \left(\frac{|a \cdot x_i + b \cdot y_i + c \cdot z_i + d|^2}{a^2 + b^2 + c^2} \right)$$

If we set the partial derivative with respect to d equal to zero, we can solve for d to get

$$d \quad = \quad -(a \cdot x_0 + b \cdot y_0 + c \cdot z_0)$$

where (x_0, y_0, z_0) is the centroid of the data. This means that the least squares plane contains the centroid. If we substitute it back into the equation for the plane we get

$$a \cdot (x - x_0) + b \cdot (y - y_0) + c \cdot (z - z_0) \quad = \quad 0$$

We can rewrite $f(a,b,c,d)$ like this

$$f(a,b,c) \quad = \quad \sum_i \left(\frac{|a \cdot (x_i - x_0) + b \cdot (y_i - y_0) + c \cdot (z_i - z_0)|^2}{a^2 + b^2 + c^2} \right)$$

Now we are going to switch over to a matrix representation. Let's define v and M such that $f(a,b,c) = f(v) = \frac{v^T \cdot (M^T M) \cdot v}{v^T \cdot v}$.

$$v = \begin{pmatrix} a \\ b \\ c \end{pmatrix} \qquad M = \begin{pmatrix} x_1 - x_0 & y_1 - y_0 & z_1 - z_0 \\ x_2 - x_0 & y_2 - y_0 & z_2 - z_0 \\ \vdots & \vdots & \vdots \\ x_n - x_0 & y_n - y_0 & z_n - z_0 \end{pmatrix}$$

Solution using the Rayleigh Quotient

Define $A = M^T M$. The function $f(v)$ is called the Rayleigh Quotient. It is minimized by the eigenvector of A that corresponds to its smallest eigenvalue. We could compute the eigenvectors of A, but this is not needed. The SVD (singular value decomposition) of M is

$$M = USV^T$$

where S is a diagonal matrix containing the singular values of M, the columns of V are its singular vectors, and U is an orthogonal matrix. This also gives us a decomposition of A as

$$\begin{aligned} A &= M^T \cdot M \\ &= (USV^T)^T \cdot (USV^T) \\ &= (VS^TU^T) \cdot (USV^T) \\ &= VS^2V^T \end{aligned}$$

This decomposition of A diagonalizes the matrix and provides an eigenvector decomposition. It means that the eigenvalues of A are the squares of the singular values of M, and the eigenvectors of A are the singular vectors of M.

The orthogonal least squares 3D plane contains the centroid of the data, and its normal vector is the singular vector of M corresponding to its smallest singular value.

B Calculation of a triangle's circumcircle

The circumcircle of a triangle is the unique circle that contains all the points of the triangle. Thus, the circumcirle's center, the circumcenter, has the same distance to all vertices of the triangle. The circumcircle of a triangle has various interesting and important properties for this thesis. For example, the circumcenter can be used to construct the Voronoi cell of a vertex: The vertices of the Voronoi polygon are the circumcenters of the adjacent faces of the vertex, and connecting these circumcenters result in the Voronoi cell.

In the following an efficient approach for calculating the circumcenter and the circumradius for a given triangle is presented. The vertices of the triangle are gives as

$$v_1 = (x_1, y_1)^T, \quad v_2 = (x_2, y_2)^T, \quad v_3 = (x_3, y_3)^T$$

The circumcircle is the locus of points $v = (x_v, y_v)^T$ with the circumcenter u and circumradius r, and we write:

$$\begin{aligned} |v-u|^2 - r^2 &= 0 \\ |v_i-u|^2 - r^2 &= 0 \quad \text{for } i \in \{1,2,3\} \end{aligned}$$

These equations can also be described as the locus of zeros of the determinant of M,

$$\det M = 0 \quad \Leftrightarrow \quad \det \begin{pmatrix} |v|^2 & v_x & v_y & 1 \\ |v_1|^2 & x_1 & y_1 & 1 \\ |v_2|^2 & x_2 & y_2 & 1 \\ |v_3|^2 & x_3 & y_3 & 1 \end{pmatrix} = 0$$

Using cofactor expansion along the first row yields:

$$S_x = \frac{1}{2} \det \begin{pmatrix} |v_1|^2 & y_1 & 1 \\ |v_2|^2 & y_2 & 1 \\ |v_3|^2 & y_3 & 1 \end{pmatrix}, \qquad S_y = \frac{1}{2} \det \begin{pmatrix} x_1 & |v_1|^2 & 1 \\ x_2 & |v_2|^2 & 1 \\ x_3 & |v_3|^2 & 1 \end{pmatrix}$$

$$a = \det \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{pmatrix}, \qquad b = \det \begin{pmatrix} x_1 & y_1 & |v_1|^2 \\ x_2 & y_2 & |v_2|^2 \\ x_3 & y_3 & |v_3|^2 \end{pmatrix}$$

For a non-degenerate triangle (with a non-zero area), we get

$$\begin{aligned} a|v|^2 - 2Sv - b &= 0 \\ |v - \frac{S}{a}|^2 &= \frac{b}{a} + \frac{|S|^2}{a^2} \end{aligned}$$

with $S = \begin{pmatrix} S_x \\ S_y \end{pmatrix}$. This gives the circumcenter $\frac{S}{a}$ and the circumradius $\sqrt{\frac{b}{a} + \frac{|S|^2}{a^2}}$.

C Area and Centroid of non-uniform Triangles

Having a non-uniform triangle, it is better to speak of the *mass* and the *center of mass* (or *center of gravity, physical barycenter*) instead of *area* and *centroid* (which are rather geometrical than physical properties). The non-uniformity is specified with three density values at the vertices of the triangle, linearly interpolated over the area. The geometric interpretation of this is a polyhedron with triangular base and a perpendicular at each vertex with the density value as height. This geometric solid has five flat faces and nine straight edges.

For the triangle $T := (v_1, v_2, v_3)$ with vertices $v_i := (x_i, y_i)$ and density values d_i at the vertices v_i, the density value of a point $v = \sum_{i=1}^{3} b_i v_i$ inside the triangle (expressed with its barycentric coordinates b_i) is given by $d_v = \sum_{i=1}^{3} b_i d_i$ and we define the density function for the triangle as $d(v) := \sum_{i=1}^{3} b_i(v) d_i$. This clearly is a linear, bivariate function.

To solve the resulting integrals for the triangle's mass and center of mass, the Bernstein-Bézier-Method [LS07] is the perfect mathematical tool for this task. Given non-negative integers i, j, k summing to the polynomial degree d, the Bernstein-Bézier-Form (BB-Form) of the polynomial p with basis polynomials B^d_{ijk} and coefficients c_{ijk} is:

$$p = \sum_{i+j+k=d} c_{ijk} B^d_{ijk} \qquad \text{with} \qquad B^d_{ijk} := \frac{d!}{i!j!k!} b_1^i b_2^j b_3^k$$

This allows to solve the integral expressions using the explicit formula

$$\int_T p(x,y)\, dx\, dy = \frac{A_T}{\binom{d+2}{2}} \sum_{i+j+k=d} c_{ijk}$$

Calculation of the Triangle's Mass

The mass m_T of the triangle T is the solution of the integral

$$m_T = \int_{v \in T} d(v)\, dx\, dy.$$

Since $B^1_{100} = b_1$ (analog for b_2 and b_3), we obtain the BB-Form for $d(v) = \sum_{i+j+k=1} d_{ijk} B^1_{ijk}$.

This enables to efficiently solve the integral

$$m_T = \int_{v \in T} d(v)\, dx\, dy = \frac{A_T}{3} \sum_{i+j+k=1} d_{ijk} = \frac{A_T}{3} \cdot (d_1 + d_2 + d_3)$$

where A_T is the area of the triangle.

Calculation of the Triangle's Center of Mass

The center of mass $c = (c_x, c_y)$ is the solution of the integrals

$$c_x = \frac{1}{m_T} \int_{v \in T} d(v) \cdot x \; dx\, dy \qquad\qquad c_y = \frac{1}{m_T} \int_{v \in T} d(v) \cdot y \; dx\, dy$$

Again, we want to switch to the BB-Form for $d(v) \cdot x$ and $d(v) \cdot y$ to solve the integrals. To do this, the BB-Form for each of the factors $d(v), x$ and y are determined and accordingly multiplied. Since each of the factors is a linear bivariate polynomial, $d(v) \cdot x$ and $d(v) \cdot y$ are quadratic, bivariate polynomials. The BB-Form for each factor is

$$\begin{aligned} d(v) &= \sum_{i+j+k=1} d_{ijk} B^1_{ijk} = d_1 b_1 + d_2 b_2 + d_3 b_3 \\ x &= \sum_{i+j+k=1} x_{ijk} B^1_{ijk} = x_1 b_1 + x_2 b_2 + x_3 b_3 \\ y &= \sum_{i+j+k=1} y_{ijk} B^1_{ijk} = y_1 b_1 + y_2 b_2 + y_3 b_3 \end{aligned}$$

Multiplying $d(v)$ and x yields the following equation (analog for $d(v) \cdot y$):

$$\begin{aligned} d(v) \cdot x &= (d_1 b_1 + d_2 b_2 + d_3 b_3) \cdot (x_1 b_1 + x_2 b_2 + x_3 b_3) \\ &= d_1 x_1 b_1^2 + d_2 x_2 b_2^2 + d_3 x_3 b_3^2 \\ &+ (d_1 x_2 + d_2 x_1) b_1 b_2 + (d_1 x_3 + d_3 x_1) b_1 b_3 + (d_2 x_3 + d_3 x_2) b_2 b_3 \end{aligned}$$

Since $B^2_{200} = b_1^2$ (analog for B^2_{020} and B^2_{002}) and $B^2_{110} = 2 b_1 b_2$ (analog for B^2_{101} and B^2_{011}), substitution of the barycentric coordinates with the basis polynomials yields

$$\begin{aligned} d(v) \cdot x &= d_1 x_1 B^2_{200} + d_2 x_2 B^2_{020} + d_3 x_3 B^2_{002} \\ &+ \frac{1}{2}(d_1 x_2 + d_2 x_1) B^2_{110} + \frac{1}{2}(d_1 x_3 + d_3 x_1) B^2_{101} + \frac{1}{2}(d_2 x_3 + d_3 x_2) B^2_{011} \end{aligned}$$

and we can take the coefficients of B^2_{ijk} to solve the integrals c_x and c_y:

$$\begin{aligned} c_x &= \frac{1}{m_T} \int_{v \in T} d(v) \cdot x \; dx\, dy = \frac{1}{m_T} \cdot \frac{A_T}{\binom{2+2}{2}} \sum_{i+j+k=2} c_{ijk} \\ &= \frac{A_T}{6 m_T} \left(d_1 x_1 + d_2 x_2 + d_3 x_3 + \frac{1}{2}(d_1 x_2 + d_2 x_1) + \frac{1}{2}(d_1 x_3 + d_3 x_1) + \frac{1}{2}(d_2 x_3 + d_3 x_2) \right) \\ &= \frac{A_T}{6 m_T} \left\langle \begin{pmatrix} d_1 + \frac{1}{2} d_2 + \frac{1}{2} d_3 \\ \frac{1}{2} d_1 + d_2 + \frac{1}{2} d_3 \\ \frac{1}{2} d_1 + \frac{1}{2} d_2 + d_3 \end{pmatrix} \middle| \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \right\rangle \end{aligned}$$

D Deriving Area Equalization

Area equalization works by moving the center vertex v to a new position inside the umbrella polygon $P = (v_1, \ldots, v_n)$ such that the triangles $T_1, \ldots, T_n$ have areas as equal as possible to each other. Thus, the task is to find a new position $v = (x, y)$:

$$(x,y) \;=\; \arg\min_{(x,y)} \sum_{i=1}^{n} \left(A_i(x,y) - \mu_i A\right)^2 \qquad \text{with} \qquad A_i(x,y) \;=\; \begin{vmatrix} x_i & y_i & 1 \\ x_{i+1} & y_{i+1} & 1 \\ x & y & 1 \end{vmatrix}$$

The area A of the umbrella polygon P may be computed as $A = \sum A_i(0,0)$. The task is now posed as optimization problem and we seek for the global minimum of $f(x,y)$. Since the function is quadratic, the only local minimum is also the global minimum.

$$\begin{aligned} f(x,y) &= \sum_{i=1}^{n} \left(A_i(x,y) - \mu_i A\right)^2 \;=\; \sum_{i=1}^{n} g_i(x,y) \\ g_i(x,y) &= \left(A_i(x,y) - \mu_i A\right)^2 \;=\; (x_i y_{i+1} + y_i x + x_{i+1} y - x y_{i+1} - y x_i - x_{i+1} y_i - \mu_i A)^2 \end{aligned}$$

To continue, $g_i(x,y)$ is differentiated using the chain rule.

$$\begin{aligned} g_i(x,y) &= u_i(x,y) \circ v_i(x,y) \;=\; v_i(x,y)^2 \\ v_i(x,y) &= x_i y_{i+1} + y_i x + x_{i+1} y - x y_{i+1} - y x_i - x_{i+1} y_i - \mu_i A \end{aligned}$$

The resulting derivatives are really simple:

$$\frac{\partial v_i}{\partial x} = y_i - y_{i+1} \qquad \text{and} \qquad \frac{\partial v_i}{\partial y} = x_{i+1} - x_i$$

After applying the chain rule, we get:

$$\begin{aligned} \frac{\partial g_i}{\partial x} &= 2\left((y_i - y_{i+1})(x_i y_{i+1} - x_{i+1} y_i - \mu_i A) + x(y_i - y_{i+1})^2 + y(y_i - y_{i+1})(x_{i+1} - x_i)\right) \\ \frac{\partial g_i}{\partial y} &= 2\left((x_{i+1} - x_i)(x_i y_{i+1} - x_{i+1} y_i - \mu_i A) + x(y_i - y_{i+1})(x_{i+1} - x_i) + y(x_{i+1} - x_i)^2\right) \end{aligned}$$

Finally, the linear system is obtained by separating the constants from the variables x, y and summing over the terms, i.e.

$$\underbrace{\begin{pmatrix} \sum_{i=1}^{n} \bar{y}_i^2 & \sum_{i=1}^{n} \bar{x}_i \bar{y}_i \\ \sum_{i=1}^{n} \bar{y}_i \bar{x}_i & \sum_{i=1}^{n} \bar{x}_i^2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}}_{\text{variable}} \;=\; \underbrace{\begin{pmatrix} -\sum_{i=1}^{n} \bar{y}_i (x_i y_{i+1} - x_{i+1} y_i - \mu_i A) \\ -\sum_{i=1}^{n} \bar{x}_i (x_i y_{i+1} - x_{i+1} y_i - \mu_i A) \end{pmatrix}}_{\text{const.}}$$

with $\bar{x}_i = \frac{\partial v}{\partial y} = x_{i+1} - x_i$ and $\bar{y}_i = \frac{\partial v}{\partial x} = y_i - y_{i+1}$.

References

[AB98] Nina Amenta and Marshall Bern. Surface reconstruction by voronoi filtering. In *SCG '98: Proc. of the fourteenth annual symposium on Computational geometry*, pages 39–48, New York, NY, USA, 1998. ACM Press.

[ABCO+01] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Point set surfaces. In *VIS '01: Proc. of the conference on Visualization '01*, pages 21–28, Washington, DC, USA, 2001. IEEE Computer Society.

[ABK98] Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *SIGGRAPH '98: Proc. of the 25th annual conf. on Computer Graphics and interactive techniques*, pages 415–421. ACM Press, 1998.

[ACDL00] Nina Amenta, Sunghee Choi, Tamal K. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. In *SCG '00: Proc. of the 16th annual symposium on Comp. Geom.*, pages 213–222. ACM Press, 2000.

[ACSD+03] Pierre Alliez, David Cohen-Steiner, Olivier Devillers, Bruno Lévy, and Mathieu Desbrun. Anisotropic polygonal remeshing. *ACM Trans. Graph.*, 22(3):485–493, July 2003.

[AD01] Pierre Alliez and Mathieu Desbrun. Progressive compression for lossless transmission of triangle meshes. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 195–202, New York, NY, USA, 2001. ACM.

[AdVDI03] Pierre Alliez, Éric Colin de Verdière, Olivier Devillers, and Martin Isenburg. Isotropic surface remeshing. In *Proceedings of Shape Modeling International*, pages 49–58, 2003.

[AFRS03] Marco Attene, Bianca Falcidieno, Jarek Rossignac, and Michela Spagnuolo. Edgesharpener: recovering sharp features in triangulations of non-adaptively remeshed surfaces. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 62–69, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[AK04] Nina Amenta and Yong Joo Kil. Defining point-set surfaces. In *SIGGRAPH '04: ACM SIGGRAPH Papers*, pages 264–270, 2004.

[Aur91] F. Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3), 1991.

[BB97] Fausto Bernardini and Chandrajit L. Bajaj. Sampling and reconstructing manifolds using alpha-shapes. In *In Proc. 9th Canad. Conf. Comput. Geom*, pages 193–198, 1997.

[BBC+94] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear*

Systems: Building Blocks for Iterative Methods, 2nd Edition. SIAM, Philadelphia, PA, 1994.

[BBCS99] F. Bernardini, C. L. Bajaj, J. Chen, and D. R. Schikore. Automatic reconstruction of 3d cad models from digital scans. *International Journal of Computational Geometry and Applications*, 9(4/5):327–369, 1999.

[BBK05] Mario Botsch, David Bommes, and Leif Kobbelt. Efficient linear system solvers for mesh processing. volume 3604, pages 62–83. 2005.

[BBX95] C. L. Bajaj, F. Bernardini, and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3d scans. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 109–118, ACM New York, NY, USA, 1995.

[Ben02] Michele Benzi. Preconditioning techniques for large linear systems: a survey. *Journal of Computational Physics*, 182(2):418–477, 2002.

[BG01] Alexander Bogomjakov and Craig Gotsman. Universal rendering sequences for transparent vertex caching of progressive meshes. In *GRIN'01: No description on Graphics interface 2001*, pages 81–90, Toronto, Ont., Canada, Canada, 2001. Canadian Information Processing Society.

[BHP06] Ragnar Bade, Jens Haase, and Bernhard Preim. Comparison of fundamental mesh smoothing algorithms for medical surface models. In *In Simulation und Visualisierung (2006)*, pages 289–304, 2006.

[BK01] Mario Botsch and Leif P. Kobbelt. A robust procedure to eliminate degenerate faces from triangle meshes. In *VMV '01: Proceedings of the Vision Modeling and Visualization Conference 2001*, pages 283–290. Aka GmbH, 2001.

[BKBH07] Matthew Bolitho, Michael Kazhdan, Randal Burns, and Hugues Hoppe. Multilevel streaming for out-of-core surface reconstruction. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 69–78, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[BO03] J. D. Boissonnat and S. Oudot. Provably good surface sampling and approximation. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 9–18, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[BR02] Fausto Bernardini and Holly E. Rushmeier. The 3d model acquisition pipeline. *Computer Graphics Forum*, 21(2):149–172, 2002.

[BV91] R.M. Bolle and B.C. Vemuri. On three-dimensional surface reconstruction methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(1):1–13, 1991.

[CC78] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350–355, November 1978.

[CGPZ05] Frédéric Cazals, Joachim Giesen, Mark Pauly, and A. Zomorodian. Conformal alpha shapes. In *Symposium on Point-Based Graphics*, pages 55–62, Stony Brook, NY, USA, 2005.

[CGS03] Xianfeng Gu Craig Gotsman and Alla Sheffer. Fundamentals of spherical parameterization for 3d meshes. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 358–363, New York, NY, USA, 2003. ACM.

[Che93] L. Paul Chew. Guaranteed-quality mesh generation for curved surfaces. In *SCG '93: Proceedings of the ninth annual symposium on Computational geometry*, pages 274–280, New York, NY, USA, 1993. ACM.

[CL96] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312, New York, NY, USA, 1996. ACM.

[CM91] Yang Chen and Gérard Medioni. Object modeling by registration of multiple range images. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 3, pages 2724–2729, 1991.

[CMS98] Paolo Cignoni, Claudio Montani, and Roberto Scopigno. A comparison of mesh simplification algorithms. *Computers and Graphics*, 22(1):37–54, 1998.

[CRS98] Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.

[DFG99] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal voronoi tessellations: Applications and algorithms. *SIAM Review*, 41(4):637–676, 1999.

[DG01] Tamal K. Dey and Joachim Giesen. Detecting undersampling in surface reconstruction. In *SCG '01: Proceedings of the seventeenth annual symposium on Computational geometry*, pages 257–263, New York, NY, USA, 2001. ACM.

[DG03] Tamal K. Dey and Samrat Goswami. Tight cocone: a water-tight surface reconstructor. In *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 127–134, New York, NY, USA, 2003. ACM Press.

[DG04] Tamal K. Dey and Samrat Goswami. Provable surface reconstruction from noisy samples. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 330–339, New York, NY, USA, 2004. ACM.

[DGH01] Tamal K. Dey, Joachim Giesen, and James Hudson. Delaunay based shape reconstruction from large data. In *PVG '01: Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics*, pages 19–27, Piscataway, NJ, USA, 2001. IEEE Press.

[DHKL01] Nira Dyn, Kai Hormann, Sun-Jeong Kim, and David Levin. Optimizing 3d triangulations using discrete curvature analysis. *Mathematical Methods for Curves and Surfaces: Oslo 2000*, pages 135–146, 2001.

[DLG90] Nira Dyn, David Levin, and John A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9(2):160–169, 1990.

[Doo78] D. Doo. A subdivision algorithm for smoothing down irregularly shaped polyhedrons. In *Proceedings on Interactive Techniques in Computer Aided Design*, pages 157–165, 1978.

[DS78] D. Doo and M. Sabin. Behavior of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10(6):356–360, 1978.

[EDD+95] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 173–182, New York, NY, USA, 1995. ACM.

[Ede92] Herbert Edelsbrunner. Weighted alpha shapes. Technical report, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1992.

[EHP02] Jeff Erickson and Sariel Har-Peled. Optimally cutting a surface into a disk. In *SCG '02: Proceedings of the eighteenth annual symposium on Computational geometry*, pages 244–253, New York, NY, USA, 2002. ACM.

[ELPZ97] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Zeevi. The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing*, 6(9):1305–1315, 1997.

[EM94] Herbert Edelsbrunner and Ernst P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, 1994.

[ES04] Carlos Hernández Esteban and Fancis Schmitt. Silhouette and stereo fusion for 3d object modeling. *Computer Vision and Image Understanding*, 96(3):367–392, 2004.

[FH02] M. S. Floater and K. Hormann. Parameterization of triangulations and unorganized points. *Tutorials on Multiresolution in Geometric Modelling*, pages 287–315, 2002.

[FH05] Michael S. Floater and Kai Hormann. Surface parameterization: a tutorial and survey. In N. A. Dodgson, M. S. Floater, and M. A. Sabin, editors, *Advances in multiresolution for geometric modelling*, pages 157–186. Springer Verlag, 2005.

[Flo97] Michael S. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14:231–250, 1997.

[Flo03] Michael S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20, 2003.

[GCS06] Michael Goesele, Brian Curless, and Steven M. Seitz. Multi-view stereo revisited. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2402–2409, Washington, DC, USA, 2006. IEEE Computer Society.

[GH97] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[GH98] Michael Garland and Paul S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 263–269, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[GSC+07] Michael Goesele, Noah Snavely, Brian Curless, Hugues Hoppe, and Steven M. Seitz. Multi-view stereo for community photo collections. In *Proceedings of the 11th International Conference on Computer Vision (ICCV 2007)*, pages 265–270, Rio de Janeiro, Brazil, 2007. IEEE.

[Har98] E. Hartmann. A marching method for the triangulation of surfaces. *The Visual Computer*, 14(3):95–108, 1998.

[HDD+92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *ACM Computer Graphics*, 26(2):71–78, 1992.

[HDD+94] Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Hubert Jin, John McDonald, Jean Schweitzer, and Werner Stuetzle. Piecewise smooth surface reconstruction. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 295–302, New York, NY, USA, 1994. ACM.

[Hop96] Hugues Hoppe. Progressive meshes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108, New York, NY, USA, 1996. ACM.

[Hop99] Hugues Hoppe. Optimization of mesh locality for transparent vertex caching. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 269–276, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[JBS98] E. Mouaddib J. Batlle and J. Salvi. Recent progress in coded structured light as a technique to solve the correspondence problem: A survey. *PR*, 31(7):963–982, July 1998.

[Kaz05] Michael Kazhdan. Reconstruction of solid models from oriented point sets. In *SGP '05: Proceedings of the third Eurographics symposium on Geometry processing*, page 73, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

[KBH06] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 61–70, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[KBSS01] L.P. Kobbelt, M. Botsch, U. Schwanecke, and H.P. Seidel. Feature sensitive surface extraction from volume data. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 57–66. ACM New York, NY, USA, 2001.

[KFU+09] Thomas Kalbe, Simon Fuhrmann, Stefan Uhrig, Frank Zeilfelder, and Arjan Kuijper. A new projection method for point set surfaces. In *Proceedings of the 2009 Eurographics*, volume 28/2, pages 77–80, Munich, Germany, 2009. Eurographics Association.

[KLS03] Andrei Khodakovsky, Nathan Litke, and Peter Schröder. Globally smooth parameterizations with low distortion. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 350–357, New York, NY, USA, 2003. ACM.

[Kol05] Ravikrishna Kolluri. Provably good moving least squares. In *SIGGRAPH '05: ACM SIGGRAPH Courses*, page 213, 2005.

[KS01] Tasso Karkanis and A. James Stewart. High quality, curvature dependent triangulation of implicit surfaces. In *IEEE COmputer Graphics and Applications*, pages 60–69, 2001.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169, New York, NY, USA, 1987. ACM Press.

[Lev98] David Levin. The approximation power of moving least-squares. *Math. of Computation*, 67(224):1517–1531, 1998.

[Llo82] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[Loo87] Charles Loop. Smooth subdivision surfaces based on triangles. Master Thesis, Department of Mathematics, University of Utah, August 1987.

[LPC+00] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3d scanning of large statues. In *Siggraph 2000, Computer Graphics Proceedings*, pages 131–144. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[LS07] Ming-Jun Lai and Larry L. Schumaker. *Spline Functions on Triangulations*. Encyclopedia of Mathematics and Its Applications 110. Cambridge University Press, New York, 2007.

[LY06] Gang Lin and Thomas P. Y. Yu. An improved vertex caching scheme for 3d mesh rendering. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):640–648, 2006.

[MD03a] Carsten Moenning and Neil A. Dodgson. Fast marching farthest point sampling. In *Proc. EUROGRAPHICS 2003*, 2003.

[MD03b] Carsten Moenning and Neil A. Dodgson. Fast marching farthest point sampling for implicit surfaces and point clouds. *Computer Laboratory Technical Report*, 565, 2003.

[Mil04] Gary L. Miller. A time efficient delaunay refinement algorithm. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 400–409, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[MM97] Robert Mencl and Heinrich Müller. Interpolation and approximation of surfaces from three-dimensional scattered data points. In *DAGSTUHL '97: Proceedings of the Conference on Scientific Visualization*, page 223, Washington, DC, USA, 1997. IEEE Computer Society.

[MSR07] Evgeni Magid, Octavian Soldea, and Ehud Rivlin. A comparison of gaussian and mean curvature estimation methods on triangular meshes of range image data. *Computer Vision and Image Understanding*, 107(3):139–159, 2007.

[MW99] Andrey A. Mezentsev and Thomas Woehler. Methods and algorithms of automated cad repair for incremental surface meshing. In *Proc. 8th Int. Meshing Roundtable, Sandia report SAND 99-2288*, pages 299–309, 1999.

[PC06] Gabriel Peyré and Laurent D. Cohen. Geodesic remeshing using front propagation. *International Journal of Computer Vision*, 69(1):145–156, 2006.

[Rup95] Jim Ruppert. A delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. Algorithms*, 18(3):548–585, 1995.

[Rus04] Szymon Rusinkiewicz. Estimating curvatures and their derivatives on triangle meshes. In *3DPVT '04: Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium*, pages 486–493, Washington, DC, USA, 2004. IEEE Computer Society.

[SAG03] Vitaly Surazhsky, Pierre Alliez, and Craig Gotsman. Isotropic remeshing of surfaces: a local parameterization approach. In *Proceedings of 12th International Meshing Roundtable*, pages 215–224, Sep 2003.

[Set99] J. A. Sethian. Fast marching methods. *SIAM review*, pages 199–235, 1999.

[SFS05] Carlos E. Scheidegger, Shachar Fleishman, and Cláudio T. Silva. Triangulating point-set surfaces with bounded error. In *Proc. of the 3rd Eurographics/ACM Symposium on Geometry Processing*, pages 63–72, 2005.

[SG02] Vitaly Surazhsky and Craig Gotsman. High quality compatible triangulations. In *Proceedings of 11th International Meshing Roundtable*, pages 183–192, Sept. 2002.

[SG03] Vitaly Surazhsky and Craig Gotsman. Explicit surface remeshing. In *Proceedings of Eurographics Symposium on Geometry Processing*, pages 17–28, Aachen, Germany, June 2003.

[SL96] Marc Soucy and Denis Laurendeau. Multiresolution surface modeling based on hierarchical triangulation. *Computer Vision and Image Understanding*, 63(1):1–14, 1996.

[SNB07] Pedro V. Sander, Diego Nehab, and Joshua Barczak. Fast triangle reordering for vertex locality and reduced overdraw. *ACM Trans. Graph.*, 26(3):89, 2007.

[SPR06] Alla Sheffer, Emil Praun, and Kenneth Rose. Mesh parameterization methods and their applications. *Foundations and Trends in Computer Graphics and Vision*, 2(2):105–171, 2006.

[SS86] Youcef Saad and Martin H. Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[SS02] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1):7–42, April 2002.

[SS03] Daniel Scharstein and Richard Szeliski. High-accuracy stereo depth maps using structured light. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1:195–202, June 2003.

[SSFS06] John Schreiner, Carlos E. Scheidegger, Shachar Fleishman, and Cláudio T. Silva. Direct (re)meshing for efficient surface processing. In *Computer Graphics Forum (Proceedings of Eurographics 2006)*, volume 25/3, pages 527–536, 2006.

[SSGH01] Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe. Texture mapping progressive meshes. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 409–416, New York, NY, USA, 2001. ACM.

[SWC00] John P. Steinbrenner, Nicholas J. Wyman, and John R. Chawner. Fast surface meshing on imperfect cad models. In *in 9th International Meshing Rountable*, page 9, 2000.

[SZL92] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 65–70, New York, NY, USA, 1992. ACM.

[TC98] Marek Teichmann and Michael Capps. Surface reconstruction with anisotropic density-scaled alpha shapes. In *In Proceedings of IEEE Visualization*, pages 67–72. IEEE, 1998.

[Tur92] Greg Turk. Re-tiling polygonal surfaces. *SIGGRAPH Computer Graphics*, 26(2):55–64, 1992.

[Tut63] William Thomas Tutte. How to draw a graph. *London Mathematical Society 1963*, 13:743–768, 1963.

[Uhr07] Stefan Uhrig. *Automatische Generierung von dreidimensionalen Dreiecksnetzen aus Punktwolken-Daten*. Diploma thesis, GRIS, University of Technology, Darmstadt, 2007.

[Uml05] Georg Umlauf. Analysis and tuning of subdivision algorithms. In *SCCG '05: Proceedings of the 21st spring conference on Computer graphics*, pages 33–40, New York, NY, USA, 2005. ACM.

[VPBM01] Alex Vlachos, Jörg Peters, Chas Boyd, and Jason L. Mitchell. Curved pn triangles. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 159–166, New York, NY, USA, 2001. ACM.

[VRS03] J. Vorsatz, Ch. Rössl, and H.-P. Seidel. Dynamic remeshing and applications. In *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 167–175, New York, NY, USA, 2003. ACM.

[WB01] Kouki Watanabe and Alexander G. Belyaev. Detection of salient curvature features on polygonal surfaces. *Computer Graphics Forum*, 20:385–392, 2001.

[WK04] Jianhua Wu and Leif Kobbelt. Optimized sub-sampling of point sets for surface splatting. *Computer Graphics Forum*, 23:643–652, 2004.

[YBS05] Shin Yoshizawa, Alexander G. Belyaev, and Hans-Peter Seidel. Fast and robust detection of crest lines on meshes. In *SPM '05: Proceedings of the 2005 ACM symposium on Solid and physical modeling*, pages 227–232, New York, NY, USA, 2005. ACM.

[YL06] Sung-Eui Yoon and Peter Lindstrom. Mesh layouts for block-based caches. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1213–1220, 2006.

[ZCS02] Li Zhang, Brian Cudess, and Steven M. Seitz. Rapid shape acquisition using color structured lightand multi-pass dynamic programming. In *In The 1st IEEE International Symposium on 3D Data Processing, Visualization, and Transmission*, pages 24–36, 2002.

[ZGM09] Mao Zhihong, Cau Guo, and Zhao Mingxi. Robust detection of perceptually salient features on 3d meshes. *The Visual Computer: International Journal of Computer Graphics*, 25(3):289–295, 2009.