Dynamic Remeshing and Applications

Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

vorgelegt von
Diplom-Mathematiker Jens Vorsatz
Max-Planck-Institut für Informatik
Saarbrücken, Germany

Saarbrücken, 2006

Dekan der Naturwissenschaftlich-Technischen Fakultät I:
Prof. Dr. Thorsten Herfet

Mitglieder des Prüfungsausschusses:
Prof. Dr. Philipp Slusallek (Vorsitzender)
Prof. Dr. Hans-Peter Seidel (1. Gutachter)
Prof. Dr. Leif Kobbelt Seidel (2. Gutachter)
Dr. Alexander Belyaev (Akademischer Mitarbeiter)

Tag des Kolloquiums
12. Juni 2006

Die sichtbare Welt meinte Platon, sei nur die Projektion einer tieferen rein mathematischen Wirklichkeit: Abstrakte Linien, die sich zu Dreiecken verbinden, die zu den Seitenflächen von Körpern werden. Und da es nur fünf völlig regelmäßige platonische Körper gibt, müssen diese den Grundbausteinen des Universums entsprechen, aus denen die verwirrende Vielzahl der sichtbaren Erscheinungen konstruiert ist.

Abstract

Triangle meshes are a flexible and generally accepted boundary representation for complex geometric shapes. In addition to their geometric qualities such as for instance smoothness, feature sensitivity ,or topological simplicity, intrinsic qualities such as the shape of the triangles, their distribution on the surface and the connectivity is essential for many algorithms working on them. In this thesis we present a flexible and efficient remeshing framework that improves these "intrinsic" properties while keeping the mesh geometrically close to the original surface.

We use a particle system approach and combine it with an iterative remeshing process in order to trim the mesh towards the requirements imposed by different applications. The particle system approach distributes the vertices on the mesh with respect to a user-defined scalar-field, whereas the iterative remeshing is done by means of "Dynamic Meshes", a combination of local topological operators that lead to a good natured connectivity. A dynamic skeleton ensures that our approach is able to preserve surface features, which are particularly important for the visual quality of the mesh. None of the algorithms requires a global parameterization or patch layouting in a preprocessing step, but works with simple local parameterizations instead.

In the second part of this work we will show how to apply this remeshing framework in several applications scenarios. In particular we will elaborate on *interactive remeshing*, *dynamic, interactive multiresolution modeling*, *semi-regular remeshing* and *mesh simplification* and we will show how the users can adapt the involved algorithms in a way that the resulting mesh meets their personal requirements.

Kurzfassung

Dreiecksnetze sind eine flexible und weit verbreitete Darstellung der Aussenhülle von komplexen geometrischen Modellen und Formen. Zusätzlich zu ihren geometrischen Eigenschaften, wie z.B. Glattheit bzw. die Darstellung von besonderen geometrischen Merkmalen wie scharfen Kanten und topologischer Schlichtheit, spielen intrinsische Eigenschaften des Netzes, wie die Form der einzelnen Dreiecke, deren Verteilung auf der Oberfläche und die lokale Vernetzung, eine entscheidende Rolle, um eine Vielzahl von Algorithmen effektiv anwenden zu können. Diese Dissertation stellt eine flexible und effiziente Methode vor, die eine neue Darstellung eines Netzes erstellt, indem ein vorgegebenen Netzes neu trianguliert wird. Das neue Netz weist dabei verbesserte intrinsichen Eigenschaften auf, während es gleichzeitig eine gute Approximation an das Ursprungsnetz darstellt.

Durch die Kombination eines Partikelsystems mit einem iterativen Retriangulierungsalgorithmus erhalten wir eine flexible Methode, um die resultierenden Netze an die Anforderungen verschiedener Anwendungen anzupassen. Mit Hilfe eines Skalarfeldes verteilt dabei das Partikelsystem die Knoten des Netzes gleichmässig über die Fläche, während einfache lokale topologische Operatoren, der Kern des Retriangulierungsalgorithmus, eine gleichmässige lokalen Vernetzung erzeugen. Ein dynamisches Skelett, bestehend aus Kanten des Netzes, stellt dabei sicher, dass erhaltenswerte Details des Ursprungsnetzes auch im retriangulierten Netz vorhanden sind. Diese Detailerhaltung ist essentiell für einen guten visuellen Eindruck und ein Qualitätsmerkmal des resultierenden Netzes. Zu bemerken ist, dass keiner der in dieser Arbeit vorgestellten Algorithmen eine globale Parameterisierung des Ursprungsnetzes benötigt, sondern lediglich einfache und lokal begrenzte Paremeterisierung

verwendet werden.

Im zweiten Teil der Arbeit zeigen wir, wie unsere Retriangularungsmethode auf verschiedene Anwendungsszenarien übertragen werden kann. Hierbei gehen wir speziell auf interaktives Retriangulieren, interaktives multiskalen Modellieren, semi-reguläres Retriangulieren und Netzsimplifikation ein. Dabei zeigen wir auf, wie die im ersten Teil der Arbeit erarbeiteten Algorithmen so adaptiert werden können, daß sich die resultierenden Netze verschiedenen Anforderungen angepassen.

Acknowledgements

This thesis would have not been possible with the help and support of many people. First of all my sincere thanks to my advisor Prof. Dr. Hans-Peter Seidel, who gave me the opportunity to write this thesis in the stimulating, inspiring and inimitable environment here at the Max-Planck-Institut für Informatik in Saarbrücken. He gave me the chance to be part of an exceptional team of researchers working on the leading edge of computer graphics.

Many appreciation goes also to my co-advisor Prof. Dr. Leif Kobbelt. He lead me to the fascinating world of triangle meshes that have been part of my life since then. It has always been a great experience to work with him. His knowledge and his great breadth of understanding complex problems in no time is amazing.

I am very grateful to the people at Ag4 for providing such an enjoyable and stimulating working environment. I would like to name explicitly Mario Botsch, Stefan Brabec, Sabine Budde, Katja Daubert, Kolja Kähler, Jan Kautz, Hendrick Lensch, Conny Liegl, Annette Scheel, Ulrich Schwanecke, Hartmut Schirmacher, Marc Stamminger, Betty Stiller, Holger Theisel, Christian Theobalt, Christel Weins and Frank Zeilfelder. A very warm and special thanks to my room-neighbor Christian Rössl who was always patient and ever helpful whenever I needed his aid. Thank you all, it was a pleasure working with you.

Lastly, and most importantly, I wish to thank my parents Ute and Michael. They have always supported and encouraged me and guided me to independence, never trying to limit my aspirations. I am grateful to them and amazed at their generosity. To them I dedicate this thesis.

Contents

Abstract **v**

Kurzfassung **vii**

Acknowledgments **ix**

1 Introduction **1**

1.1 Main Contributions 5

1.2 Chapter overview 6

2 Background and Previous Work **9**

2.1 Triangle Meshes 9

2.2 Topological Operators on Meshes 11

2.3 Parameterizing Triangle Meshes 14

2.4 Remeshing Algorithms 15

2.5 Meshes with Subdivision Connectivity 17

2.6 Semi-Regular Remeshing 20

2.7 Mesh Simplification 24

3 The Remeshing Framework **27**

3.1 Relaxation 29

3.2 Dynamic Connectivity Meshes 37

3.3 Preserving Features 43
3.3.1 Alias reduction by feature snapping 45
3.3.2 The Snapping algorithm 53
3.3.3 Tagging a Skeleton 56

4 Applications **61**
4.1 Interactive Remeshing 62
4.2 Mesh Simplification 63
4.3 Semi-Regular Remeshing 65
4.4 Interactive Multiresolution Modeling with Changing Connectivity 67
4.4.1 Freeform modeling 70
4.4.2 Multiresolution modeling 73
4.4.3 Robust Multi-Band Detail Encoding 74
4.4.4 Hierarchy levels 81
4.4.5 Multiresolution modeling with changing connectivity . 84
4.4.6 Discussion 85

5 Conclusion and Future Work **89**

A A Framework to Implement Dynamic Connectivity Meshes **93**
A.1 The Callback Mechanism 95
A.2 Passing Data to `MyInfo` 98
A.3 Distributing to Multiple Clients 99
A.4 An Example Application 101
A.5 Extensions 104
A.6 Results and Conclusion 106

B Calculating Phong-Detail **109**

Chapter 1

Introduction

Information technology has found its way into almost all areas of modern life and nowadays it is even hard to imagine life without it. Information technology has changed the business of almost all enterprises and is often the source for strategic advantages. We use it to make our daily work easier and in particular in private life information technology is the key to ever increasing entertainment offerings.

The first statement is particularly true for geometry processing. Computer based geometry processing has become an appealing alternative to working with hand crafted real world geometric models in the last decades. On the one hand the underlying models share all the advantages inherent to computer models. Just to name a few, with the ever increasing speed of the internet, computer models are interchangeable over large distances which for instance enables distributed work on shared models over several continents. One can duplicate them at almost no cost, i.e., generate several instances of one and the same model or one can even put a modified version back to its original state if the newer version does not meet the expectations one was aiming at. On the other hand representing geometric shapes with computer models is particularly beneficial since one can, e.g., easily prescribe exact measures and high precisions in terms of geometric tolerances, symmetries can easily be achieved. Geometric models are scalable and one can view an object from arbitrary perspectives. Even partial views like slices or inside-

out-views of closed objects are possible. Last but not least due to excellent research in this area there exist a multitude of algorithms that work on geometric computer models that provide the users with potent means to perform challenging operations in an intuitive manner. Nowadays even unexperienced users can create complex geometric models, the skills needed for generating and modifying geometric models have shifted from mechanical skills towards profound knowledge of geometric modeling software.

Even though the designers, the artists and the engineers mainly want to focus on the actual modeling, the underlying algorithms and the mathematical object representations are still crucial for the degrees of freedom, the power and also the robustness of the available modeling operations. Over the years a variety of different object representations have established themselves, each of which inherently has its own advantages and drawbacks.

As an initial coarse classification, object representations may be described as volume-, surface- or point-based. An object has a volume-based representation if the object's interior is described by solid information; it is surface-based, if the object's surface is represented by surface primitives and it is point-based, if the object is represented by a cloud of points that give the viewer the impression of a solid object. We will elaborate on the most prominent representations of geometric shapes in Chapter 2 and will for now give a sneak preview of the representation we have turned our attention to in this thesis. It investigates different aspects of *triangle meshes*, one particular surface-based shape representation. Triangle meshes have become a generally accepted and versatile boundary representation for complex geometric shapes. To give a first impression of this representation Figure 1.1 shows two different triangle meshes approximating a technical part as it is used in mechanical engineering. The main advantage of this representation is on the one hand its simplicity of its base primitive, the triangle, and on the other hand one can approximate objects of arbitrary complexity with them. In particular the simplicity of the base primitive and due to this the availability of specialized 3D graphics libraries and highly optimized graphics hardware for their efficient display has reinforced the trend to make triangle meshes the de facto standard for display and exchange of 3D data sets.

Nowadays the literature on triangle meshes comprises a huge amount of excel-

lent work and is still growing rapidly. The virtual mesh-processing pipeline starting from acquisition down to rendering is well covered, thus triangle meshes are widely spread and used in a variety of application scenarios.

Many algorithms have been carried over from semantically rich and mathematically more sophisticated representations like spline-surfaces (cf. e.g., [64, 19]). That way one can combine the flexibility inherently provided by triangle meshes with powerful methods originally developed to represent and modify spline surfaces.

Opposed to meshes that, e.g., represent characters in the animation-/game industry, which are often hand-made and highly optimized with respect to triangle count, visual quality, and kinematics, we turn our attention to densely sampled triangle meshes often stemming from 3D scanning devices, iso surface extraction from volumes, or the sampling of parametric surfaces by CAD software. These meshes are often challenging when it comes to work with them beyond merely displaying them. For instance, models generated from CAD software often reflect a regular sampling of the underlying parameter domain, which might result in triangles with poor aspect ratios. Models generated by scanning devices suffer from holes due to limited visibility of the "camera" or due to the surface reflectance of the scanned object. As a consequence the meshes cannot be used as-is for 3D applications. An intermediate step, correcting the mesh geometry and connectivity is required. This is either done in a tedious and error prone manual optimization procedure or with semi-automated software tools. Such corrections, commonly known as *remeshing* is a fundamental part of the digital mesh processing pipeline.

In this thesis we develop a framework for dynamic remeshing, i.e., a framework that takes an arbitrary triangle mesh as input and iteratively changes the mesh connectivity and the sampling of the original mesh due to some quality criteria. In its simplest form triangle meshes with an equal vertex distribution and a regular connectivity are generated and the resolution of the mesh can be adapted to the users needs. Our approach is flexible in a way that different objective functions can be plugged in. For instance we can adapt the vertex distribution with respect to some scalar field and we can also preserve or recapture feature lines inherent to the original model. But our approach is also versatile in a way that we can use it in various application

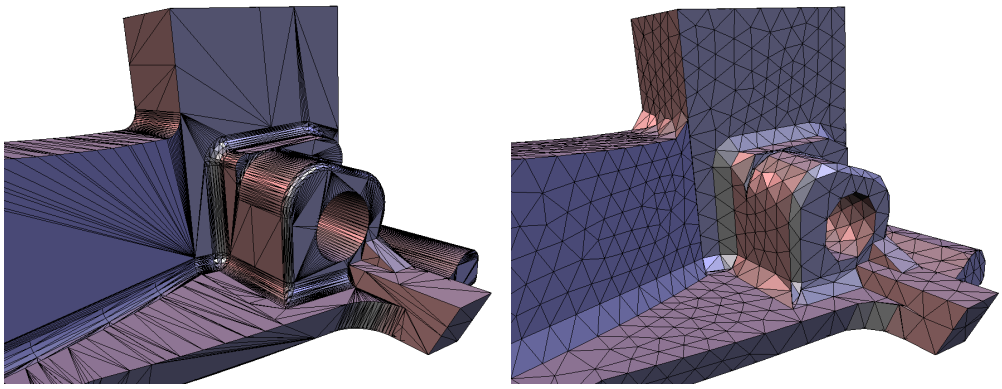

Figure 1.1: With this figure we want to give a first example for a triangle mesh. Triangle meshes are able to approximate even complex geometric models and they are particularly well supported by today's graphics hardware. At the same time this figure shows a glimpse of the algorithm we have developed during our research on triangle meshes and that we are going to describe in this thesis. The left image shows a triangle mesh of a mechanical part as it is used in mechanical engineering. This mesh is stemming from a meshing algorithm, which is state of the art in industry. A regular approximation generated with our framework is shown on the right. At a first glance the left triangulation seems to reflect the geometric properties of the object better than the right model. However, even very recent and powerful algorithms are not capable to further process this model due to the shapes of the triangles that form the mesh. Long and thin triangles lead to degenerated normals or derivatives and make numerical calculations unstable. Opposed to this, our remeshed version of the model is a "good natured" triangulation and it is well suited for further processing.

scenarios like mesh simplification, multiresolution modeling or semi-regular remeshing. We will formally introduce these techniques in Chapter 2 and describe applications of our approach that realize these techniques in Chapter 4. The focus application of our remeshing framework is multiresolution modeling and we will therefore elaborate on this and put it into a broader perspective. However, we will also describe how we use our framework in the

above mentioned areas. With our approach a user can manipulate a triangle mesh in various ways with one common framework and we hope that many more algorithms based on our work will be developed.

1.1 Main Contributions

Throughout the course of this work, parts have already been published at different conferences and journals [47, 83, 84, 46, 48, 85]. This thesis builds on these publications, presents them in a unified framework, and includes improvements and updated results.

The main contributions of this thesis can be summarized as follows.

- An novel algorithm for regularly sampling a surface with a triangle mesh. A particle system which merely utilizes simple local parameterizations evenly distributes vertices on the surface.
- A method for incrementally adapting the sampling density of a given shape. Simple local topological operators change the connectivity of the mesh in a way that approximations to the original shape can reach from very coarse to (theoretically) arbitrarily fine. Moreover a regular connectivity can be achieved.
- An algorithm for capturing and preserving sharp features, i.e., corners and edges. This includes a vertex snapping algorithm for recapturing surface features in regions with high curvature and a dynamic skeleton preserving feature lines and vertices while the underlying mesh is retriangulated.
- The application of the above methods and adaptation to different applications. In particular to interactive remeshing, interactive multiresolution geometric modeling, semi-regular remeshing and mesh simplification.
- A software design based on a callback mechanisms that enables the users to efficiently implement algorithms that can handle dynamic

meshes while maintaining encapsulation of the single reusable components.

1.2 Chapter overview

We will first introduce the notations we are going to use throughout this thesis in Sections 2.1 to 2.3 which gives us the common bases to start with. In particular we will introduce a more precise definition of triangle meshes and notions and concepts associated with them. In Sections 2.4 to 2.7 we will elaborate on previous investigations on adapting the representation of a given geometric shape to different application scenarios and will show how our work fits into this context. The technical part of this thesis starts with Chapter 3. At first we briefly discuss the overall idea of our remeshing framework and go into the details of it. In Section 3.1 we describe a particle-system approach which lets vertices of our objective mesh float on an input mesh in order to redistribute them equally. In this context we will show that different (local) parameterization methods influence the relaxation process and we will discuss how to construct parameterizations based on a minimal local domain. Section 3.2 introduces the notion of Dynamic Connectivity Meshes (DCM), a technique for integrated connectivity optimization that enables us to adapt the complexity of a triangle mesh to our needs. Here we will explain how the vertex-relaxation in the particle-system and DCM are combined. Section 3.3 is dedicated to features, i.e., prominent creases, sharp edges and corners that we strive to preserve. We will demonstrate an effective "feature snapping" technique that is able to recapture such features and we will introduce the notion of a skeleton that enables us to preserve these important surface features even if the representation of shape changes. Chapter 4 shows application scenarios for our dynamic remeshing approach. In particular we will go into interactive multiresolution modeling, semi-regular- and interactive remeshing and mesh simplification. This chapter also includes results we have achieved with the different techniques. In the Appendix we present a software design that facilitates implementing our remeshing framework and shows how one can easily incorporate it in complex applications. Moreover

we give some mathematical background on geometric calculations in order to understand and reproduce some results of this thesis more easily.

Chapter 2

Background and Previous Work

In this chapter we introduce the terms and notions and the background material that is needed to understand the new algorithms and techniques that we are going to explain in the subsequent chapters. We will first formally introduce the notion of *triangle meshes* since they are at the core of our work. We will then show how the connectivity of a triangle mesh can be changed with simple local operators. After that we explain the term *parameterization* of a mesh. In our context a parameterization is a mapping of a 2-dimensional domain to a triangle mesh living in 3-space. Among others these parameterizations differ in the nature of the domains and we use this characteristic to explain different approaches that have been published to generate them. Moreover, parameterizations are tightly coupled with the actual *remeshing* and we again use the same characteristic of different base domains to give detailed background information about the most prominent remeshing techniques known from literature throughout the remainder of this chapter .

2.1 Triangle Meshes

One can think of a simple triangle mesh as follows. The 3-dimensional boundary of an object is covered by vertices defining positions in 3-space. The mesh itself is formed by triangles that connect three vertices at a time in a way

that the surface of the object is completely covered by the triangles (cf. Figure 1.1).

In literature one finds various abstract and formal definitions for triangle meshes. For instance Lee et al. [54] adapt from an algebraic approach of [74] and formally define a triangular mesh as a pair (P, K), where P is a set of N point positions $p_i = (x_i, y_i, z_i) \in \mathbb{R}^3$ with $1 \leqslant i \leqslant N$, and K is an abstract simplicial complex which contains all the topological, i.e., adjacency information. However, since our work does not draw on specific algebraic properties of meshes, we choose to introduce triangle meshes in a less strict and more depictive manner. Throughout the definition we also build our common terminology that we will use in this thesis.

We say that a triangle mesh $\mathcal{M}$ is represented by its *vertex data* and by its *connectivity*. Vertex data can be seen as a set $\mathcal{V}$ of vertices comprising all coordinates (and optionally associated properties such as normal information, color etc.) and the connectivity which captures the incidence relation between the vertices as follows. The connectivity is defined by a set $\mathcal{H}$ of *half-edges*. A half-edge connects two vertices (its *startpoint* and its *endpoint*), three consecutive half-edges form a *triangle*. $\mathcal{H}$ and the set of all triangles $\mathcal{T}$ have an explicit mapping and depending on the context we use either the one or the other term.

In practice $\mathcal{M}$ often obeys additional constraints associated with specific topological restrictions which we are going to discuss below.

$\mathcal{M}$ is a *2-manifold mesh* if the interiors of its triangles are pairwise disjoint and if the set of all triangles $\mathcal{T}$ forms a connected 2-manifold surface with boundary. This implies that the neighborhood of each point p in $\mathcal{T}$ is homeomorphic to a disc or to a half disc. We call a half-edge, a triangle, or a vertex that contains a point that is homeomorphic to a half disc a *boundary-edge,boundary-triangle* or *boundary-vertex*. The remaining half-edges, triangles and vertices are called *interior half-edge*, *interior triangle* or *interior vertex* respectively. This also implies, that every interior half-edge has exactly one adjacent *neighbor half-edge*. If clear without ambiguity we call two adjacent half-edges simply an *edge*.

We say that a triangle mesh whose set of boundary-edges is either empty or

consists of a single loop has *no holes* and say that a mesh has *no handles*, if any closed loop of edges within $\mathcal{M}$ separates $\mathcal{T}$ into two disjoint sets. Finally we call a connected manifold mesh with no holes and no handles *simply connected*.

Within the scope of our work we restrict ourselves to manifold meshes and explicitly note, if an algorithm requires additional properties of a mesh. In practice, in particular for meshes generated by laser range scanning and merging processes, we often observe, that undesired holes or handles occur. This would clearly limit the scope of meshes our framework can process. However, recent work [11, 30, 58] effectively address this problem and thus a focus on manifold meshes with no topological noise, i.e., with no unwanted holes or handles, is not really a restriction.

2.2 Topological Operators on Meshes

Triangle meshes arise in many different contexts and it is obvious that their initial triangulation cannot always be optimal with respect to all possible application scenarios. Before we elaborate on how to re-triangulate a given geometric shape in Section 2.4 we want to show how to locally change the connectivity of a triangle mesh. We will introduce the basic operators that change the connectivity of a mesh while preserving the topology and the 2-manifoldness. Of course one can think of many more local topological operators working on triangle meshes. We limit ourselves to the ones that we are using in our framework. These operators are also the most prominent ones known from literature (cf. [49] for a detailed overview). We will introduce three edge based and two face based operators. Furthermore, we will illustrate that the probably most prominent refinement operator for triangle meshes, the 1–to–4-split can easily be derived from edge based operators [80].

But let us start with the description of the different topological operators. First we describe edge-based operators (cf. Figure 2.1) followed by triangle based operators (cf. Figure 2.2 and Figure 2.3):

edge-flip An edge "flips" and therafter connects the two vertices of the adjacent triangles, that were previously separated by this edge. This

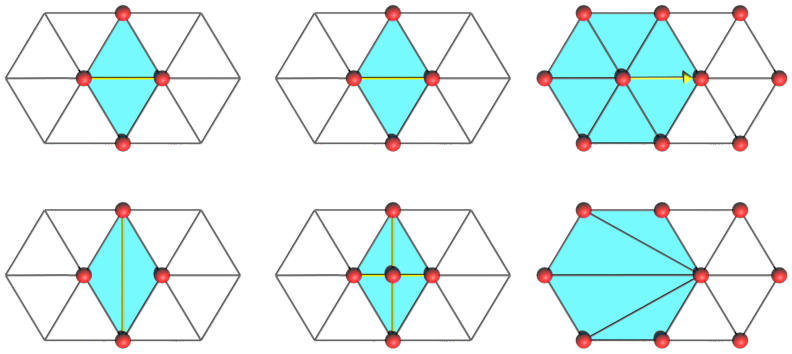

Figure 2.1: The figure shows three edge based topological operators that incrementally perform local changes of the connectivity of a triangle mesh. The top row shows the initial configuration, the lower row shows the altered mesh after having applied the operator. The area that is affected by the operator is marked in blue.
The left most column shows the *edge-flip*, where an edge "flips" and therafter connects the two vertices of the adjacent triangles.
The *edge split* shown in the middle column inserts one new vertex on an edge and connects it with the opposite node(s) of the adjacent triangle(s).
The *half-edge collapse* shown on the right lets one vertex "collapse" along an edge into the vertex on the opposite side and remove all degenerated triangles.

operator can of course only be applied to edges inside the mesh and is not allowed for boundary-edges.

edge-split It inserts one new vertex on an edge and connects it with the opposite node(s) of the adjacent triangle(s). In the general case of inner edges, two new triangles are introduced. Opposed to the edge-flip operator, the edge-split can be extended to boundary-edges in a natural way. In this case just one new triangle is introduced.

half-edge collapse The half-edge collapse shown on the right lets one vertex "collapse" along an edge into the vertex on the other side of this

edge while removing all degenerated triangles. This way one can reduce the complexity of the mesh by one vertex and two triangles.

1-to-3-split The 1-to-3-split of a triangle is a face based topological operator for triangle meshes. It inserts a new vertex inside the triangle (e.g. in its centroid) and connects all vertices of this triangle with it. This way the triangle is split in a one to three manner.

1-to-4-split This split introduces one new vertex on every edge of a triangle. In a second step the vertices are connected by edges in a way that the original face is split in a 1-to-4 manner. Figure 2.3 shows that this operator is not atomic, but can be decomposed into three edge-splits followed by an edge-flip.

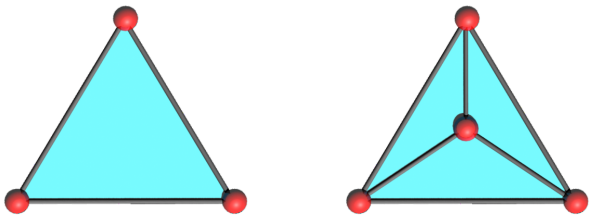

Figure 2.2: The 1-to-3-split of a triangle is a face based topological operator for triangle meshes. It inserts a new vertex inside the triangle (usually in its centroid) and connects all the vertices of this triangle with it. The triangle is split in a one by three manner.

Note that for now we have introduced the basic topological properties of the operators in order to have a foundation for the discussion in the remainder of this chapter. By simply using the three edge based operators we would be in the situation to define Dynamic Connectivity Meshes (DCM), a class of meshes that incrementally optimize their connectivity. This class of meshes was originally introduced by Kobbelt et al. [44]. DCM show their full potential only if we combine the connectivity optimization with an additional optimization of the geometry, i.e., the optimization of its vertices. For this

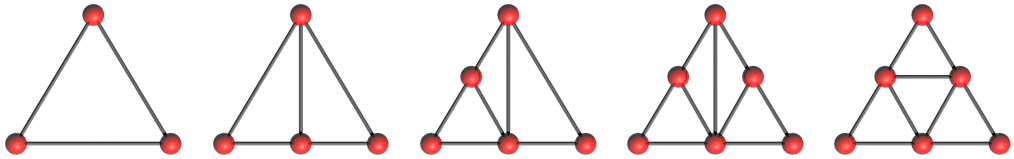

Figure 2.3: For a 1–to–4-split of a triangle, every edge is bisected by insertion of three new vertices followed by a re-triangulation. The sequence of images shows, that this subdivision can be divided into three edge-splits followed by one edge-flip [80]. Note that by using this refinement procedure we automatically handle the problem of "dangling nodes" that arise when naïvely 1–to–4-splitting just a single triangle within a mesh.

reason we dedicate Section 3.2 to this class of meshes after we have discussed the geometric optimization in Section 3.1.

2.3 Parameterizing Triangle Meshes

Before coming to the remeshing background, we would like to give a brief overview over parameterization techniques for triangle meshes. Even though at a first glance, these two topics are not directly related, they turn out to be tightly coupled.

A parameterization of a surface can be seen as a one-to-one mapping from a domain to the surface. In case the surface is approximated by a triangle mesh, the vertices are mapped and the interiors of the triangles are linearly blended. The inverse mapping from the surface to the domain is called embedding, this is the mapping that usually is constructed for a given surface.

Parameterizations have many applications from engineering to science, e.g., texture mapping [57, 69], approximation [53] and interpolation of scattered data, editing [6], etc., and we refer to [33, 20] for an excellent overview and an in depth discussion of the theory and the various parameterization methods that were proposed in the last decades. One major problem with such a global parameterization is the impact of the distortion imposed by

the specific parameterization that is used.

Basically two degrees of freedom can be exploited in order to minimize the unavoidable distortion and obtain a mapping that is close to isometric. One degree is the construction of the domain (also called the base complex), the other is the functional (cf. [14] for further details) that computes the actual mapping. A good layout of the patches of a domain can greatly decrease the distortion. In order to illustrate this, consider the two extreme cases, where the mesh is parameterized over itself. This would lead to no distortion at the cost of a large base complex. The other extreme would parameterize the entire surface over one single patch [27] at the cost of having to cut the surface along seams, thus having to deal with smooth transitions across them.

Recent algorithms try to base the parameterization on as few domain patches as possible while keeping the distortion low [14]. Since most of the recent mesh parameterization methods aim at remeshing we will discuss them in the next section. Remeshing by means of a global parameterization can be done by resampling of the parameter domain. The sample points are connected in the parameter domain. After this one evaluates the parameterization in order to map the samples back into 3D.

2.4 Remeshing Algorithms

As we have discussed in the last chapter, there is obviously a multitude of possible mesh representations for a given geometric shape. This is true even if we prescribe a certain approximation tolerance. Depending on the application for which the mesh is supposed to be used, different quality requirements have to be satisfied. Such requirements can be as diverse as, e.g., a low triangle count, a certain regularity of the connectivity (cf. next section), the triangles aspect ratios, adaptive sampling with respect to some underlying scalar field, alignment of edges with respect to an underlying tensor-field, for instance with respect to the directions of main curvature, or the normal jump between adjacent triangles. Hence, to prepare a given mesh $\mathcal{D}$ for a specific application, we need *remeshing algorithms* which take a triangle mesh and resample it in a way that the new tessellation still approximates the same ge-

ometric shape by a 2-manifold mesh $\mathcal{M}$, but additionally satisfies the quality requirements imposed by a specific application.

The first methods we are going to discuss in this section all belong to the 1-step-methods. Opposed to an incremental method, that performs the remeshing in small atomic optimization steps, these methods analyze a given input mesh, disregard the original connectivity and generate a new optimized connectivity.

Sometimes remeshing a triangle mesh of arbitrary connectivity tries to establish a special connectivity, the so called subdivision-connectivity, and we dedicate the next section to it. However, beyond this semi-regular remeshing, a number of remeshing techniques that merely re-sample and reconnect the original mesh and again generate a triangle mesh of arbitrary connectivity have been proposed.

In his pioneering work in this area, Turk [77] advocated introducing new points onto a mesh by a method of point repulsion which adds more vertices in regions of higher curvature. The old points can then be discarded, with repeated local re-triangulation (or re-tiling) as necessary, until a new mesh consisting of the new points is obtained. This approximated surface is guaranteed to be topologically equivalent to the original. This approximation technique works best for smooth surfaces with no discontinuities.

Alliez and coworkers [4] proposed an interactive isotropic remeshing scheme that uses a halftoning technique known from image analysis and sample a set of patches. They triangulate the samples and by using parameterizations that map each of the patches to parts of the original 3D surface a remeshing is obtained. In [3] they extend their method to sampling triangles instead of a regular grid. Alliez was also the first who came up with a 3D anisotropic remeshing scheme [2]. In this approach they generate a coarse base mesh by tracing lines of principal curvature along the mesh. The result is a remesh composed of quadrilaterals and triangles. Due to the global parameterization that is used, this method is limited to genus-0 patches.

An approach that avoids generating a global parameterization but operates directly on the surface is presented in [72]. In this approach geodesic curves are generated by means of geodesic distances, which are more expensive to

compute compared to our approach. They introduce a segmentation into regions in order to avoid crossing geodesics. This implies that the resulting submeshes have to be joined at the end of the algorithm.

An incremental method that is similar to the method we are proposing is presented by Surazhsky and Gotsman [75]. They perform local topological operations in order to set the target complexity of the remesh. In a second step they move vertices in a way that the areas of all triangles are equalized. In a third step they perform edge-flips in order to further regularize the connectivity. Their method leads to highly regular meshes, but it is also computationally expensive.

2.5 Meshes with Subdivision Connectivity

In the preceding Sections we have discussed several approaches that take a mesh of arbitrary connectivity as input and resample the original geometry as faithfully as possible. The output is again a triangle mesh of arbitrary connectivity that usually satisfies additional quality requirements. The schemes are primarily designed in a way that the afore mentioned goals are fulfilled. Besides having a valence close to 6 (Euler's formula), the connectivity is usually of minor importance and does not have any specific structure. But there is the important class of *subdivision-connectivity* meshes having the so called *subdivision-connectivity* that have established themselves in the literature in the past two decades and now play an important role in the world of geometry processing. Figure 2.4 shows a hierarchy of meshes that have this special connectivity. *Subdivision* is one of the most prominent sources for these meshes and has become a popular tool to generate smooth surfaces. A subdivision scheme generates a coarse–to–fine hierarchy of meshes by successively refining a coarse base mesh (i.e., by inserting new vertices and triangles). A smoothing rule places the newly inserted vertices such that the resulting meshes become (a discrete approximation to a) smooth and visually appealing limit surface. A lot of research has been devoted to analysis and design of the smoothing rules in the past and we refer to ,e.g., [18] for a detailed discussion.

After having inserted new vertices, the resulting degrees of freedom can be used in two ways. Whereas subdivision schemes position the new vertices in a way that the limit surface becomes smooth, *semi-regular remeshing methods* on the other hand take an input surface and position the vertices such that more and more geometric detail becomes visible. This is done by placing points on the original surface. The result is a mesh with subdivision-connectivity approximating the input surface. The approximation quality can be improved by further refining the mesh.

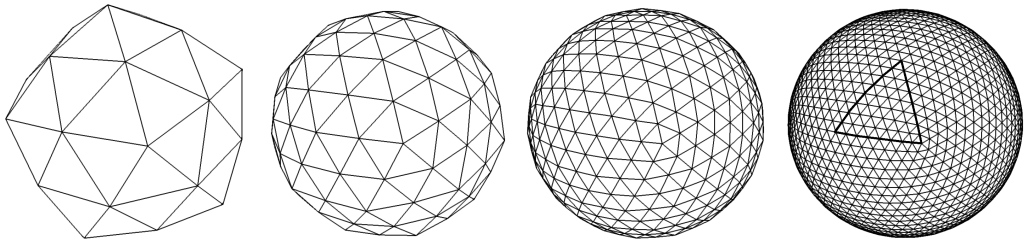

Figure 2.4: Semi-regular meshes with subdivision–connectivity are generated by uniformly subdividing a coarse base mesh (leftmost image). The figure shows a classical scheme where each triangle is split into four sub-triangles by inserting 1 new vertex in the middle of each edge. On the refined meshes we can easily identify regular submeshes which topologically correspond to a single triangle of the base mesh (right image).

Until recently, subdivision and remeshing of triangles has almost exclusively used the 1–to–4-split [16, 60] as it is shown above, that recursively splits each triangular face into 4 subtriangles (cf. also Section 2.2). With the $\sqrt{3}$-subdivision scheme Kobbelt and Labsik [43] introduced a new class of semi-regular meshes. For the uniform version, a 1-to-3-split is applied to every triangle followed by flipping all the edges of the mesh. Such a step is called $\sqrt{3}$-split since two such steps generate a regular 1–to–9 split of each face similar to the 1–to–4-split, but this time, each edge is trisected. Since for both mesh types every submesh that corresponds to one base triangle has the structure of a regular grid and the whole hierarchy is based on a not necessarily regular coarse base mesh (cf. Figure 2.4), the resulting

subdivision-connectivity meshes are also said to be *semi-regular*.

The implicitly defined connectivity established on a coarse base mesh and the direct availability of multiresolution semantics gives rise to many techniques exploiting this convenient representation as the following enumeration shows.

Compression/progressive transmission Lounsbery et al. [12] perform a multiresolution analysis, i.e., they introduce a wavelet decomposition for meshes with subdivision–connectivity. By suppressing small wavelet coefficients, a compressed approximation within a given error tolerance can be achieved. Moreover such a wavelet representation can easily be transmitted in a progressive fashion. (Send the base mesh first and refine it with successively arriving wavelet coefficients.)

Multiresolution editing For instance Zorin and co–workers [86] combine subdivision and smoothing techniques and present an interactive multiresolution mesh editing system, which is based on semi–regular meshes and enables efficient modifications of the global shape while preserving detailed features.

Parameterization Each submesh *(subdivision–patch)* can be parameterized naturally by assigning barycentric coordinates to the vertices. Combining the local parameterizations of the subdivision–patches yields a global parameterization (cf. Section 2.3). Texturing is just one application of such a parameterization.

Level–of–detail control Standard rendering libraries are able to display objects at various levels of detail, that is they display a coarse approximation, if the object is far away and switch to a finer one, if the viewer zooms in. The different subdivision levels naturally support this feature. In combination with multiresolution analysis, switching to finer resolutions can be done smoothly by fading in the wavelet coefficients.

Recent investigations show, that compact and convenient representations for multiple of the applications above can be derived when using semi-regular meshes [29, 53, 41].

2.6 Semi-Regular Remeshing

However, even if semi–regular meshes are particularly convenient for various types of applications, in practice it is rather unlikely that a given triangle mesh has this special type of connectivity (except those meshes originating from subdivision). During the last years, a couple of methods have been presented to convert a manifold triangle mesh into one having subdivision–connectivity and thus having access to the powerful methods developed for semi–regular meshes even if an input mesh of arbitrary connectivity is given.

Before we give an overview over three conversion schemes, we start by establishing the notation for subdivision-connectivity meshes and describe some quality criteria. Let an arbitrary (manifold) triangle mesh $\mathcal{D}$ be given. In this context *semi-regular remeshing* means to find a sequence of meshes $\mathcal{S}_0, \ldots, \mathcal{S}_m$ such that each $\mathcal{S}_{i+1}$ emerges from $\mathcal{S}_i$ by applying a uniform subdivision operator (either 1–to–4- or $\sqrt{3}$-split) on every triangular face of $\mathcal{S}_i$. Since the $\mathcal{S}_i$ should be differently detailed approximations to $\mathcal{D}$, the vertices $\mathbf{p} \in \mathcal{S}_i$ have to lie on the continuous geometry of $\mathcal{D}$ but they do not necessarily have to coincide with $\mathcal{D}$'s vertices. Furthermore it is allowed but not required, that the vertices of $\mathcal{S}_i$ are a subset of $\mathcal{S}_{i+1}$'s vertices.

In general it would be enough to generate the mesh $\mathcal{S}_m$ since the coarser levels of detail $\mathcal{S}_i$ can be extracted by subsampling. Nevertheless, building the whole sequence $\mathcal{S}_0, \ldots, \mathcal{S}_m$ from coarse to fine often leads to more efficient multi–level algorithms.

The quality of a subdivision–connectivity mesh is measured in different aspects. First, we want the resulting parameterization, which maps points from the base mesh $\mathcal{S}_0$ to the corresponding points on $\mathcal{S}_m$, to be close to *isometric*, i.e., the local distortion of the triangles should be small and evenly distributed over the patch. To achieve this, it is necessary to adapt the triangles in the base mesh $\mathcal{S}_0$ carefully to the shape of the corresponding surface patches in the given mesh $\mathcal{D}$. The second quality requirement rates the base mesh $\mathcal{S}_0$ itself according to the usual quality criteria for triangle meshes.

Last but not least $\mathcal{S}_0$ should contain as few triangles as possible because the algorithms working on them can directly benefit from this. This is obvious

in the context of progressive transmission and for the level-of-detail control, but also for example the "radius" of a multiresolution editing operation is limited by the coarsest mesh in the hierarchy, which implies that a coarse base domain enables edits with larger support.

As we have explained in Section 2.3, a resampling of $\mathcal{D}$ can be performed, if a global parameterization is available. In particular, if the parameterization is built upon triangular charts, a semi–regular mesh can easily be constructed. Therefore, semi-regular remeshing schemes often implicitly build a set of parameterizations based on triangular domains and evaluate them to generate semi-regular meshes. The first two schemes we are going to discuss in this Section take this venue. However, it is also possible the other way around, i.e., a global (chart based) parameterization of $\mathcal{D}$ can easily be derived if $\mathcal{S}_m$ is given. The third scheme we are going to discuss falls into this category. The discussion shows that it is difficult to distinguish between semi-regular remeshing schemes and schemes that calculate a global parameterization. However, since the classical remeshing schemes that convert an input mesh into one with subdivision–connectivity play an important role, we are discussing them separately.

In 1995 Eck et al.[17] were the first who came up with a three–step remeshing scheme. They partition a mesh $\mathcal{D}$ of arbitrary connectivity and topology into regions $\mathcal{T}_0, \ldots, \mathcal{T}_r$ using discrete *Voronoi tiles*. The algorithm ensures that the union of these tiles is dual to a triangulation which is used as the base mesh $\mathcal{S}_0$. Using harmonic maps, a close to isometric parameterization for each submesh of $\mathcal{M}$ and thus for each base triangle of S_0 is constructed in a second step. This results in a global parameterization which is C^0 continuous over the edges of the base triangles only. Eck applies recursive 1–to–4-splits to the domain triangles to generate a semi-regular mesh and evaluates the global parameterization to map the new vertices onto $\mathcal{D}$. The user has little control over the base domain. This means that the amount of base triangles is difficult to control and the base triangles are usually not aligned with surface features. Due to this the resulting remesh cannot always capture them faithfully. This is an important aspect when evaluating the quality of the remesh and we pay special attention to this in our algorithm.

An alternative approach that also constructs parameterizations on a base

domain consisting of triangles is the MAPS algorithm [54] and its improvement [40]. Opposed to Eck's scheme $\mathcal{S}_0$ is found by applying an incremental mesh decimation algorithm to the original mesh (cf. Section 2.7) This approach provides more control on the generation of the base mesh since feature lines and local curvature estimates can be taken into consideration. Again, by means of local parameterizations, that are built on top of $\mathcal{S}_0$'s triangles, the final remesh is obtained. An additional smoothing step based on a variant of Loop's subdivision scheme [60] is used to generate a certain smoothness across patch boundaries. The major shortfall of this method is the fact, that the decimation algorithm can run into local minima and thus the base mesh $\mathcal{S}_0$ can have small base triangles in certain areas.

A completely different approach to the remeshing problem for genus–zero objects is presented by Kobbelt et al. [47]. We are going to discuss this approach in more detail since it is in spirit similar to the remeshing scheme we are going to develop throughout this thesis.

The physical model behind the algorithm is the process of *shrink wrapping*, where a plastic membrane is wrapped around an object and shrunk either by heating the material or by evacuating the air from the space in-between the membrane and the object's surface. At the end of the process, the plastic skin provides an exact imprint of the given geometry. To simulate the shrink wrapping process, we approximate the plastic membrane by a semi–regular mesh $\mathcal{S}_m$. Two forces are applied to its vertices. An attracting force pulls them towards the surface. A relaxing force is applied in order to optimize the local distortion energy and to avoid folding. This ensures an even distribution of the vertices. The attracting part is realized by a projecting operator $\mathbf{P}$ that simply projects $\mathcal{S}_m$'s vertices onto $\mathcal{D}$. The relaxing is done by applying an operator $\mathbf{U}$ to all vertices of $\mathcal{S}_m$. This iteratively balances the vertex distribution. Thus, shrink–wrapping is an iterative process, where one alternates the operators $\mathbf{P}$ and $\mathbf{U}$.

Nevertheless, the proposed scheme works slightly different in order to accelerate the underlying optimization process. Instead of immediately shriveling up $\mathcal{S}_m$, the remeshing process starts with an initial convex mesh $\mathcal{S}_0$ (e.g. an icosahedron). Once the iteration converges on level $\mathcal{S}_i$, the scheme switches to the next refinement level $\mathcal{S}_{i+1}$. Hence, this multi–level approach generates

intermediate levels, which are close to the final solution, with relatively low computational costs.

Unfortunately, the algorithm described so far works for simple input meshes only. One of the problems that arise is that especially for the coarser approximations, the projection operator $\mathbf{P}$ might produce counter–intuitive results.

For this reason, the basic shrink–wrapping algorithm is extended with the aid of a parameterization F of $\mathcal{D}$ over the unit sphere. Both, $\mathcal{D}$ (using F's inverse) and $\mathcal{S}_0$ (projection) are mapped onto a sphere. Thus, $\mathbf{P}$ becomes trivial. The relaxation operator $\mathbf{U}$ is adapted to this in such a way, that it still measures the geometry on the original surface. This is done by associating triangles of $\mathcal{D}$ to corresponding surface areas of $\mathcal{S}_0$ (which is trivial, if both meshes are mapped to a sphere). This guarantees an equal distribution of $\mathcal{S}_0$'s vertices on $\mathcal{D}$ when evaluating $F(\mathcal{S}_0)$. In areas where the surface metric of $\mathcal{S}_0$ and $\mathcal{D}$ differ considerably, which would lead to severe stretching in the resulting remesh, new vertices are inserted into $\mathcal{S}_0$ by performing edge-splits. Once $\mathcal{S}_0$ is found, successive levels can be computed by either using the stabilizing parameterization over the sphere or directly, if $\mathcal{S}_i$ and $\mathcal{D}$ do not differ too much.

Similar to the shrink wrapping algorithm, our framework does not explicitly calculate a parameterization, but we can generate a semi-regular mesh in the first place and thus implicitly generate a parameterization over triangular base patches. Our framework improves the shrink wrapping process in two ways. From a technical point of view, our framework is able to process arbitrary manifold meshes with holes and handles, since we replace the parameterization over the sphere with small local parameterizations over the input mesh. Moreover we can tag and recapture features on the mesh and calculate base meshes with few triangles that are aligned with them. From an application point of view, we have extended the algorithm, that we can use it in different scenarios beyond semi-regular remeshing (cf. Chapter 4). For instance the process of generating the coarsest version of our remesh (with our without tagged feature lines) can be seen as a mesh simplification algorithm.

2.7 Mesh Simplification

Mesh simplification schemes in general also belong to the class of remeshing algorithms since they change the mesh connectivity (while reducing the overall complexity). Due to their diversified applications, these schemes belong to the well-studied areas in computer graphics and a variety of sophisticated techniques for simplifying polygonal models have been published. Even though the remeshing scheme we are going to present within the scope of this thesis can be put into action in the context of mesh simplification, the well-known techniques differ considerably from our framework and we thus want to give just a very brief overview of this field.

Mesh simplification schemes differ primarily in that they either are custom tailored for specific application scenarios or in that they trade speed and/or memory consumption for quality of the resulting mesh. Among all simplification schemes, incremental decimation algorithms that perform atomic local decimation operators (generally a single vertex gets removed) have gained remarkable popularity. A list to prioritize these removals is computed and the removals are performed with respect to this list until a target complexity of the mesh is reached. Usually vertices that do not contribute significantly to the geometrical appearance of the mesh get scheduled first, if the resulting mesh still fulfills quality criteria.

Pioneering work in this field has been done by Schroeder et al. [71]. They perform a vertex-removal followed by a re-triangulation of the resulting hole. An even simpler form of this operator are the pair-contraction schemes, that replace two vertices by one single vertex and remove all the degenerated geometry. A special case of this technique is the so called *halfedge-collapse* (cf. Fig.2.1 right) which can be considered as mere sub-sampling [45] of the original geometry. A true re-sampling is performed if new vertex positions are generated during the decimation [26]. Geometric and topological quality criteria can be used to control the algorithm, i.e., to decide in a greedy fashion which decimation step to apply next. Notable in this context is the progressive mesh representation proposed by Hoppe [32]. It uses the halfedge collapse as incremental simplification operator and stores the resulting coarse base mesh together with a sequence of vertex splits (the inverse of

the halfedge-collapse) which can be used to reconstruct the original mesh. We mention this representation since we will be using it in a different scenario and generate a hierarchy of meshes with varying geometric smoothness (cf. Chapter 4.4).

Rossignac et al. [66] take another approach and apply *vertex clustering*. A vertex clustering step can be viewed as an extension of localized vertex removal in that a group of vertices are identified for elimination and then they are replaced by a single representative vertex. The simplicity of this step permits its application to meshes with billions of triangles [59, 36] and at this time, mesh simplification of huge meshes is in the focus of current research efforts. However, the major drawback of this technique is the still rather poor quality of the output meshes (even non-manifold meshes can occur).

Chapter 3

The Remeshing Framework

Before we go into the details of our remeshing framework in the next three Sections, we want to give a high level overview, that shows the basic idea of our framework in a more prescriptive manner. Here we skip all the technicalities that we will go into later on. The goal of our work is to have an adaptable method that takes a triangle mesh $\mathcal{D}$ as input and is able to generate a preferably regular remesh $\mathcal{M}$ that is a good approximation to the input mesh. The remesh should capture details like creases or sharp edges and we also want it to gradually change, whenever the underlying input mesh changes its geometry. This property is particularly useful in the context of multiresolution modeling. For this reason we developed the incremental remeshing framework.

The idea is, that we see our remesh as an object of nodes that are connected by springs which is spanned around the input mesh. It can either be that we start with a copy of the input mesh as remesh, but our framework is also capable of taking an arbitrary remesh that is merely attached to the input mesh. If we let the nodes of the remesh go, the springs ensure, that the nodes slide over the surface of the input mesh and reposition themselves in a regular way. A key property of this repositioning is the fact, that all calculations are done locally in the vicinity of that node, i.e., no calculations of global parameterizations etc. are necessary.

A mechanism that is able to adapt the density of the spring mesh by removing

or inserting nodes and by reconnecting the springs makes sure that we can approximate the input mesh as densely as desired. The density of the remesh is not restricted by the underlying input mesh or a precalculated patch layout. Together, the springs and the density adaptation lead to a regular remesh, that approximates the input mesh.

In order to improve the quality of the remesh, we have introduced a skeleton. The skeleton makes sure, that fine detail like corners, creases or sharp edges of the input mesh is present in the remesh. The skeleton is not rigid and gives as much flexibility to the spring model as possible. In case we start off with an arbitrary remesh, we present a feature snapping algorithm that is even capable of recapturing detail of the input, which was not present in the remesh in the first place.

3.1 Relaxation

In this section we introduce the concept of *vertex relaxation*, the first building block of our remeshing framework. This section describes the geometric part of our incremental remeshing framework. For clarity reasons, we introduce the geometrical and topological optimization separately, even though they run in parallel in our implementation. Leaving our goal of remeshing a triangle mesh aside for the moment, the basic idea is quite simple and can be described as follows. We are given a set of vertices that we want to (evenly) distribute on a 3D surface. All these vertices already reside on that surface and we let them float on it with respect to a relaxing force. At the point when the relaxation procedure is in an equilibrium the vertices reach their final position. In order to achieve this goal, that initially sounds simple, we are facing two challenges. First, we have to define a proper relaxation operator that mimics the relaxation force. We want this operator to be flexible in a way that we can define different vertex distributions. Since we want to make the vertices float *on* a 3D surface the second challenge is to define this force in a way, that the movement of the vertices is restricted to the surface. There are different ways to approach this problem.

One could define a true 3D relaxation operator, that defines a 3D direction in which a vertex moves. In a second step one has to make sure, that the vertex is shifted back to the surface. This can for instance be done by a projection or a similar operator. Since the relaxation operator can be defined independently from the surface, this method can be implemented quite efficiently. However, one can easily construct examples, where the shift operator generates degeneracies. In general, such an approach is only applicable for densely sampled and smooth surfaces without sharp corners or creases.

Another way to approach the relaxation challenge would be to do all calculations *on* the surface directly and thus avoiding the shift back to the surface. Using this approach, one has to define the relaxation operator and with it all input parameters like, e.g. distances, directly on the surface. Theoretically this is an elegant approach and Sifri et.al [72] use it to calculate geodesic distances on meshes for their remeshing framework. However in our incremental setting, we found that repeatedly calculating these distances is way

to time consuming and cannot be used if interactive response times for input models of realistic sizes are required.

One way to achieve quick response times while avoiding artifacts as they occur when using a true 3D relaxation operator is to calculate local parameterizations in the vicinity of the vertex that is subject to the relaxation step. That way one could do 2D calculations in the parameterization plane and eventually map the vertex back onto the surface. As we have described in Section 2.3 there exist a multitude of parameterization schemes. For our purpose of vertex relaxation, we have to deal with the trade-off between the following two questions:

- How often do we have to recalculate the mapping?
- How faithful is the mapping?

On the one hand one can precalculate a global parameterization of that surface the vertices are floating on in advance with all the challenges that come along with it. These challenges are mainly the need to cut the surface if it is topologically complex, numerical instabilities when parameterizing large areas of a surface and last but not least the distortion that is introduced. We found that calculating a global parameterization is only suitable for rather flat (sub-)regions of a mesh. In fact we use this method in Chapter 4. Since using this method is rather straightforward once the global parameterization is known, we will not detail this approach. One could also precalculate parameterizations patch-wise and thus reducing the distortions of the individual parameterizations. This method has the drawback however, that the remeshing is sensitive to the patch structure and it is difficult to work across patch boundaries.

The other approach is to repeatedly calculate small local parameterizations in the vicinity of every vertex that is to be shifted. In the remainder of this section we will describe this approach in detail. We will show how one can compute minimal local parameterizations that minimize the distortion of the mapping while keeping the number and the effort of recalculations as small as possible.

After having informally described the basic procedure and discussed why we

are taking the approach that is based on local parameterizations, we now want to come back to our remeshing problem and describe more formally how our relaxation procedure works. We optimize the vertex distribution of $\mathcal{M}$ by relaxing its vertices while restricting their positions to the input mesh (domain) $\mathcal{D}$. This way we make sure, that our algorithm is faithful to the topology of $\mathcal{D}$. We define a particle-system that allows the vertices to float on the original surface represented by $\mathcal{D}$. A relaxation operator locally repositions a vertex with respect to its direct neighbors (1-ring). This optimization process iteratively applies the local relaxation. Again, the physical interpretation of this process is the minimization of the energy of a global system of spring-edges connecting the vertices.

The relaxation is done in a 2D parameter domain. So we need to be able to parameterize regions of $\mathcal{M}$ over the plane. For this purpose we first parameterize $\mathcal{M}$ over the domain mesh $\mathcal{D}$ by assigning to every vertex $v_i \in \mathcal{M}$ the domain triangle $\tilde{\Delta}_j \in \mathcal{D}$ that includes v_i and barycentric coordinates w.r.t. $\tilde{\Delta}_j$. We call the mapping

$$L : v_i \mapsto \big(\tilde{\Delta}_j, (\alpha_{i1}, \alpha_{i2}, \alpha_{i3})\big), \sum_{k=1}^{3} \alpha_{ik} = 1 \wedge \alpha_{ik} \geq 0$$

of all vertices $v_i \in \mathcal{M}$ the *link* between $\mathcal{M}$ and $\mathcal{D}$. So if we flatten a region of $\mathcal{D}$ to the plane, the link will provide us also with a mapping of the associated vertices and triangles of $\mathcal{M}$ to the plane.

Now we define the local *relaxation operator* $\mathcal{U}$ as the so called *weighted 2D Umbrella* operator. It shifts a vertex v_i depending on a convex combination of its direct neighbors v_{ij}. Let p_i and p_{ij} be the respective parameter values obtained from the link to $\mathcal{D}$. Then $\mathcal{U}$ is defined as

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

$$p_i \leftarrow p_i + \lambda U(p_i)\,, \quad 0 < \lambda \leq 1$$

The appropriate choice of the weights ω_{ij} will be discussed at the end of this section. The Umbrella operator $\mathcal{U}$ shifts vertices in the parameter domain,

that are lifted back to 3-space using the link. This allows the restriction to $\mathcal{D}$ without expensive and error-prone projection operators.

The actual relaxation process is similar to parameter-based fairing applied in FSR [83] where a *global* parameterization (MAPS [54]) of $\mathcal{D}$ is used. In contrast to FSR (and also, e.g. [4, 3]), we give up the global parameterization and replace it by a set of *local* parameterizations mapping regions of $\mathcal{D}$ to the plane. The benefit is that we do not have to rely on the quality of the global parameterization that may in some cases (depending on the input mesh) even be hard to construct. For a global parameterization of $\mathcal{D}$ that is defined over a coarse base domain, this prescribed domain limits the coarsest resolution of the remesh, as the size of a 1-ring in $\mathcal{M}$ is then restricted to that of a 1-ring in $\mathcal{D}$. On the other hand, for global flattening methods that induce cuts to the original surface, it is not clear how the relaxation operator should behave if a 1-ring intersects a cut. The price we have to pay with the new approach is the on-the-fly construction of the local parameterizations which cannot be done in a preprocessing step anymore. For this reason we keep the domains of the different local parameterizations as small as possible and apply caching whenever feasible.

We construct a set of local parameterizations $\{\Phi_i\}$. Each parameterization Φ_i provides a piecewise linear mapping from a set of domain triangles – the *local domain* $\mathcal{D}_L^i \subset \mathcal{D}$ – to the plane. With the *link* to the remesh $\mathcal{M}$ we can thus map associated vertices and triangles of $\mathcal{M}$. When we start remeshing with $\mathcal{M} = \mathcal{D}$ the initial link provides a 1-to-1 mapping between the corresponding vertices. Formally, for a vertex $v \in \mathcal{M}$ corresponding to $\tilde{v} \in \mathcal{D}$ we can choose a single triangle $\tilde{\Delta} \in \mathcal{D}$ that contains $\tilde{v}$ as its local domain.

As we start the optimization process, $\mathcal{U}$ will compute a shift vector for v from a convex combination of the direct neighbors v_j in the 1-ring. Hence shifting v requires a larger domain, since a parameterization of the 1-ring is needed. We try to keep this local domain small as well as the updates on it. So we restrict the shift to parameter points located in the intersection of the 1-ring $\Delta^1(v) \subset \mathcal{M}$ and the subset $\mathcal{D}_L^v \subset \mathcal{D}_L$ of triangles that this vertex v is assigned to. This way we ensure that the new position is well defined, i.e. has a mapping in the parameterization, and we avoid a more expensive point-in-triangle test.

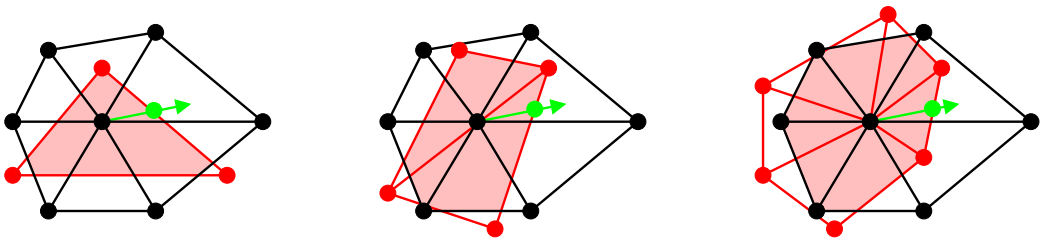

Figure 3.1: Example for calculating the shift vector for the vertex inside triangle (left), on edge (middle), and on vertex (right) case. The remesh $\mathcal{M}$ is drawn in black, the domain mesh $\mathcal{D}$ is drawn in red. The green arrow shows the vector calculated by the relaxation operator $\mathcal{U}$. The shift is restricted to the red area, the green dot marks the final position of the center vertex.

This determines the *minimal domain* that has to be covered by the local parameterization. It consists of the union of

- triangles of $\mathcal{D}_L$ covered by the triangles of v's 1-ring, i.e., $\{\tilde{\Delta} \in \mathcal{D}_L | \Phi(\tilde{\Delta}) \cap \Phi(\Delta^1(v)) \neq \emptyset\}$,

and $\mathcal{D}_L^v$ which is defined as

- a single triangle $\tilde{\Delta} \in \mathcal{D}_L$, if the vertex is inside this triangle, i.e., $\Phi(v) \in \Phi(\tilde{\Delta})$, or
- two neighboring triangles $\tilde{\Delta}_1, \tilde{\Delta}_2 \in \mathcal{D}_L$, if v is located on their common edge, i.e., $\Phi(v) \in \Phi(\tilde{\Delta}_1 \cap \tilde{\Delta}_2)$, or
- a 1-ring in $\mathcal{D}_L$, if v corresponds to a vertex $\tilde{v}$ in $\mathcal{D}_L$, i.e., $\Phi(v) = \Phi(\tilde{v})$.

Fig. 3.1 shows the three different cases.

The reasons for keeping $\mathcal{D}_L$ small are the lower costs for the construction of a local parameterization as well as the lower distortion that is induced. In fact, we might not even be able to construct a reasonable parameterization in situations where the area is topologically or geometrically complex. Thus, $\mathcal{U}$ cannot be computed for the corresponding vertex and we keep its position. This rarely happens, and as the neighborhood of the vertex is relaxed we are likely to find a parameterization in the next iteration.

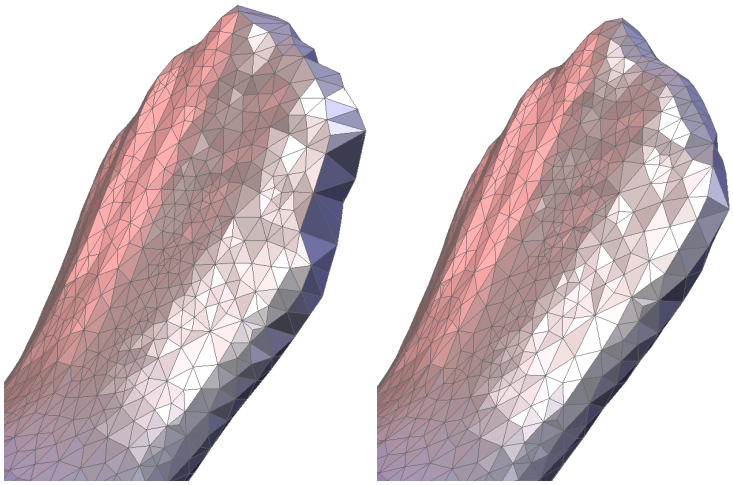

Figure 3.2: A closeup view of Tweety's tail. A local parameterization based on a projection into a fitting plane leads to degeneracies in areas with high curvature (left), whereas a local mapping with Floater's shape preserving parameterizations performs well (right).

In order to significantly speed up the algorithm, we use a much simpler method to generate the local parameterization. If the cone of normals of the triangles in $\mathcal{D}_L$ has only a small opening angle we can even use a simple projection to a fitting plane for constructing the parameterization. This is especially useful in the case when $\mathcal{D}$ is relatively coarse compared to $\mathcal{M}$. (Just consider the trivial case when a 1-ring of $\mathcal{M}$ is completely contained in a single domain triangle of $\mathcal{D}$.) In general we have to apply a more sophisticated parameterization scheme. Figure 3.2 shows the degeneracies that occur, when exclusively using the projection operator.

There are a number of methods for flattening disk-like 2-manifold meshes to the plane [21, 14, 34, 73]. We use Floater's [21] shape preserving parameterization after projecting the boundary of $\mathcal{D}_L$ to a plane. Due to the definition of $\mathcal{U}$ the boundary is usually "fairly convex" and we almost always get a valid, bijective parameterization without foldovers of triangles. In case of an invalid parameterization we either could map the boundary to a circle and

apply Floater's method again, thus ensuring a valid mapping, or give up for this iteration as mentioned above. Our experiments show that the less expensive second alternative works well. We use the first alternative only after constructing a local parameterization for this vertex has failed several times.

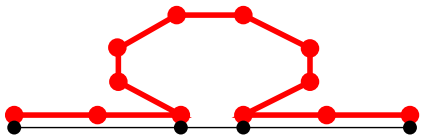

Figure 3.3: The remesh $\mathcal{M}$ (black) might miss some feature part of the domain mesh $\mathcal{D}$ (red) if we set $\omega_{ij} = 1$. We have to incorporate the area of the triangles of $\mathcal{D}$ that are covered by a 1-ring of a vertex in $\mathcal{M}$ into the relaxation operator to solve this problem and get a uniform sampling of $\mathcal{D}$.

Consider a configuration as illustrated in Fig. 3.3. Despite the fact that a projection parameterization is in general not a good choice for highly curved surface regions (cf. Fig. 3.2), there is the problem that the remesh $\mathcal{M}$ misses some feature part of the domain $\mathcal{D}$. This situation is likely to happen if $\mathcal{M}$ is coarse (cf. the connectivity optimization described in the next section) compared to the surface feature on $\mathcal{D}$. In order to avoid this situation the relaxation operator $\mathcal{U}$ will take into account not only a 1-ring of $\mathcal{M}$ but also the covered domain triangles, i.e., the convex weights ω_{ij} used for relaxation depend on the area of domain triangles. This explains why the local domain $\mathcal{D}_L$ must cover a 1-ring in $\mathcal{M}$.

Formally, we can assign a local domain to every 1-ring. Practically, we assign to every mesh triangle $\Delta \in \mathcal{M}$ the set

$$C(\Delta) := \{\tilde{\Delta} \in \mathcal{D} | \Phi(\Delta) \cap \Phi(\tilde{\Delta}) \neq \emptyset\}$$

of domain triangles that have a non-empty intersection with Δ in the parameter domain. The application of $\mathcal{U}$ requires the union of all $C(\Delta_j)$ with triangles Δ_j from the respective 1-ring.

In every single relaxation step only a single vertex v is repositioned. So the corresponding $C(\Delta_j)$ have to be updated. We intersect each parameter

triangle $\Phi(\Delta'_j)$ of the new 1-ring with all triangles of $\mathcal{D}_L^v$ and reassign them to the new triangles Δ'_j. This update step can be implemented efficiently since the intersection tests are done for all triangles of the 1-ring which are sharing common edges. Thus, intermediate intersection results can be reused. Moreover, the outer hull of the 1-ring remains unchanged, hence there is no need to test against it. Also, we only need to recompute a parameterization if domain triangles have been added to the set of triangles assigned to a 1-ring. This caching of local parameterizations considerably speeds up our algorithm.

The choice of the weights ω_{ij} determines the energy to be minimized in the optimization process. Uniform weights will provide a uniform distribution of triangles in the local parameter domains. As the local domains are kept small, the resulting parameterizations are near isometric resulting in a uniform point distribution in 3-space. However, Fig. 3.3 illustrates, that surface features of $\mathcal{D}$ might be missed in the remesh. Hence we choose the weights ω_{ij} for the weighted umbrella proportional to the area (in 3-space) of all those domain-triangles that are covered by the 1-ring of v_{ij}.

Before we show results of this relaxation procedure, in addition to the geometrical optimization we first have to introduce the topological optimization that adapts and optimizes the connectivity of the remesh. This optimization is subject to the next section.

3.2 Dynamic Connectivity Meshes

Whereas the last section dealt with the *geometric* properties, i.e., the repositioning of $\mathcal{M}$'s vertices, this section presents a technique that is capable of adapting the resolution of a mesh to a target complexity by changing its *connectivity*. Since changing the connectivity of $\mathcal{M}$ is one of the main building blocks of our remeshing framework, we will explain in detail how to achieve our goal of adapting $\mathcal{M}$'s resolution. Here, we will focus on the *topological* operation and its operators, which locally change $\mathcal{M}$'s connectivity, as we have introduced them in Chapter 2. (Figure 3.4 again shows the operators we are going to use in our framework.)

We will also show how to combine adapting $\mathcal{M}'s$ connectivity with the relaxation procedure from Section 3.1 and explain how to efficiently put the techniques into practice.

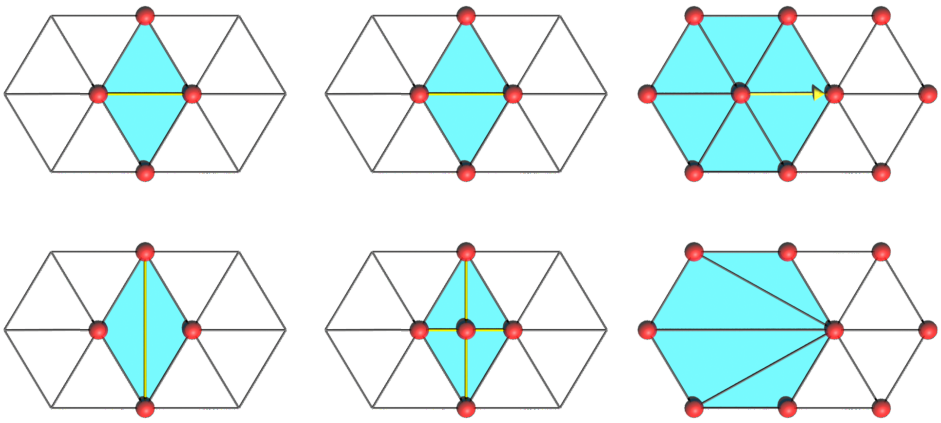

Figure 3.4: The three basic topological operations involved in DCM. The top row shows the initial configuration, the lower row shows the altered mesh applying an operator. From left to right: edge-flip, edge-split and halfedge-collapse. The affected area is marked in blue.

By means of the particle system, up to now, our remeshing scheme is able to reposition the vertices of $\mathcal{M}$ on $\mathcal{D}$. As already mentioned the other im-

portant component is to adapt the resolution of $\mathcal{M}$ while optimizing the connectivity. Starting from the initial connectivity of $\mathcal{M}_0$, we apply simple topological operators that insert or remove vertices and regularize the connectivity. This way we can incrementally adjust the complexity of $\mathcal{M}$ to a desired target resolution. In order to achieve this, we apply an algorithm similar to *DCM* [44], since we want to obtain a good mesh quality according to the following criteria:

- No edge should be shorter than $\varepsilon_{\min}$.
- No edge should be longer than $\varepsilon_{\max}$.
- According to Euler's formula, a vertex' valence should be 6 (or 4 at boundaries of a surface)

In order to achieve our first goal, we apply the half-edge collapse (cf. Figure 3.4), which reduces the triangle count of $\mathcal{M}$ by removing two triangles (or one triangle for boundary edges) by collapsing one edge into a single vertex. This is done to all edges that fall below a length of $\varepsilon_{\min}$. Note again, that a half-edge collapse is allowed only if the resulting mesh remains two manifold (cf. [82] for a detailed explanation).

In order to increase the triangle count of $\mathcal{M}$, we perform an edge-split and insert a vertex on every edge that is longer than $\varepsilon_{\max}$ and connect it with the opposite vertex in the adjacent triangle(s). The new vertex will be positioned on one of the endpoints of that edge e that was split. We call this vertex the parent-vertex of the newly inserted vertex. In fact, we are creating two geometrically degenerated triangles in the first place. But inserting the new vertex on the midpoint of e would require an extra point-in-triangle-search (cf. [9]) in the parameter plane, since we have to establish the link to $\mathcal{D}$ for the new vertex as we have described it in the previous section. Instead we let the relaxing procedure do the repositioning.

To regularize the connectivity, we perform edge-flipping. This becomes necessary since both edge-collapse and edge-split affect the vertex balance, and according to Euler's formula we want most inner vertices to have valence six. Consequently, for every two neighboring triangles $\triangle(A, B, C)$ and

$\triangle(A, B, D)$ we flip the common edge between A and B, if that reduces the total valence excess:

$$\sum_{p \in \{A,B,C,D\}} (valence(p) - 6)^2$$

Note that all three topological operations are inexpensive and local since they do not need any global optimization and since they only affect a small region on $\mathcal{M}$. Moreover, $\mathcal{M}$ always remains a valid 2-manifold (opposed to the basic 1–to–4 triangle-split which is used in many subdivision schemes).

Now the desired resolution of $\mathcal{M}$ can be set by adapting $\varepsilon_{\min}$ and $\varepsilon_{\max}$. Notice that $\varepsilon_{\max} > 2\varepsilon_{\min}$ has to be satisfied in order to avoid generating two (with respect to $\varepsilon_{\min}$) invalid edges during the split operation.

In theory, both algorithms, i.e., relaxation and DCM work independently of each other. The relaxation operator $\mathcal{U}$ is iteratively applied to all the vertices. Here, the algorithm simply steps through the list of vertices in a linear order. As soon as an edge exceeds $\varepsilon_{\max}$ or falls below $\varepsilon_{\min}$, the corresponding topological operation is performed. In practice however, we additionally schedule those vertices for immediate relaxation that were affected by the topological operation (the remaining vertex during the half-edge collapse and the newly inserted vertex introduced by the edge–split) thus ensuring a faster convergence of the particle system. Moreover, we particularly test the affected regions for possible edge-flips, since both half–edge–collapse and edge–split change the local valence-excess.

Obviously changes in the connectivity enforce an update of the local domains that are used in the relaxation step. This is trivial for the edge-split, since we introduce degenerated triangles which by definition cannot have an intersection with domain triangles. The recalculation of the local parameterization is done in the next relaxation step for the new vertex. During the half-edge collapse, we assign those domain triangles that were associated to the removed triangles to their respective neighbors. Again, since the recalculation is local it does not require much effort. Last but not least, the reassignment in case of an edge-flip is straightforward.

In addition to this basic adaptation of the connectivity which takes merely the edge length and the valence into account, we also experimented with

a variety of geometric constraints that could prevent certain geometrically degenerated situations. For instance one could prevent edges from flipping or from collapsing, whenever the normals of the affected triangles would flip over, i.e., one normal would change by more than 90 degrees. We found, that these restraints (that in addition would prevent the remesh striving for a regular connectivity) can indeed temporarily increase the mesh quality, but the basic algorithm is capable of fixing local degeneracies and thus we abandoned additional geometric restraints completely.

An approach that leads to an even more regular structure than the meshes we are generating is proposed by Surazhsky et al. [75]. They color the mesh according to valences of vertices and let edges drift over the whole mesh in order to find edges that can be flipped and thus decrease the total valence excess. Since this searching procedure is not local and the valence excess is decreased only punctually by one, we do not include this approach in our framework and stick to the simple and basic optimization method.

Now we can show first results that we have achieved with the framework we have built so far. A video, showing our dynamic remeshing framework in action is available on our web-site (www.mpi-sb.mpg.de/~vorsatz). This video shows, that the relaxation procedure combined with DCM runs at interactive speed for meshes with about 100k$\triangle$.

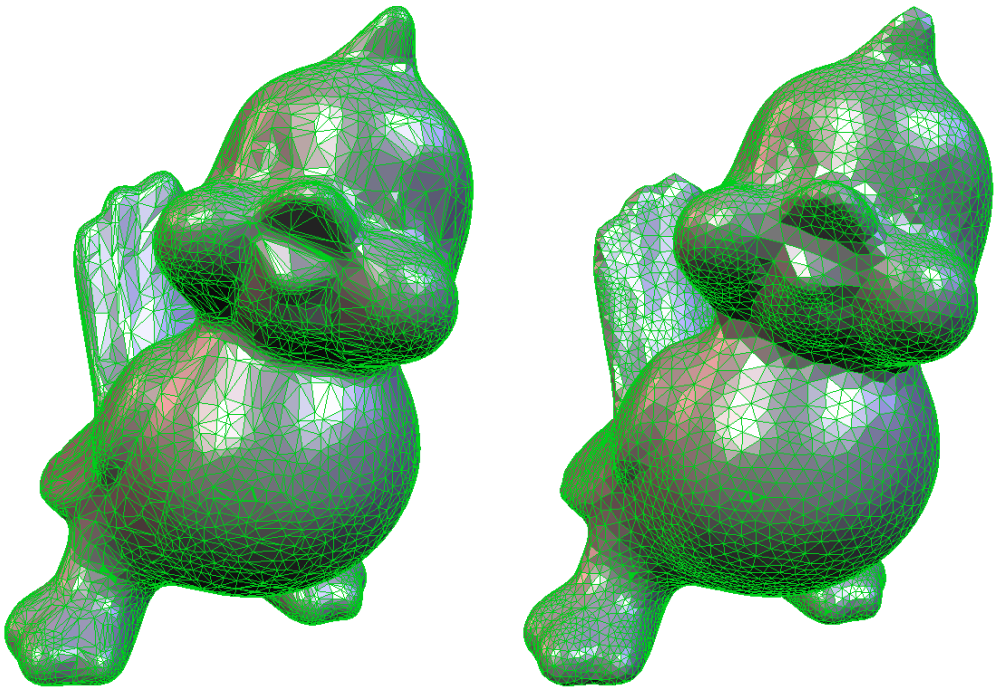

Figure 3.5: Remeshing of Tweety, a geometrically more complex model (original data-sets on the left, remeshed version on the right). The result uses the plain vanilla remeshing approach, except for setting the target resolution, no interaction of the user is necessary. The remeshing procedure for this 20k$\triangle$ model runs in less than 5 seconds. Thin parts of the model are faithfully recaptured (closeup view of Tweety's tail can be found in Fig. 3.2).

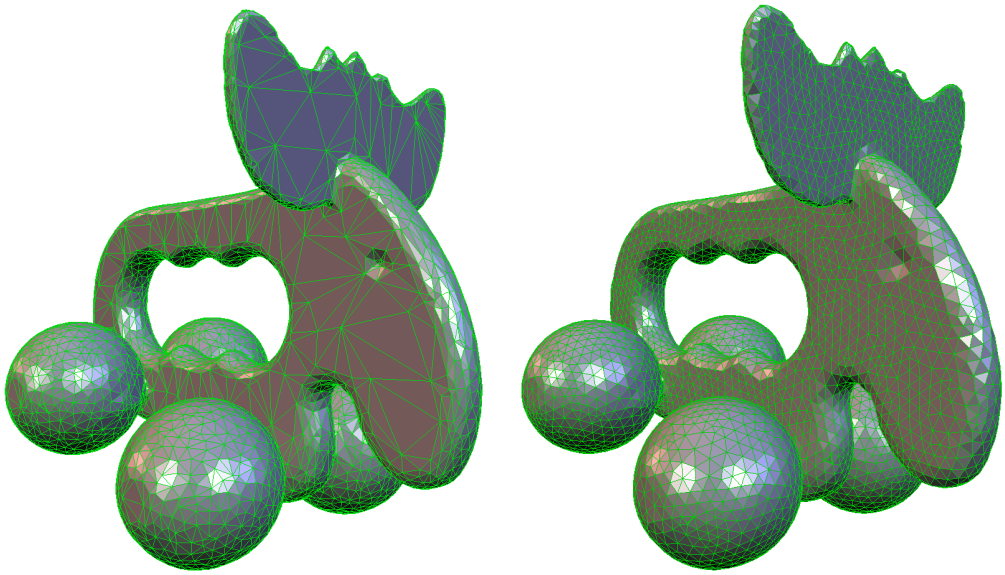

Figure 3.6: Remeshing of the Elch model, a geometrically and topologically complex model Opposed to the original model (left) , no degenerated triangles are present in the remesh (right) which facilitates further processing steps. Even though the geometry is topologically challenging for a remeshing algorithm, our algorithm can handle such type of models, since it works without an explicit patch-layout or a global parameterization but uses small local parameterizations instead.

3.3 Preserving Features

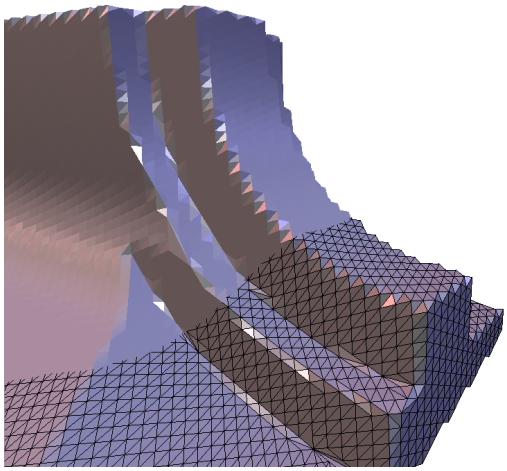

Figure 3.7: Example for aliasing artifacts. Without taking care of feature lines, aliasing artifacts cannot be avoided even though a regular sample grid is used. Edge-flipping and aligning the sample grid with the features slightly improves the situation but still, sharp edges are lost.

A general challenge of most geometry resampling techniques is the *alias problem*. If the geometry has salient features, such as creases where the original geometry does not have a continuous tangent plane, or ridge lines, we can often observe surface reconstruction artifacts (cf. Figure 3.7). Feature detection is a well studied problem in a variety of areas in particular for images and volumetric data defined on regular grids. For triangle meshes however, the literature on feature detection is by far not as mature.

Although, e.g., some mesh decimation techniques are quite reliable in preserving feature edges, we are generally facing the problem of finding the exact location of the features when placing the surface samples on the mesh. But the underlying fundamental problem is in fact much deeper. Features are not solely defined by, e.g., a single mathematical property such as curvature, rather than they are a subjective and application dependent property of the mesh and it is difficult to specify them in the first place. However, if we do

consider regions of high (or even infinite) curvature to be good indicators for feature edges there are also problems of technical nature to identify surface features. For instance it turns out to be rather difficult to estimate curvatures in a reliable and robust fashion on piecewise linear surfaces. Although many powerful curvature discretizations are known [63, 76, 13, 5], these operators still suffer from random noise affecting the vertex positions. Hence simple thresholding techniques to detect sharp features almost never work fully automatically.

Very recently *geometric snakes*[56] a generalization of Kass et al's image based *snakes* [38] to interactively reconstruct features has been proposed. The user prescribes a polygonal curve as a rough estimate of the feature s/he wants to capture. This curve is deformed by an energy functional. The result is a piecewise linear curve that traces the exact surface feature. Hubeli et al. [35] presented a semi-automatic framework to extract a collection of piecewise linear curves describing the salient features of a surface. Their two step multiresolution approach first classifies patches of high curvature and extracts feature edges by a skeletonizing algorithm.

The method we are going to present in this section not only computes piecewise linear curves representing salient features of a mesh, but starts out with an input mesh $\mathcal{M}_0$ and remeshes it in a way that it reconstructs features of a domain mesh $\mathcal{D}$. In order to do the remeshing, we merely require M_0 to be *linked* (cf. Section 3.1) to $\mathcal{D}$. One source for problems to recapture sharp features is also the fact that placing some samples on the feature does not guarantee a correct reconstruction. To achieve this, we additionally have to align mesh edges to the feature. This however implies that for correct feature reconstruction we have to change the mesh connectivity in general. Hence the task of resampling a given mesh requires geometric *and* topological optimization. In the last sections we have already built a powerful framework to address both optimization problems.

In the next section we extend the relaxation procedure (cf. section 3.1) by an effective mechanism to attract vertices of $\mathcal{M}$ to feature edges of $\mathcal{D}$ to exactly find the surface features [83]. The attracting force is imposed by means of a hierarchical curvature field. Again, DCM provides the means to align edges of $\mathcal{M}$ to feature edges. Our algorithm does not require any thresholding

parameters to classify the features but works fully automatically.

Subsequent to this Section 3.3.3 describes a method to preserve already tagged features edges and feature vertices when they are subject to the relaxation procedure or changing connectivity as introduced in the last sections. The method in its basic form is restricted in that it requires the initial remesh $\mathcal{M}_0$ to be identical to the domain mesh $\mathcal{D}$ though. In many situations, e.g., in the application scenario we have described in the preceding chapter, however we are facing exactly this situation and do not have to deal with the more complicated case where $\mathcal{M}_0 \neq \mathcal{D}$. Nevertheless we will also describe a way how to extend the basic method to the more complicated case. As a prerequisite for this extension, we use the feature snapping procedure as it is described in the next section.

3.3.1 Alias reduction by feature snapping

In Section 3.1 we saw how to equally distribute the vertices of the remesh $\mathcal{M}$ on $\mathcal{D}$, such that $\mathcal{M}$ consists of nicely shaped triangles. Moreover, the DCM approach presented in Section 3.2 makes sure that we get a regularized connectivity. But without explicitly taking care of it, the relaxed vertices and the connecting edges not necessarily sample feature regions accurately. The reconstructed mesh still suffers from aliasing (cf. Figure 3.7). In this section we propose a technique that reduces aliasing by snapping vertices to features of the original surface. In Section 3.3.2 these techniques will be integrated into an explicit snapping algorithm.

The underlying idea is to start a relaxation step with different forces that make the vertices move towards feature regions of $\mathcal{D}$. We do not explicitly classify these regions by thresholding curvature or more sophisticated techniques as used, e.g., in the context of surface segmentation in reverse engineering [79], texturing [57], or parameterizations [68]. This way we avoid specifying any application specific parameters. As a consequence we prefer the metaphor of vertices that move into the direction of higher curvature to that of a force that is induced by the feature and pulls vertices towards it.

Note that even though the method we are going to present in this chapter

is in spirit similar to the general remeshing framework we have described in the previous sections, we do not mix these two methods, but mainly use the feature detection as a preprocessing step to recapture and tag the features only. One advantage of this two step approach is the fact that, we can process almost arbitrary input meshes as long as we can establish a link to a domain mesh. We get a reasonable initial remesh with reconstructed features in the first step and fine tune the remeshing with our more general framework including the skeleton approach from Section 3.3.3. We have done this separation mainly for two reasons. First, and more importantly, since the particle system is a versatile approach that can be trimmed towards many different objective functions, we do not want to have the process of feature recapturing interfere with it. This combination would imply that two possibly contradicting goals would have to be incorporated into the relaxation operator of the particle system. The second reason is due to the fact that separating the two objectives is not really a restriction, since we have developed a method to preserve tagged features during the remeshing process. We are going to present this method in the next section.

The short answer to the question what makes vertices move in the direction of the features is that we use a hierarchical scalar curvature field defined on the domain surface $\mathcal{D}$ whose gradient provides the (candidate) directions towards which the vertices can move.

The *link* establishes a one-to-one correspondence between the two meshes, and points on $\mathcal{M}$ can be mapped to points on $\mathcal{D}$ (and vice versa). Hence, having this correspondence, directions defined on $\mathcal{M}$ can be transferred to directions or motion of vertices on $\mathcal{D}$. At first, we will focus on the definition of the curvature field on $\mathcal{D}$ and treat the details of vertex shifting after that.

Mathematically, curvature is defined for smooth, differentiable surfaces only. Regarding a piecewise linear triangle mesh as an approximation to a continuous surface leads to curvature discretizations as proposed ,e.g., in [63, 76, 13]. For our purpose a rather simple curvature measure is sufficient, as we do not need convergence to a smooth surface in the limit case. We require that flat regions have smaller curvature than feature edges that in turn have lower curvature than “corners”. This setting allows vertices from flat regions to snap onto edges first, and vertices residing on feature edges to snap to corners.

We calculate a discrete analogon to mean curvature at a vertex $\mathbf{v}$ as follows: regard all edges emanating from $\mathbf{v}$. For every edge we calculate the angle between the normals of the two attached triangles. The sum of all these angles is the curvature value of this vertex. It is straightforward that this measure is reasonable according to our requirements. This can be illustrated for a cube: the eight corner vertices have curvature $\frac{3}{2}\pi$, on edges the curvature is π, and all other vertices in the flat regions have zero curvature.

Up to now we defined curvature for $\mathcal{D}'s$ vertices only, but we can easily assign curvature values to all points of the piecewise linear manifold $\mathcal{D}$ by simple linear interpolation of the curvature values that are defined at the vertices.

Using this simple measure there is one challenge we have to pay attention to. Suppose two vertices with high curvature are connected by an edge. Although this edge may be in a flat region, i.e., the normal deviation of its attached triangles is zero, it is assigned a high curvature due to linear interpolation between the two high curvature vertices. This situation can be avoided by preprocessing the original mesh $\mathcal{M}_0$: bisecting every edge (cf. Section 3.2) is a straightforward solution to this problem. In practice we restrict the splitting to regions with noticeable curvature. Note that this is done for reasons of efficiency only. The induced curvature threshold does not affect the rest of the algorithm, in particular it is not used for classification of features and their detection, but just truncates regions of very low curvature. This is no problem in principle, but just an artifact caused by linear interpolation of this simple curvature measure.

From the scalar curvature field we can construct a gradient field that indicates the direction into which a vertex should move in order to snap to some feature. However, the naïve approach of calculating the curvature gradient for every triangle greatly suffers from (high frequency) noise effecting in more or less random vertex shifting. Imagine, e.g., a planar region: even minimal geometric disturbance will produce a meaningless direction field. Our aim is to move vertices of $\mathcal{M}$ towards prominent sharp feature edges of $\mathcal{D}$ while minimizing disturbance by noise.

In order to fit our needs we make use of a noise reduction scheme that smoothes the curvature and hence the gradient field. The initial scalar val-

ues are low pass filtered by weighted averaging and again truncated. This is iterated several times resulting in a hierarchy of curvature fields on different frequency bands. So the gradient of the lowest frequency field points reliably towards sharp feature edges over a wide region, but smaller features may be missed. In contrast, the higher frequency fields faithfully respect smaller features but also geometric noise and are thus less reliable. Figure 3.9 illustrates curvature fields of the fan-disk model at three different levels of the hierarchy.

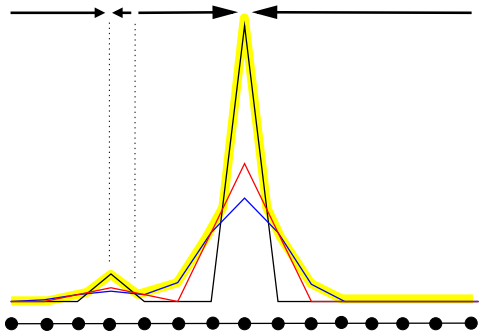

Figure 3.8: Curvature filtering. This example shows a 1D setup of vertices with their curvature values at levels 1 (input values, light grey line), 2 (red line) and 3 (blue line) of the smoothing hierarchy (y-axis). We use the curvature field that is defined by the maximum curvatures over all levels (thick yellow line). The arrows show the direction of the resulting gradient field in the different intervals. Notice how the smoothing has enlarged these intervals allowing more distant vertices to snap onto the sharp features.

The idea is to first move vertices along the direction of the lowest frequency gradient. Higher frequency bands of the scalar curvature field are subsequently faded in by taking the maximum of the values of different levels in the hierarchy. This defines new gradients on successive levels respecting smaller features while the "support" of a feature depends on its significance. The filtered curvature field provides sharpness in the vicinity of features and smoothness elsewhere as illustrated by Figure 3.9 and 3.8. We found that in practice it is sufficient to use the gradients of *this* scalar field for moving

vertices rather than to successively shift vertices according to gradients on different filter levels.

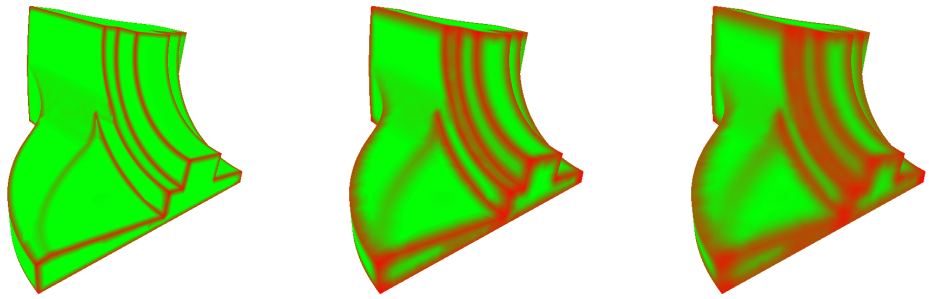

Figure 3.9: Three different levels of the hierarchical curvature on the fan-disk model. The red colors indicate regions of high curvature, the green regions represent areas of lower curvature. The leftmost image shows the unfiltered curvature field where due to the definition of the curvature measure the influence of the dominant features is limited to triangles that are adjacent to them. As the curvature gets filtered (2 times in the middle, 5 times in the right image), sharp features get blurred, but cover larger areas of the mesh.

Certainly, the number of filter levels could be used to suppresses small features and hence to serve as feature selection. This is not the intention of the algorithm: the main purpose of filtering is to provide a meaningful gradient field over a considerably large support with enough candidate vertices to be attracted by the features.

Ideally, a vertex is moved along the filtered direction field. If a local curvature maximum is detected on its route, the vertex snaps to this position. Such a maximum occurs on edges/vertices only due to its construction. In order to avoid a constant flow in the direction of the maximum curvature, the vertex will remain at its original position if there is no local maximum. This way vertices are blocked by adjacent vertices that are already closer to the feature and as curvature maxima can appear only on the edges of $\mathcal{D}$ it is sufficient to test for maxima on (intersections with) these edges only.

Using this snapping technique we can reconstruct features of the original surface by plain geometric optimization. Of course, the quality of the result-

ing remesh is unacceptable without introducing additional constraints that prevent degeneration such as orientation flipping of triangles. In our method this is achieved by restricting a vertex' movement to the "kernel" of its 1-ring. The kernel is the convex subregion of the 1-ring that is bounded by the straight lines defined by outer edges. Figure 3.10 shows an example.

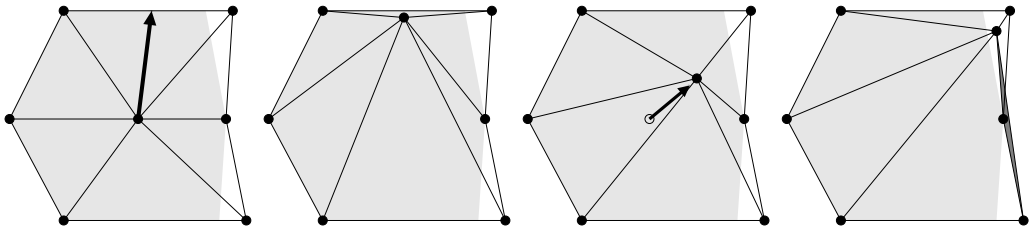

Figure 3.10: Starting from the current position (left), the vertex is shifted with respect to the gradient field indicated by the black arrow. Moving in the direction of the gradient can cause caps in the vicinity of features (center left). This can be avoided by shifting along the edge that encloses the smallest angle with the gradient (center right). A potential short edge gets removed during the restructuring process (cf. Section 3.2). Note that the shift has to be restricted to the kernel of the 1-ring (grey area) to avoid flipping of triangles (right).

When shifting the vertex inside its kernel along the gradient, caps may arise (Figure 3.10, center left). These are triangles with almost zero area and one inner angle close to π. We must avoid such configurations as neither a snapped vertex will move any further nor the topological optimization (cf. Section 3.2) can remove all of them. For this reason we discretize directions: curvature maxima are searched on the edge emerging from a vertex $\mathbf{v}_i$ of the current remesh $\mathcal{M}$ that encloses the smallest angle with the gradient at $\mathbf{v}_i$ (Figure 3.10, center right). Of course, only edges with an enclosing angle less then $\frac{\pi}{2}$ are allowed as we want to shift vertices in the direction of the higher curvature. Since the mesh is consistent we can always find at least one edge with this property. Restricting the direction this way ensures that vertices are still moved with respect to the gradient, merely the convergence to features is slowed down.

Hence, the direction along that a vertex $\mathbf{v}_i \in \mathcal{M}$ is attracted towards the

feature always points to some neighboring vertex $\mathbf{v}_j$. We now evaluate curvature on $\mathcal{D}$ at the intersections of the edge between $(\mathbf{v}_i, \mathbf{v}_j)$ with all edges of $\mathcal{D}$. If a local curvature maximum is detected the vertex is moved to this position, else it remains untouched, i.e., it stays in its current position. Again, in order to avoid constant flow, vertex movement is restricted: $\mathbf{v}_i$ may not move further than half of the length of the edge $(\mathbf{v}_i, \mathbf{v}_j)$. The reason for this damping is that the neighbor $\mathbf{v}_j$ may reach the same position easier and we do not want to be dependent on the order we feed the vertices to the snapping procedure. In the extreme case $\mathbf{v}_j$ lies on the feature already and collapsing of two vertices is prevented that way.

Vertices that do not snap to a feature are subject to a relaxation step (cf. Section 3.1) instead of moving along the gradient. This ensures that vertices remain equally distributed. At a first glance the need to relax vertices that were not allowed to move is not obvious, but the rationale behind it is rather simple. As outlined before, vertices are allowed to move within the kernel of its one-ring only. Since the relaxation procedure is built in a way that it produces equilateral triangles wherever possible, we are facing convex one-rings for almost all vertices which in turn implies that the kernel is equal to the one-ring and that vertices are allowed to move freely within this region. This way we are able to reduce situations with "locked" vertices and thus speed up the algorithm significantly.

Vertices that have been shifted onto an edge of $\mathcal{D}$ due to a curvature maximum are restricted to move only on edges in $\mathcal{D}$ in the following stages of the algorithm. This allows us to use the same snapping technique in a one dimensional subspace of the parameter domain in order to resample corners of the domain surface $\mathcal{D}$. Again, a vertex $\mathbf{v}_i \in \mathcal{M}$ tries to move towards high curvature, but this time along edges of $\mathcal{D}$. Its movement is still restricted to the kernel of the one-ring. If a local maximum is found at a vertex $\mathbf{v} \in \mathcal{D}$ a corner is assumed, and $\mathbf{v}_i$ is shifted to that position. $\mathbf{v}_i$ remains untouched either in case of there is no maximum or if there is another vertex $\mathbf{v}_j \in \mathcal{M}$ already sampling $\mathbf{v}$. All vertices that have not snapped to corners are finally relaxed on their feature edges in $\mathcal{D}$.

After feature snapping to edges (in 2D) and to corners (in 1D) the vertices of the remesh $\mathcal{M}$ are equally distributed, and they sample feature edges and

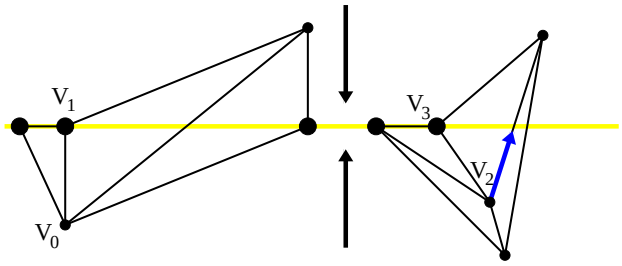

Figure 3.11: Even after the relaxation procedure some special cases may occur. The gradient field (black arrows) shifts vertices towards a feature (horizontal line). Vertex $\mathbf{v}_0$ is locked by an adjacent vertex $\mathbf{v}_1$ that is already residing on the edge (left). Flipping the edge that crosses the feature line aligns $\mathcal{M}$ with $\mathcal{M}_0$. On the right, vertex $\mathbf{v}_2$ moves onto the horizontal line, generating a degenerated triangle. Relaxing $\mathbf{v}_3$ fixes this problem.

corners of the original surface. Now we make sure that edges connecting vertices in $\mathcal{M}$ are aligned to sharp features of $\mathcal{D}$. Due to the previously described discretization of directions for vertex movement, a vertex might be locked which leads to an edge crossing a feature of $\mathcal{D}$. Figure 3.11 (left) illustrates the situation. These cases rarely happen due to the intermediate relaxation and can be solved by flipping the affected edge. Since the procedure can be applied iteratively, the algorithm works in the same fashion for several edges attached to one vertex.

In a final step we take care of triangles with poor aspect ratios. Such a configuration can occur, if the vertex is shifted towards the border of the kernel in a non-convex one-ring. Figure 3.11 (right) shows such a configuration. In situations where we detect these triangles, we apply the relaxation operator to the vertex with the largest enclosing angle. In case this vertex happens to reside on a feature, we let it drift away from it, since the edge that is opposite to the vertex is a good approximation to the feature edge already.

3.3.2 The Snapping algorithm

In the previous sections we discussed the components of our feature framework and elaborated on the different aspects. Now we are able to outline the overall snapping algorithm that recaptures features and is applied in a preprocessing step prior to the actual remeshing process.

We assign every vertex $\mathbf{v}_i \in \mathcal{M}$ a state that will be evaluated and modified by the algorithm. A vertex can either have status *free*, *snapped–to–edge* or *snapped–to–corner*. Initially, all vertices are *free*.

In step one an initial remesh can optimally be built using DCM as described in Section 3.2. The resulting mesh has the desired complexity in the number of vertices and is optimal with respect to the connectivity criteria. This way we can adjust how densely the features of $\mathcal{D}$ get sampled. This step also sets all vertex states to *free*.

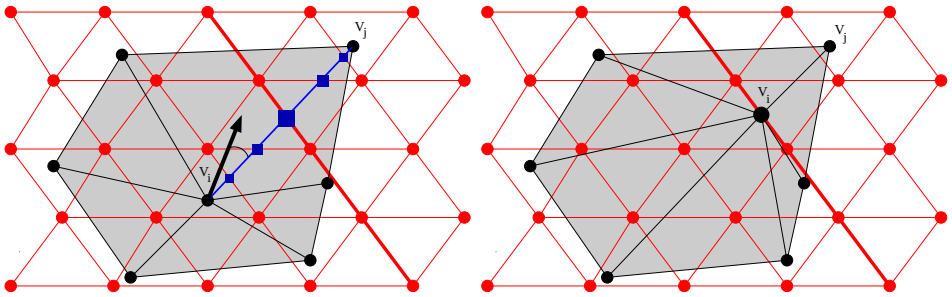

Figure 3.12: Illustration of our feature snapping algorithm for a single vertex $\mathbf{v}_i$. Left: The regular, light mesh denotes the original mesh $\mathcal{M}_0$ with a feature edge (thick). The dark mesh shows a 1-ring of $\mathcal{M}$ with the sampled curvature gradient (arrow). All vertices are in *free* state. The search direction is determined by the edge enclosing the smallest angle with the gradient. Curvature values c_k are sampled at the intersections with $\mathcal{M}_0$, displayed by the boxes. The size of the boxes reflects the value of c_k. Right: The center vertex snaps to the local maximum of the c_k and changes its state to *snapped–to–edge*.

The optimization only effects the connectivity of the triangle mesh but not its geometry. In the next step we start the geometric optimization (cf. Sec-

tion 3.1): the positions of all *free* vertices are shifted using the simple area based relaxation operator. This results in a mesh with a uniform vertex distribution.

Still, the surface is an unacceptable reconstruction of the original geometry due to aliasing in feature regions. Naïve local refinement cannot solve this problem. In order to ensure that vertices of $\mathcal{M}$ lie on the features of $\mathcal{D}$ we now apply the snapping procedure.

The third step snaps vertices of $\mathcal{M}$ to feature edges of $\mathcal{D}$ (cf. Section 3.3.1). The filtered curvature field on $\mathcal{D}$ provides a gradient direction that guides vertices towards features. Note that no feature regions or edges of $\mathcal{D}$ are classified, so no application specific parameters are needed. As a vertex $\mathbf{v}_i \in \mathcal{M}$ may only move towards a feature edge along an edge $(\mathbf{v}_i, \mathbf{v}_j)$, only intersections of $\mathcal{D}$ with this edge are new candidate positions for $\mathbf{v}_i$. Curvature is evaluated for these potential target positions only and used as the snapping criterion to find the final position.

If a local curvature maximum is encountered between $\mathbf{v}_i$ and $\mathbf{v}_j$, a feature edge is assumed. As a consequence $\mathbf{v}_i$ is snapped to the intersection point of this maximum, and its state changes to *snapped–to–edge*. Figure 3.12 illustrates this situation. If there is no such maximum, $\mathbf{v}_i$ stays at its position thus avoiding constant flow and degenerated edges. Also, $\mathbf{v}_i$ is not allowed to snap onto a *snapped–to–edge* vertex to avoid edge collapses.

Vertices that remain in state *free* may now be scheduled for another relaxation step. Steps two and three are iterated until no more *snapped–to–edge* vertices are generated. This ensures that there are enough candidates for snapping. Now vertices of the resulting remesh $\mathcal{M}$ sample edges of the original surface, but still corners are subject to aliasing.

It turns out that the corner problem can be solved in just the same way as we have addressed the edge problem. We have to snap vertices to corners of the original surface, e.g., to vertices where feature edges meet. In contrast to step three the search space is restricted from the entire 2D parameter domain to the 1D feature edges. We have already detected these edges, as they connect *snapped–to–edge* vertices.

So step four can be formulated as follows: For every vertex $\mathbf{v}_i \in \mathcal{M}$ with

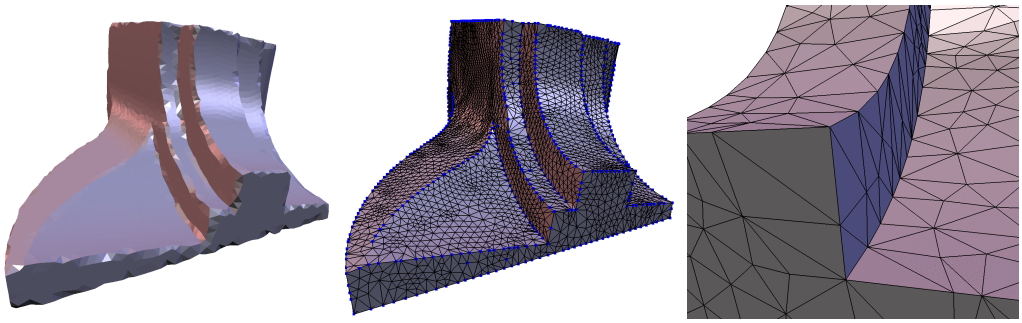

Figure 3.13: The particle system it its simple form is not sensitive to features which results in aliasing effects (left). After the snapping algorithm, all the features are reconstructed (middle), snapped vertices are colored in blue. A closeup view of the same model at a coarser resolution is shown on the right.

state *snapped–to–edge* the curvature values at the two endpoints of the corresponding edge in $\mathcal{D}$ are used to select one of the two possible search directions. If the vertex happens to lie on a vertex in $\mathcal{D}$ and there are more than two candidate directions, the vertex remains untouched as it is assumed that a corner is found already.

Again, we are looking for a local maximum between $\mathbf{v}_i$ and its neighbor $\mathbf{v}_j$ with the same state. If a curvature maximum is detected, $\mathbf{v}_i$ snaps to the associated vertex in $\mathcal{D}$ and its state changes to *snapped–to–corner*. If the maximum happens to be in $\mathbf{v}_j$, the vertex $\mathbf{v}_i$ is left unchanged. The vertices remaining in state *snapped–to–edge* are scheduled for relaxation on their feature edge. Corner snapping and 1D relaxation are iterated until there are no more vertices that snap to corners. Now $\mathcal{M}$ reconstructs edges and corners of $\mathcal{D}$ in so far as vertices are placed on feature edges respectively corners.

The last step in the algorithm fixes edges that are still not aligned and cross a feature. This is achieved by edge flipping. This is in contrast to the requirements of balanced number of adjacent vertices, but has to be tolerated for the sake of an improved approximation to $\mathcal{D}$. In this step we also release vertices that form degenerated triangles as illustrated in Figure 3.12 and reset it from the snapped–to–edge status into the free status.

3.3.3 Tagging a Skeleton

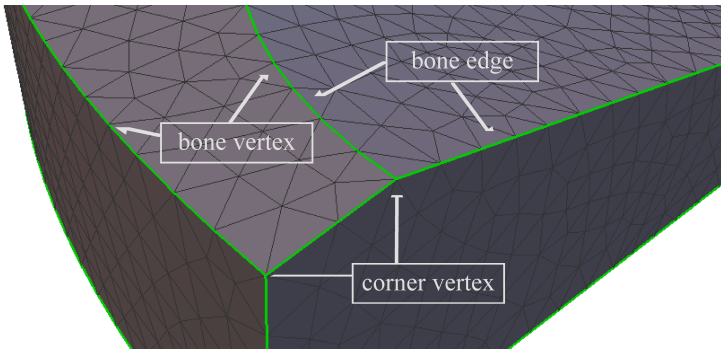

Figure 3.14: The skeleton on the fan-disk model was extracted by thresholding of dihedral angles between adjacent triangles. The figure shows the three different primitives the skeleton consists of before the remeshing starts. The corner vertices remain fixed, bone-vertices are allowed to freely float on the bone-edges.

For now let us consider the case where $\mathcal{M}_0 = \mathcal{D}$. In this simpler case similarly to [37, 3], we make use of skeletons that are attached to both meshes. In principle, we define a skeleton to be a set of edges of $\mathcal{D}$ that can be either selected by an automatic [67, 57] or semi-automatic [56, 35] algorithm or even by an interactive selection done by the user. Additionally the user can add vertices to the skeleton that s/he wants to be preserved during the remeshing.

In order to preserve this basic set of vertices and edges, we define the skeleton of $\mathcal{M}$ to consists of the following three primitives. In doing so, we exploit the fact that the two meshes are identical. See Figure 3.14 for an example.

bone-edges are edges that were selected by the user/feature detection algorithm.

bone-vertices are vertices that have exactly 2 adjacent bone-edges.

corner-vertices are vertices that have $\neq 2$ adjacent bone-edges or vertices that are explicitly selected by the user.

The skeleton which is attached to $\mathcal{D}$ remains fixed during the whole remeshing process while its counterpart on $\mathcal{M}$ is modified. In order to ensure that the skeleton on $\mathcal{M}$ preserves the structure of $\mathcal{D}$'s skeleton, the key idea is to restrict the relaxation operator $\mathcal{U}$ for the bone-vertices of $\mathcal{M}$ to the bone-edges of $\mathcal{D}$. At the same time we ensure that the three topological operations, namely edge-flip, edge-split and half-edge collapse used in the re-triangulation process do not destroy this structure.

Hence, we apply the following restrictions to the relaxation operator $\mathcal{U}$ and to the topological operations on $\mathcal{M}$.

- *Corner-vertices* remain fixed and never get touched by any geometrical nor topological operation. We legitimate this approach with the argument that vertices that have exactly one adjacent bone–edge are likely to be an endpoint of a ridge line. If the vertex has more than two bone–edges adjacent to it, it is probably a complex node of the skeletal structure of a feature and should be preserved.

- *Bone-vertices* are moving on *bone-edges* of $\mathcal{D}$ exclusively – $\mathcal{U}(p)$ is simply projected back to that *bone-edge*, that has the smallest enclosing angle with $\mathcal{U}(p)$. This strategy is effective for vertices that reside in the middle of a ridge line and ensures that bone-vertices are able to float freely within this ridge. If the movement of a bone-vertex is prohibited since it represents, e.g., some important surface feature, the user simply tags it as corner-vertex.

- A half-edge-collapse of a *bone-edges* are allowed only if both endpoints and the connecting edge belong to the skeleton. This also implies, that no bone–vertex is allowed to drop out of the skeleton by a half-edge collapse. However, vertices that do not belong to the skeleton are allowed to collapse into the skeleton, and become part of it in that case (cf. Figure 3.14).

- *Bone-edges* never get flipped since they represent an important surface feature that has to be preserved.

- If a *bone-edge* e gets split, the new vertex is a *bone-vertex* as well. If the parent-vertex (see Section 3.2) happens to be a *corner-vertex*, it is allowed to move in the direction of that *bone-edge* of $\mathcal{D}$ that has the smallest enclosing angle with e and where the other endpoint of e is reachable (cf. Figure 3.15).

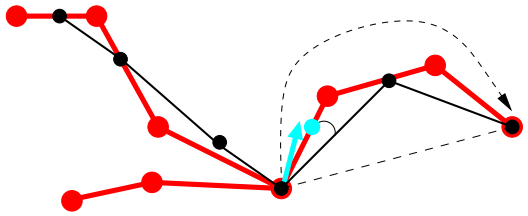

Figure 3.15: After a split of a bone-edge of $\mathcal{M}$ the newly inserted bone-vertex (green) gets attached to a bone-edge of $\mathcal{D}$ (red). If the new vertex has a corner-vertex as its parent (central vertex), we attach it to that bone-edge that has the smallest enclosing angle with the bone-edge that was split. Additionally we require, that the opposite vertex of $\mathcal{M}$ can be reached via $\mathcal{D}$'s skeleton (dotted arc). After that, the new vertex is allowed to move on bone-edges of $\mathcal{D}$ exclusively.

With these restrictions we are able to guarantee that the topology of the skeleton does not change, while the vertices and edges faithfully sample the initial skeleton. Even though we impose the above restrictions to the particle system and the DCM as we have introduced them in the first place, we found that preserving the skeleton does not severely affect the quality of the resulting remesh. Still, a line of bone-edges separates the "freely floating" areas that are adjacent to it and particles are not allowed to cross this barrier. Future work in this area might therefore include a less rigid skeleton metaphor where particles are allowed to leave the skeleton while at the same time an adjacent vertex snaps to the ridge line and thus preserves a good feature-sampling.

In the last part of this section we want to describe a way to extend our skeleton approach to the case where $\mathcal{M}_0$ differs from $\mathcal{D}$. The only aspect we

have to take care of is the creation of the skeleton that is attached to $\mathcal{M}_0$. As soon as this skeleton is defined, we are in the same situation as described above and can use the restricted topological and geometrical operators. As input we are assuming that we are given a set of polygonal lines that consists of edges of $\mathcal{D}$. These lines form $\mathcal{D}$'s skeleton, i.e., all the edges are tagged as bone-edges. We also assume that we are given a set of vertices V_s from $\mathcal{M}_0$ (the potential bone- respectively corner-vertices) and a *link* between $\mathcal{D}$ and M_0. We can, e.g., get this set of vertices as output of the feature snapping procedure from the last section. The remaining subtask is to classify the vertices of V_s to be either corner- or bone-vertices and to identify the bone-edges of $\mathcal{M}_0$'s skeleton.

Initially we tag all vertices of V_s to be corner vertices. By means of the link we immediately know if a vertex from V_s is residing on a bone-edge of $\mathcal{D}$. These vertices are candidates to become bone-vertices. In order to eventually determine $\mathcal{M}_0$'s bone-edges and thus completing M_0's skeleton, we apply the following strategy. For every $\mathbf{v} \in V_s$ we regard all emanating edges and check if the vertex $\mathbf{v}_j \in V_s$ on the opposite side is *reachable*. If this is the case, the corresponding edge becomes part of M_0's skeleton. $\mathbf{v}_j$ is said to be reachable, if there exists a connected set of bone-edges of $\mathcal{D}$'s skeleton that connects $\mathbf{v}$ and $\mathbf{v}_j$ and there is no other vertex of M_0 on this path. Now we can proceed by tagging bone- and corner-vertices according to their number of emanating bone-edges just as previously described. Figure 3.16 shows a result we have generated with the skeleton approach we have escribed throughout this section. It shows a remeshing of the fandisk data-set where a skeleton is preserved at different levels of resolution.

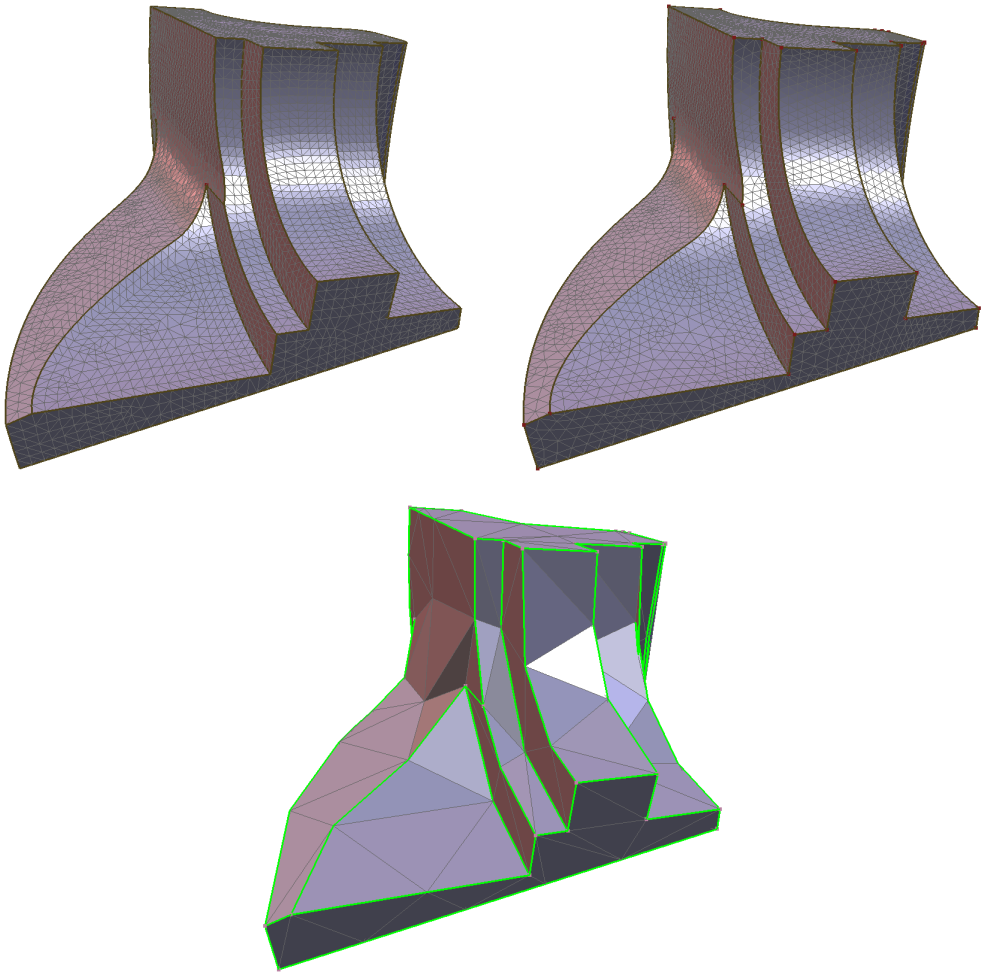

Figure 3.16: The original fandisk data-set with its skeleton and corner-vertices (top left) gets remeshed (top right). Due to the restrictions imposed on the relaxation and topological operators, the skeleton is preserved even though we generated a really coarse approximation to the original mesh (bottom).

Chapter 4

Applications

After having the detailed technical description of the general framework in place we can now put it into practice. With our framework a user can manipulate a triangle mesh in various ways with one common approach and we hope that many more algorithms based on our work will be developed. In this chapter we will describe how to adapt the general technique to different application scenarios. We will explain when to plug in different relaxation functions or parameterizations in order to have a hands on description for an application of our work. At first we will describe a method for interactive remeshing, a straightforward application of the framework. We explicitly mention it since it is a valuable tool when a user wants to locally improve a given mesh and wants to have the improvement seamlessly integrated into the input mesh. We will briefly describe mesh simplification that generates a coarser version of a base mesh. Based on this simplified version we will describe several ways to generate semi–regular remeshes. The focus application of our remeshing framework is multiresolution modeling and we will therefore elaborate on this and put it into a broader perspective in the last section of this chapter.

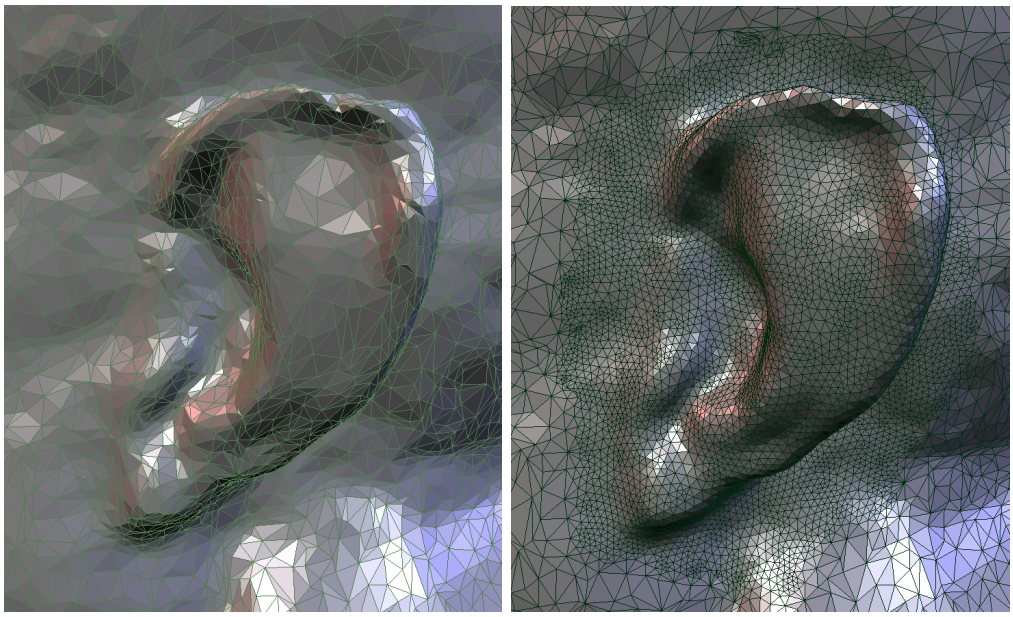

Figure 4.1: An interactive remeshing session operating on the ear of the Max-Planck model. The original triangulation on the left gets refined. Note that the partially remeshed area automatically connects to the fixed vertices on the boundary and still forms a valid 2-manifold mesh.

4.1 Interactive Remeshing

While the remeshing per se is fully automatic once the user has specified input parameters like, e.g., the target resolution and/or a feature skeleton, we can also use it to provide a framework for interactive mesh optimization. Therefore, the optimization process is visualized by updating the displayed remesh after each iteration. The user can change input parameters during the optimization process, and gets immediate visual feedback as the mesh converges to the desired remesh.

Even more important in this context is the fact that one can specify only certain regions on the mesh that are subject to the remeshing process while the rest remains fixed. This can be regions on the original surface that include only few sample points for example. The optimization process runs just as

before but schedules only vertices and edges in the specified regions for optimization. Again, this automatically ensures that the remeshed regions stay connected to the fixed part of the mesh as its resolution and its connectivity is optimized, and no additional zippering or stitching [78] is necessary.

In practice the user-defined regions are often small enough to allow the parameterization of a whole region over the plane. As no local parameterizations have to be updated, this will significantly speedup the algorithm. Plugging in different parameterizations can be quite efficient, but it also demonstrates how flexible the general framework is.

Of course, the user can always modify the feature skeleton during the remeshing process. Fig. 4.1 shows an example of an interactive remeshing session. Here we locally increased the vertex density in order to have additional degrees of freedom for a subsequent step in the virtual geometric modeling pipeline.

4.2 Mesh Simplification

Due to the ever increasing size of geometric models and the limited amount of hardware resources, mesh simplification has been one of the most active research areas in geometric geometry processing for the last decade. We have already given a brief overview over the relevant research directions in this field in Section 2.7. Incremental mesh simplification steered by a priority queue and by means of the edge-collapse as atomic decimation operator is unquestionable the most popular method to reduce the size of a geometric model. Our remeshing framework is in some way similar to this approach in that it utilizes the same topological operator but in contrast to these methods we steer the decimation process by the particle system while prescribing certain edge-lengths. However, by simply increasing the minimal and maximal edge length we can easily adopt our scheme for mesh simplification and can thus generate “fairly” coarse approximations to the original input mesh. Our approach has the theoretical advantage, that it is less likely to run into local minima during the decimation. In practice however, we observe an advantage for fairly coarse and irregularly sampled input meshes only. By

using the skeleton approach we are in addition able to preserve features of the input mesh which drastically improves the quality of the decimated mesh. Theoretically, for genus zero objects, we can perform a simplification down to a tetrahedron as long as we find a valid parameterization for every 1–ring in $\mathcal{M}$. In practice, for objects with features, we are restricted by the rules imposed by the skeleton (see Section 3.3.3).

We do not claim, that our method outperforms any of the sophisticated, application specific simplification schemes known from literature, but nonetheless we see several aspects where using our scheme in the context of mesh simplification is useful.

- First, and most importantly, the meshes that come out of our remeshing framework are particularly well suited as base-meshes in the context of semi-regular remeshing and as meshes that are subject to discrete fairing. We will detail this in the subsequent sections.

- A link between the original mesh and the remesh is always available, thus we can easily exchange properties between corresponding regions of both meshes.

- As we have seen in Section 4.1 on interactive remeshing, we can apply the remeshing to small parts of the complete mesh while ensuring an intact triangulation. Even for very coarse triangulations the particle system equally distributes vertices on the original geometry. By attaching skeletons to the original mesh and its remesh the user can thereby exert influence on the overall result.

- Our remeshes interpolate the original geometry at its vertices and we do not just approximate it.

- Last but not least, we have developed an application that implements all the algorithms that were described so far in one single framework. Performing mesh simplification with this application comes at no extra costs, since it is inherently part of our general framework already.

4.3 Semi-Regular Remeshing

Arbitrary (irregular) meshes are the most general and flexible contiguous boundary representation and meanwhile we have become familiar with them throughout this thesis. But there is also the important class of *semi–regular* – or *subdivision–connectivity* – meshes often stemming from subdivision-algorithms [18] that offer many advantages over the irregular setting. On the one hand this is due their regular structure and on the other hand one can exploit their mathematical proximity to polynomial surfaces. Many algorithms, in particular in the context of rendering, filtering, texturing, and compression, can benefit from this special structure. In the past, a number of algorithms have been proposed to convert an arbitrary input mesh into one having subdivision–connectivity. Since we have already introduced the notion of semi–regular meshes in Chapter 2, please refer to page 20 for a more detailed description of this special and important class of meshes.

In this section we want to show that our remeshing framework can be directly applied to perform such a conversion. Since we have described all the required algorithms already, the conversion is rather straightforward. We merely have to deal with generating the subdivision–connectivity. Once this special connectivity is established, the geometric part of the remeshing procedure, can be adopted "as is" from Section 3.1. This eventually guarantees the desired vertex density on the final remesh. In other words the particle system guarantees fine grained control over the vertex distribution by adapting the relaxation operator in the desired way. By means of the skeleton (Section 3.3) we can additionally set edge and vertex constraints in order to preserve and eventually reconstruct feature lines of the domain mesh.

The conversion is done in two phases. Similarly to the *MAPS*-algorithm [54] or its successor [40], we first generate a coarse version of the input mesh $\mathcal{M} = \mathcal{D}$ just as we have described it in the last section on mesh simplification. Again, in terms of coarseness of the base-mesh we are solely restricted by the constraints imposed by the skeleton (if at all present) and by the availability of parameterizations for every one-ring of $\mathcal{M}$. Thus we can build the hierarchy of meshes upon a very coarse base domain with a small number of triangles. We have given the rationale why this is desirable in Section 2.6.

The particle system ensures, that the areas of the triangles of this coarse approximation (all all finer approximations in the course of the remeshing) to $\mathcal{D}$ are uniform.

Once we have generated this coarse base-domain, we can start with the refinement phase in a second step. This is done similarly to the above remeshing algorithm in that we apply topological changes combined with permanent relaxation. But instead of using the *DCM*-approach, we go from coarse to fine by applying the topological operations as they are given by the underlying subdivision scheme.

We can plug in either the classical dyadic refinement (repeated 1–to–4–split operation). This does not imply dealing with non-manifold "intermediate" meshes containing T-junctions. As mentioned in the Chapter 2, we split the operator into edge-flips and vertex-splits. In doing so we are in the DCM situation in that we are dealing with the same topological operators and we can thus let our framework to the remeshing.

We can also follow the idea of Kobbelt [43] and perform $\sqrt{3}$-adic splits. Again, we can reuse the basic topological operators from Section 3.2. (The 1–to–3 split can be implemented similar to the edge–split operation, i.e., we perform a topological 1–to–3 split of a triangle, place the newly inserted vertex on one of corner vertices of the triangle which was just split and let the relaxation operator handle the repositioning.) Note that due to the hierarchical approach, the particle system converges quickly and we need just a few relaxation steps until the length of the relaxation vectors falls below some user defined threshold.

For both methods, we have to make one restriction in order to preserve the skeleton of $\mathcal{M}$. As mentioned in Section 3.3.3, a skeleton-edge is not allowed to flip in the first place. However, applying two refinement steps at once leads to a complete dyadic or triadic split of all triangles (and edges). Thus we have a 1–to–1 correspondence of edges from the coarser level to those on the finer level and consequently are able to preserve the skeleton. The limitation however is, that we always have to perform two refinement steps at once and might eventually end up with a model that is finer and thus more complex than required by the application.

Figure 4.2 shows one example of a $\sqrt{3}$ remeshing. Here a user first tags feature points on the original model and and the algorithm generates the coarse base domain(upper right. The feature points ensure that vertices that are important to the user are present in are present in all models of the hierarchy. The figure shows the coarsest model we can generate with our algorithm while preserving the feature points. In the next step we start refining by applying the topological $\sqrt{3}$ split operator twice per step (middle row). The lower row shows the final remesh. Even at highest resolution (77KΔ), the response times for this model never exceeded 4-5 seconds.

4.4 Interactive Multiresolution Modeling with Changing Connectivity

Having all the means at hand, we are able to focus on (interactive) multiresolution modeling, where a designer modifies a given surface using editing operations. There are many different ways and means to modify a given surface. Traditionally, geometric modeling is based on polynomial surface representations [23, 50, 55]. However, while special polynomial basis functions are well suited for describing and modifying smooth triangular or quadrilateral *patches*, it turns out to be rather difficult to smoothly join several pieces of a composite surface along common (possibly trimmed) boundary curves. As flexible patch layout is crucial for the construction of non–trivial geometric shapes, spline–based modeling tools spend much effort on maintaining the global smoothness of a surface. The situation is simpler for triangle meshes.

This section discusses our approach that make freeform and multiresolution modeling with dynamic connectivity available for triangle meshes. Opposed to splines, where the control vertices provide a convenient way to smoothly edit the surface, this is a challenging task, since plain triangle meshes do not have any reasonable control mechanism to perform large scale edits. Before we describe in detail, how intuitive modeling metaphors for triangle meshes can be accomplished, we describe the general requirements a modeling tool should satisfy.

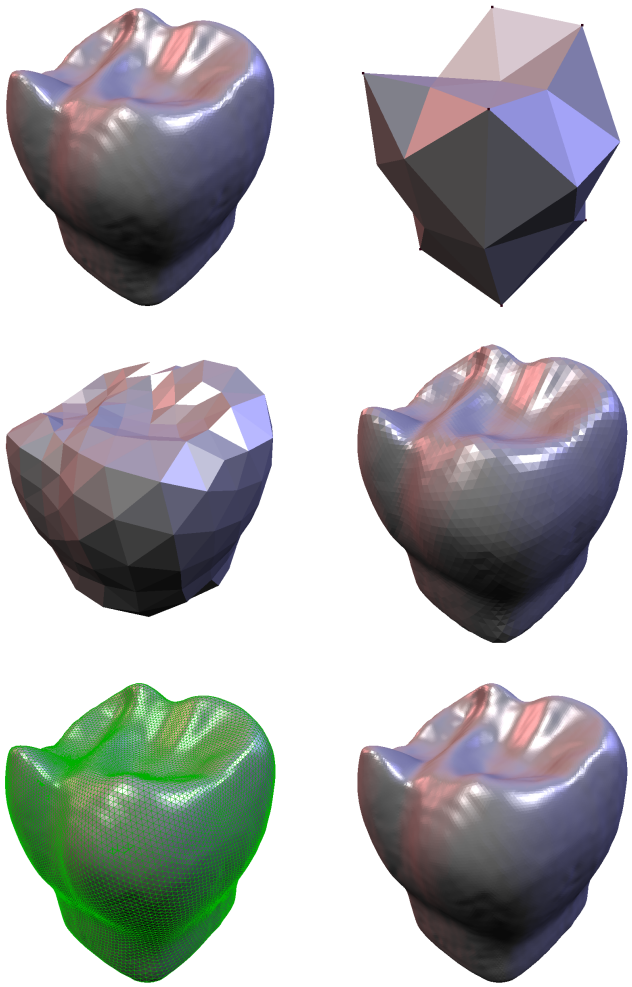

Figure 4.2: $\sqrt{3}$-remeshing of a tooth-model (original upper left). The coarse base mesh with user defined feature points (top right) is refined (middle row) until the target resolution is reached (bottom left). The final remesh (bottom right) cannot be distinguished visually from the original mesh, but now has subdivision connectivity.

Intuitive I.e., editing the overall shape with an easy to use control mechanism (cf. control vertices of splines) in a broad, smooth manner while

preserving little features residing on the surface should be possible.

Independent The editing interface should abstract from the underlying mesh representation, since in general a designer is not interested in how the surface is actually represented.

Interactive This is crucial, since a designer heavily depends on immediate visual feedback when performing further edits.

We distinguish between two different approaches: *Freeform modeling* is the process of modifying subregions of the surface in a *smooth* manner whereas the notion of *multiresolution modeling* describes edits where we can additionally preserve little geometric features [86, 46]. Our work aims at multiresolution modeling. In addition to merely changing the geometry of the model we also change the underlying representation, i.e., the connectivity of the triangle mesh itself. This is done in order to adapt it to the modeling operation. However, we use freeform modeling for the underlying modification of the surface. Detail preservation and the changing connectivity is done on top of the modified freeform surface.

We split the description of our approach into three parts. In the first part of this chapter we will deal with freeform modeling. This can be done with the help of *discrete fairing* or *subdivision* respectively. We will review these methods just very briefly. Here we are aiming at developing a control mechanism (similar to the control–points of spline–based methods) for these methods only and thus do not explain these methods in full detail.

After that we will first show how to build a hierarchical structure for semi–regular meshes before we describe how this can be done for arbitrary, i.e., unstructured meshes. Combined with freeform modeling, this enables us to perform true multiresolution modeling.

In the third part we will extend the multiresolution modeling with a dynamic remeshing component, which eventually completes the modeling approach.

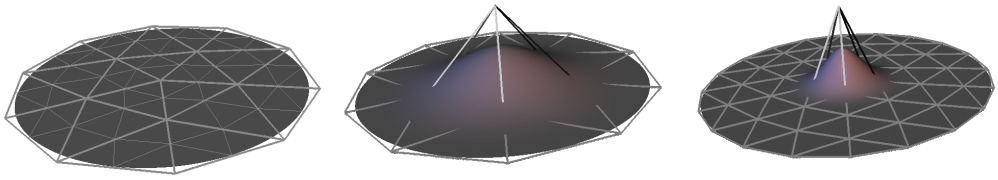

Figure 4.3: A simple subdivision–surface (left) is modified by moving the vertices of corresponding control meshes. Editing the coarse control mesh leads to a wide "bump" (middle), whereas altering a vertex on a finer level affects a smaller area (right).

4.4.1 Freeform modeling

In Chapter 2 we have briefly mentioned *Subdivision schemes* that can also be considered as the algorithmic generalization of classical spline techniques, enabling control meshes with arbitrary topology. They provide easy access to globally smooth surfaces of arbitrary shape by iteratively applying simple refinement rules to the given control mesh. A coarse–to–fine hierarchy of meshes generated by this process quickly converges to a smooth limit surface. For most practical applications, the refined meshes are already sufficiently close to the smooth limit after only a few refinement steps. Since our modeling approach extends the classical approach for semi-regular meshes, lets for now assume we are given a semi–regular mesh $\mathcal{S}_n$. $\mathcal{S}_n$ was generated by applying some subdivision operator $\mathbf{S}$ to a base mesh $\mathcal{S}_0$, and we want to modify $\mathcal{S}_n$ with specific support. The usual way to implement this operation is to run a decomposition scheme several steps until the desired resolution level corresponding to the mesh $\mathcal{S}_i$ is reached. In our setting, this can simply be done by sub-sampling, i.e., we just switch to $\mathcal{S}_i$. On this level the mesh $\mathcal{S}_i$ is edited. Applying $\mathbf{S}$ to the modified mesh $\mathcal{S}_i'$ $(n-i)$–times yields the final result. This operation can be performed quite efficiently due to the simplicity and numerical robustness of $\mathbf{S}$. Figure 4.3 illustrates the varying support of modifications at different levels. The major drawback of this procedure is the fact, that edits are restricted to vertices residing on a specific level. However,

one can fake low–frequency modifications by moving a group of vertices from a finer level simultaneously. But besides being cumbersome, this annihilates the mathematical elegance of the multiresolution representation.

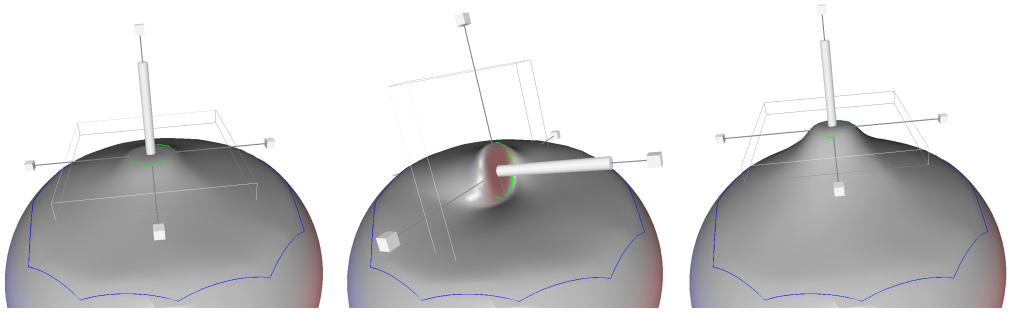

Figure 4.4: Freeform edits for unstructured meshes (cf. [46]): The dark line indicates the area which is subject to the modification. The bright line defines the handle geometry which can be moved by the designer (middle,right). Both boundaries can have an arbitrary shape and hence they can, e.g., be aligned to geometric features in the mesh. The dark and the bright line impose $\mathcal{C}^1$ and $\mathcal{C}^0$ boundary conditions to the mesh respectively and the modified smooth version is found by discrete fairing while preserving these conditions. Notice, that the designer can apply arbitrary modifications to the handle polygon and does not have to take the mesh connectivity into account.

In order to apply global and smooth modifications to *arbitrary (manifold) triangle meshes* we make use of a slightly different approach. We first define the area of influence, the boundary of the modification, and a set of vertices (inside this area) that will remain fixed. This defines the boundary conditions for the calculation of a smooth or fair surface that follows the principle of the simplest shape [70]. Significant amount of work has been dedicated to this problem. We refer to [49] for an in depth discussion, since our work only marginally touches this area. The key idea for the actual freeform modeling is now relatively simple and can roughly be stated as follows:

Define the edit by altering boundary conditions to the mesh and

recalculate the fair surface by solving the corresponding optimization problem.

Fig. 4.4 shows a convenient way how boundary conditions can be defined by the user. However, more sophisticated methods can easily be derived. The following sections show how to apply discrete fairing in the context of interactive modeling.

Note that since interactivity is crucial, an efficient solver for the chosen fairing scheme has to be available. We use multi–level schemes to solve the problem on a coarse level first and use this solution to predict initial values for a solution on the next refinement level [31]. In our case, we can use incremental mesh decimation (cf. Sec. 2.7) to build a fine–to–coarse hierarchy of the mesh in such a way that intermediate meshes can be used to solve the optimization problem (`OP`). The optimization algorithm in our case can be described in the following way.

```
go to coarsest level
solve OP directly
Repeat:
  reinsert some vertices
  solve OP in vicinity of new vertices
Until mesh is reconstructed
```

Note, that we do not make use of the relaxing procedure we have defined, but merely use plain mesh simplification here. However, we can exploit one feature of the remeshing framework to stabilize the calculation of the fair surface. We can remesh the input mesh prior to the modeling just as we have done it during the interactive remeshing. That way we feed a fairly regular mesh to the optimization problem. This reduces the computational effort and at the same time enhances the numerical robustness of many algorithms.

For now we have described the freeform modeling for arbitrary triangle meshes, a way to modify triangle meshes in a smooth manner. In the remainder of this chapter we use this modification mechanism and define multiresolution modeling on top of it.

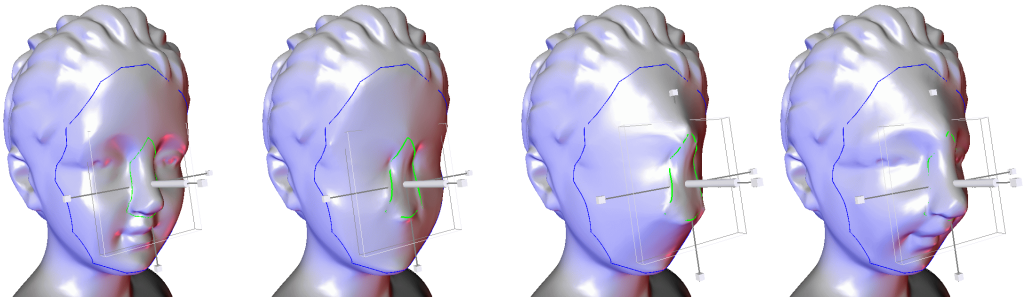

Figure 4.5: A flexible metaphor for multiresolution edits. On the left, the original mesh is shown. The smooth version of the original mesh is found by applying discrete fairing while preserving the boundary constraints (dark and bright line, cf. Fig. 4.4). The center left shows the result of the optimization. The geometric difference between the two left meshes is stored as detail information with respect to local frames. Now the designer can move the handle polygon and this changes the boundary constraints for the optimization. Hence the discrete fairing generates a modified smooth mesh (center right). Adding the previously stored detail information yields the final result on the right. Since we can apply fast multi-level smoothing when solving the optimization problem, the modified mesh can be updated with several frames per second during the modeling operation. Notice that all four meshes have the same connectivity.

4.4.2 Multiresolution modeling

The previous section shows how to perform freeform modeling on triangle meshes. Let us now assume we want to modify the face of the bust model (see Fig. 4.5) and we would, e.g., like to shift its nose. This could be accomplished with the above methods, but the face would lose its features like eyes and mouth since this detail information would be removed by the optimization process. In order to enable such types of edits, we extend freeform modeling to *multiresolution modeling*. This means that we have to be able to distinguish between high–frequency detail information that has to be preserved and the low–frequency shape we want to edit. This is where multiresolution representations for triangle meshes come in. In this chapter we already got

to know two different ways to build hierarchies. Coarse–to–fine hierarchies in the context of semi-regular remeshing and fine–to–coarse hierarchies in the section about mesh simplification. In the context of multiresolution modeling however, we do not want hierarchies of different *coarseness*, i.e., with varying triangle count, but of different *smoothness*. Nevertheless, it turns out, that both types of hierarchies are closely related.

Given an arbitrary surface $\mathcal{S}_m$, a multiresolution *decomposition* consists of a sequence of topologically equivalent surfaces $\mathcal{S}_{m-1}, \ldots, \mathcal{S}_0$ with decreasing level of geometric detail. The difference $\mathcal{D}_i = \mathcal{S}_{i+1} - \mathcal{S}_i$ between two successive surfaces is the *detail* on level i which is added or removed when switching between the two approximations. The reconstruction $\mathcal{S}_m = \mathcal{S}_i + \mathcal{D}_i + \ldots + \mathcal{D}_{m-1}$ of the original surface $\mathcal{S}_m$ can start on any level of detail $\mathcal{S}_i$. Multiresolution modeling means that on some level of detail, the surface $\mathcal{S}_i$ is replaced by $\mathcal{S}_i'$. This operation does not have any effect on $\mathcal{S}_0, \ldots, \mathcal{S}_{i-1}$ but $\mathcal{D}_{i-1}$ and hence $\mathcal{S}_{i+1}, \ldots, \mathcal{S}_m$ change since the (unchanged) detail information $\mathcal{D}_i, \ldots, \mathcal{D}_{m-1}$ is now added to the modified base surface $\mathcal{S}_i'$ for the reconstruction of $\mathcal{S}_m'$. In order to guarantee the intuitive preservation of the shape characteristics after a modification on some lower level of detail, this basic setting has to be extended in the sense that the detail information $\mathcal{D}_i$ is encoded with respect to *local frames*. These frames are aligned to the surface geometry of $\mathcal{S}_i$ [24, 23, 46, 86]. The next section will put the calculation of the local frames into a broader perspective and will derive a multiresolution representation for arbitrary triangle meshes.

4.4.3 Robust Multi-Band Detail Encoding

Again like their real world equivalent, surfaces often carry detail information on various scales and often give the surface its characteristic look. Whenever one wants to change the overall shape of this surface, these characteristic properties should still be part of the altered surface or at least they should be preserved as good as possible. To give an example, Figure 4.6 shows Max-Planck's head model at three different *frequencies of geometric detail*. If one wants to perform a smooth deformation on the geometric level of the rightmost image, i.e., a deformation that affects the whole face, the levels of

higher geometric detail like, e.g., the skin texture (cf. images to the left of that level) should be preserved.

In this section we want to describe a hierarchical representation of a geometric model. We generate a sequence of differently smooth versions of the original model. as described in the previous section. This way we can apply modifications at different geometric scales while preserving detail information of the higher levels.

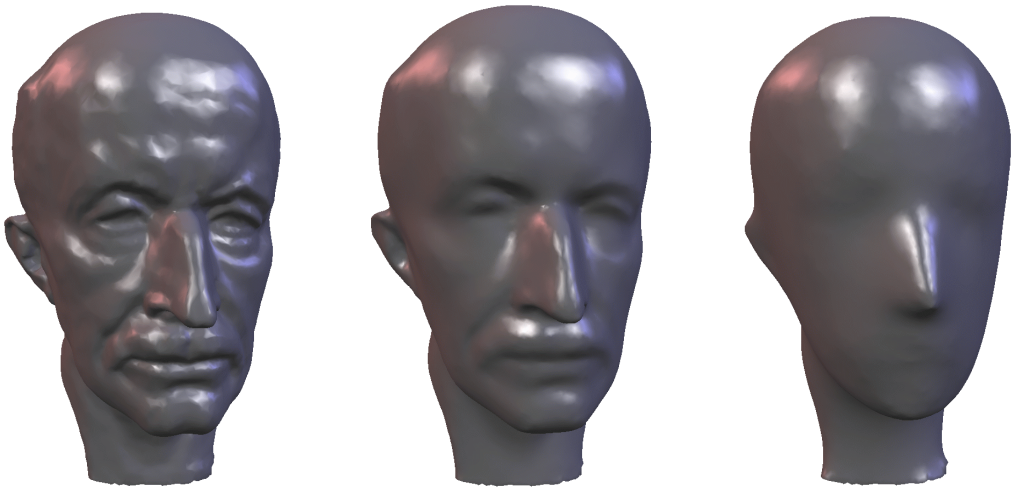

Figure 4.6: Different scales of detail on Max-Planck's head. Whereas the leftmost image shows the head with full detail, i.e., high geometric frequencies, the geometric frequencies decrease in the middle and in the right image.

During the last years, hierarchical representations of geometric shape have become the de facto standard for those purposes. The basic idea is to encode a high-frequency detail level relative to a coordinate frame induced by a coarser approximation of the original shape such that modifications on a coarser level can be propagated to the finer ones. Pioneering work in this area was done by Forsey and Barthels in [24, 23], where they used hierarchical polynomial patches (H-Splines) to represent and edit a surface. Though splines have a straightforward shape control mechanism based on control vertices, it is well-known to be rather complicated to preserve boundary conditions when handling complex geometry.

This is one of the reasons, why the interest in surface representations based on triangular meshes increased over the last years. Generalizing the patch-based concepts, the wide family of subdivision techniques (cf. [18] for an overview) start with a coarse base mesh approximating a geometric shape of arbitrary topology and refine it iteratively. An exponential number of vertices is introduced to capture finer detail information, until a prescribed tolerance is reached. When refining a mesh, the position of the inserted control-vertices is predicted by the smoothing-rule of a subdivision-scheme. A detail vector (relative to linked to the coarser level (and follows modifications, if the global shape changes). Storing the base mesh and the sequence of detail vectors for a fixed subdivision scheme leads to a hierarchical representation of the original shape [61, 86]. In order to generate a smooth low–frequency approximation of $\mathcal{S}_m$, we simply suppress the detail reconstruction starting from some intermediate level j ($\mathcal{D}_i = 0, i \geq j$).Note that the smooth mesh and the mesh with reconstructed detail information share the same connectivity, but are merely differently smooth versions of the original shape.

Within the scope of this thesis however, we are in a more general context, i.e., we are dealing with meshes of arbitrary connectivity. Of course we could apply our semi-regular remeshing framework we have developed to generate this special structure, but this way we would loose the flexibility of arbitrary meshes. Similarly to the technique we have described to generate differently smooth semi-regular meshes, we now generate a hierarchy of differently smooth arbitrary meshes. A popular way for arbitrary meshes is to build the hierarchial structure the other way around, i.e., from fine to coarse. For this, mesh simplification (cf. Section 2.7) can be used. Note that the technique we are going to describe in the following uses differently coarse meshes (with low triangle count) merely to efficiently generate differently smooth versions of a mesh using multilevel methods. The method can be seen as a generalization from semi-regular meshes to arbitrary meshes. All meshes share the same connectivity after they are fully reconstructed. However, the smooth hierarchy of meshes could also be the result of a smoothing scheme that does not change the connectivity of the mesh at all [49].

Multiple levels of resolution are produced by incrementally decimating the fine mesh (cf. 2.7). This is done by applying the half-edge collapse as decom-

position operator. To capture the detail information, which would be lost otherwise, similar to the semi-regular setting, detail vectors have to be stored. For a hierarchical representation, a proper reconstruction has to be ensured. Hence, we need a base point, where the detail vector could be attached to. In contrast to the subdivision scheme, where the base point is predicted by the subdivision operator, no such point exists for the coarse to fine approach, since the mesh-connectivity does not provide the necessary regular structure. For this reason, a vertex removal is split into two steps. First, the original position is altered such that it is optimal with respect to the optimization problem just as we have described it for the freeform modeling. The second step removes the original vertex and encodes the position with respect to its optimized counterpart. This would require a optimization process for every single vertex. One could also apply the optimization to *all* vertices before storing the detail information to lower the computational costs. This would lead to a two-band representation, i.e., a smoothed version, and the original mesh. Both meshes would be linked by a set of detail vectors. In practice, a multi-band hierarchy, similar to a level of detail representation would be desirable. This could reflect the multiple scales of features on the surface to stabilize the modeling-process on the one hand and keep down the costs on the other hand.

Hence, to build an appropriate hierarchical structure of a triangular mesh for our modeling purposes, we have to solve two problems. First, we have to choose the right intermediate frequency-bands, such that a modification of a coarser level will lead to reasonable changes of the finer ones. On the other hand, the detail has to be encoded with respect to a proper base point, to ensure a stable reconstruction. The following sections discuss several approaches for both problems.

As mentioned before, we cannot simply store the detail vectors with respect to a global coordinate system but have to define them with respect to local frames which are aligned to the low-frequency geometry [24, 23, 62, 65, 77]. This guarantees the intuitive detail preservation under modification of the global shape. Usually, the associated local frame for each vertex has its origin at the location predicted by the reconstruction operator with suppressed detail. However, in many cases this can lead to rather long detail vectors

with a significant component within the local tangent plane. Since we prefer short detail vectors for stability reasons, it makes sense to use a different origin for the local frame. In fact, the optimal choice is to find that point $\mathbf{q}$ on the low-frequency surface whose normal vector points directly to the original vertex $\mathbf{p}$. In this case, the detail is not given by a three dimensional vector $(\triangle x, \triangle y, \triangle z)^T$ but rather by a base point $\mathbf{q} = \mathbf{q}(u, v)$ on the low-frequency geometry plus a scalar value h for the displacement in normal direction. If a local parameterization of the surface is available then the base point $\mathbf{q}$ can be specified by a two-dimensional parameter value (u, v).

The general setting for detail computation is that we have given two meshes $\mathcal{M}_{m+1}$ and $\mathcal{M}'_{m+1}$ where $\mathcal{M}_{m+1}$ is the original data while $\mathcal{M}'_{m+1}$ is reconstructed from the low-frequency approximation $\mathcal{M}_m$ with suppressed detail, i.e., for coarse-to-fine hierarchies, the mesh $\mathcal{M}'_{m+1}$ is generated by applying a stationary subdivision scheme and for fine-to-coarse hierarchies $\mathcal{M}'_{m+1}$ is optimal with respect to some global bending energy functional. Encoding the geometric difference between both meshes requires to associate each vertex $\mathbf{p}$ of $\mathcal{M}_{m+1}$ with a corresponding base point $\mathbf{q}$ on the continuous (piecewise linear) surface $\mathcal{M}'_{m+1}$ such that the difference vector between the original point and the base point is parallel to the normal vector at the base point. In order to do so, an arbitrary point $\mathbf{q}$ on $\mathcal{M}'_{m+1}$ can be specified by a triangle index i and barycentric coordinates within the referred triangle.

To actually compute the detail coefficients, we have to define a normal field on the mesh $\mathcal{M}'_{m+1}$. The most simple way to do this is to use the normal vectors of the triangular faces for the definition of a piecewise constant normal field. The point $\mathbf{q} = \mathbf{q}(i, u, v)$ can be computed efficiently by a simple projection and works fine, if the resulting coefficient is short compared to the edges of the assigned triangle and if $\mathcal{M}'_{m+1}$ is sufficiently smooth. But since the orthogonal prisms spanned by a triangle mesh do not completely cover the vicinity of the mesh, we have to accept negative barycentric coordinates for the base points if it does not lie within such a prism. This leads to non-intuitive detail reconstruction if the low-frequency geometry is modified (cf. Fig. 4.7).

A technique used in [46] is based on the construction of a local quadratic interpolant $\mathbf{F}$ to the low-frequency geometry. For a vertex $\mathbf{p} \in \mathcal{M}_{m+1}$ it is

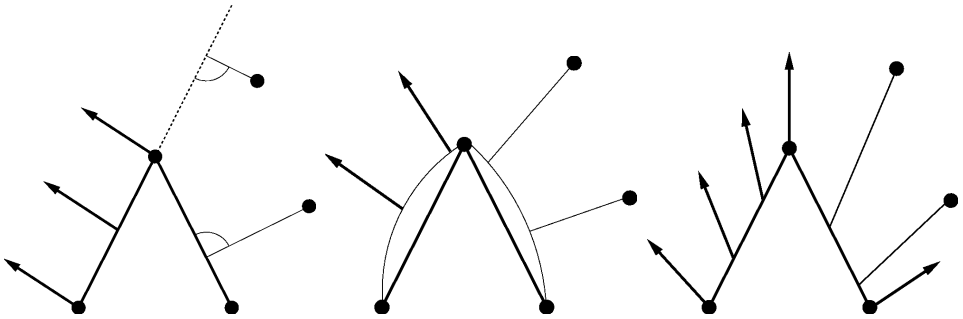

Figure 4.7: The position of a vertex in the original mesh (high-frequency geometry) is given by a base point on the low-frequency geometry plus a displacement in normal direction. There are many ways to define a normal field on a triangle mesh. With piecewise constant normals (left) we do not cover the whole space and hence we sometimes have to use virtual base points with negative barycentric coordinates. The sketch shows, that this can lead to non intuitive reconstructions, if the "base mesh" is for example flattend out. The use of local quadratic patches and their normal fields (center) somewhat improves the situation, but problems still occur since the overall normal field is not globally continuous. Such difficulties are avoided if we generate a Phong-type normal field by blending estimated vertex normals (right).

based on the closest triangle $\mathbf{T} \in \mathcal{M}'_{m+1}$ and its adjacent vertices, which can be found in constant time by a simple local search procedure, starting from $\mathbf{p}$'s corresponding vertex $\mathbf{p}' \in \mathcal{M}_{m+1}$. Since now a local parameterization is given, parameter values (u,v) defining the base point $\mathbf{q}$ can be found by a multidimensional Newton-iteration. We start off from the center of $\mathbf{T}$ at $\mathbf{q}_0 = \mathbf{F}(\frac{1}{3}, \frac{1}{3}), \mathbf{q}_{n+1}$ is defined by the projection of $\mathbf{p}$ into the tangent plane of $\mathbf{F}$ at $\mathbf{q}_n$. In terms of parameter values (u, v), this leads to the simple update rule $(u_{n+1}, v_{n+1}) \leftarrow (u_n, v_n) + (\triangle u, \triangle v)$, where $(\triangle u, \triangle v)$ is the solution of the linear system

$$\begin{pmatrix} F_u^T F_u & F_u^T F_v \\ F_u^T F_v & F_v^T F_v \end{pmatrix} \begin{pmatrix} \triangle u \\ \triangle v \end{pmatrix} = \begin{pmatrix} F_u^T d \\ F_v^T d \end{pmatrix} \tag{4.1}$$

with detail vector $\mathbf{d} = \mathbf{p} - \mathbf{q}_n$, which is perpendicular (within a pre-

scribed tolerance) to $\mathbf{F}(u_n, v_n)$ after a few steps. The absolute value of the displacement-coefficient h is set to $\|\mathbf{d}\|$ and has to be multiplied by -1 if $d^T(f_u(u_n, v_n) \times f_v(u_n, v_n)) < 0$. Although this reduces the number of pathological configurations with negative barycentric coordinates for the base point, we still observe artifacts in the reconstructed high-frequency surface which are caused by the fact that the resulting global normal field of the combined local patches is not continuous (cf. Fig 4.8 center).

We therefore propose a different approach which adapts the basic idea of Phong-shading [22] where normal vectors are prescribed at the vertices of a triangle mesh and a continuous normal field for the interior of the triangular faces is computed by linearly blending the normal vectors at the corners. We use the same search procedure as described above and obtain a triangle $\triangle(\mathbf{a}, \mathbf{b}, \mathbf{c})$ with the associated normal vectors $N_\mathbf{a}$, $N_\mathbf{b}$, and $N_\mathbf{c}$. For each interior point

$$\mathbf{q} \;=\; \alpha\,\mathbf{a} + \beta\,\mathbf{b} + \gamma\,\mathbf{c}$$

with $\alpha + \beta + \gamma = 1$ we find the associated normal vector $N_\mathbf{q}$ by

$$N_\mathbf{q} \;=\; \alpha\,N_\mathbf{a} + \beta\,N_\mathbf{b} + \gamma\,N_\mathbf{c}.$$

Hence, we are searching for scalars α, β such that

$$\mathbf{p} \;=\; \mathbf{q} + \lambda N_\mathbf{q}$$

Computing α and β is slightly more involved than it seems at a first glance and we have thus included a detailed description of this calculation in Appendix B.

As a result we get a similar update rule as described in Equation 4.1. Starting with $(\alpha_0, \beta_0) = (\frac{1}{3}, \frac{1}{3})$, the difference $(\triangle\alpha, \triangle\beta)$ between two consecutive steps can be denoted as follows.

$$\begin{aligned} \triangle\alpha &= (F_u^T F_v \cdot F_v^T F - F_v^T F_v \cdot F_u^T F)/s \\ \triangle\beta &= (F_u^T F_v \cdot F_u^T F - F_u^T F_u \cdot F_v^T F)/s \end{aligned}$$

with $s = F_u^T F_u \cdot F_v^T F_v - (F_u F_v)^2$.
In case one of the barycentric coordinates of the resulting point $\mathbf{q}$ is negative,

we continue the search for a base point in the corresponding neighboring triangle. Since the Phong normal field is globally continuous we always find a base point with positive barycentric coordinates. Fig. 4.7 depicts the situation schematically and Fig. 4.8 shows an example edit where the piecewise constant normal field causes mesh artifacts which do not occur if the Phong normal field is used.

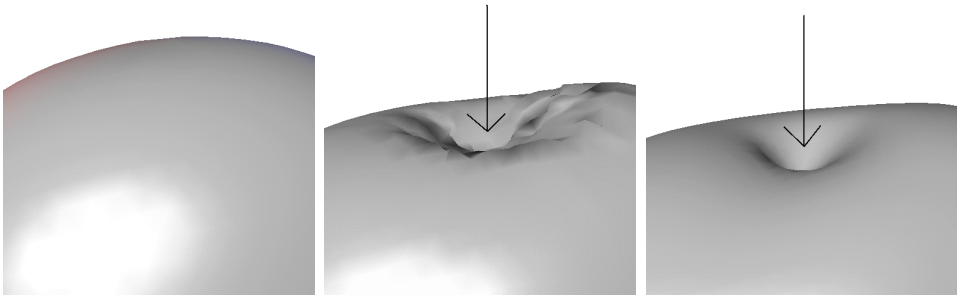

Figure 4.8: The original shape (left) is modified by pushing a single vertex while minimizing a membrane energy functional. A piecewise linear normal field leads to undesirable mesh artifacts (middle), while storing detail information with respect to a Phong normal field (left) performs a satisfying modification.

4.4.4 Hierarchy levels

For coarse-to-fine hierarchies the levels of detail are determined by the uniform refinement operator. Starting with the base mesh $\mathcal{M}_0$, the mth refinement level is reached after applying the refinement operator m times. For fine-to-coarse hierarchies there is no such canonical choice for the levels of resolution. Hence we have to figure out some heuristics to define such levels.

In [46] a simple two-band decomposition has been proposed for the modeling, i.e., the high frequency geometry is given by the original mesh and the low-frequency geometry is the solution of some constrained optimization problem. This simple decomposition performs well if the original geometry can be projected onto the low-frequency geometry without self-intersections.

Fig 4.9 schematically shows a configuration where this is not satisfied and consequently the detail feature does not deform intuitively with the change of the global shape.

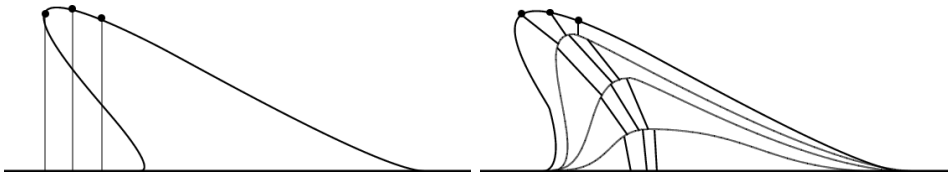

Figure 4.9: If the high-frequency detail cannot be projected onto the successive level (top), intermediate levels have to be inserted to guarantee a feasible detail reconstruction (bottom).

This effect can be avoided by introducing several intermediate levels of detail, i.e., by using a true multi-band decomposition. The definition of the Phong-type normal field introduced in the last section provides the means to guarantee a stable reconstruction. The number of hierarchy levels has to be chosen such that the $(i+1)$st level can be projected onto level i without self-intersection. Detail information has to be computed for every intermediate level.

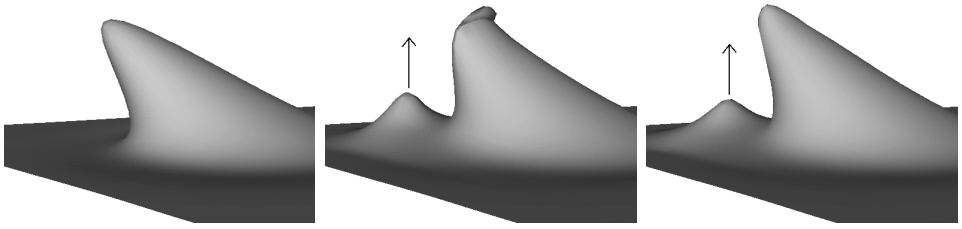

Figure 4.10: Starting from the original shape (left), a two-band decomposition (middle) can lead to long detail-vectors and hence to exaggerated modifications or even self-intersections for relatively small edits. Multiple levels of detail avoid these artifacts and the modifications behave in a natural fashion (right).

Intermediate levels can be generated by the following algorithm. We start with the original mesh and apply an incremental mesh decimation algorithm which performs a sequence of edge collapse operations. When a certain mesh complexity is reached, we perform the reverse sequence of vertex split operations which reconstructs the original mesh connectivity. The position of the re-inserted vertices is found by solving a global bending energy minimization problem [42, 46, 28]. The mesh that results from this procedure is a smoothed version of the original mesh where the degree by which detail information has been removed depends on the target complexity of the decimation algorithm.

Suppose the original mesh has n_m vertices, where m is the number of intermediate levels that we want to generate. We can compute the meshes $\mathcal{M}_m, \ldots, \mathcal{M}_0$ with fewer detail by applying the above procedure where the decimation algorithm stops at a target resolution of $n_m, \ldots, n_0$ remaining vertices respectively. The resulting meshes yield a multi-band decomposition of the original data. When a modeling operation changes the shape of $\mathcal{M}_0$ we first reconstruct the next level $\mathcal{M}_1'$ by adding the stored detail vectors and then proceed by successively reconstructing $\mathcal{M}_{i+1}'$ from $\mathcal{M}_i'$.

The remaining question is how to determine the numbers n_i. A simple way to do this is to build a geometric sequence with $n_{i+1}/n_i = \text{const}$. This mimics the exponential complexity growth of the coarse-to-fine hierarchies. Another approach is to stop the decimation every time a certain average edge length $\bar{l}_i$ in the remaining mesh is reached.

A more complicated heuristic tries to equalize the sizes of the differences between levels, i.e., the sizes of the detail vectors. We first compute a multi-band decomposition with, say, 100 levels of detail where we choose $\sqrt[i]{\bar{n}_i} = \text{const}$. For every pair of successive levels we can compute the average length of the detail vectors (displacement values). From this information we can easily choose appropriate values $n_j = \bar{n}_{i_j}$ such that the geometric difference is distributed evenly among the detail levels.

In practice it turned out that about five intermediate levels is usually enough to guarantee correct detail reconstruction. Fig. 4.10 compares the results of a modeling operation based on a two-band and a multi-band decomposition.

We have presented a new method to encode high-frequency detail with re-

spect to a low-frequency base mesh. Now, we are able to perform a robust true multi-band decomposition for a given fine triangular mesh of arbitrary connectivity. This leads to intuitive modifications of global shapes under preservation of detail features. However, the user can still apply particular edits, where undesirable effects like self intersection of detail vectors during the reconstruction process happen. A promising volume preserving approach to address this challenge is presented in [7]. However, this approach is computationally more involved than the Phong-type normals. Since we are aiming at interactive rates for our modeling tool, we decided to stick with the Phong approach.

4.4.5 Multiresolution modeling with changing connectivity

In the last section of this chapter we want to discuss several approaches, how in addition to performing multiresolution edits one can change the connectivity of the resulting mesh. This is in particular necessary, if the edit induces severe stretches or compressions to certain parts of the mesh which might lead to badly shaped triangles. We distinguish between two different approaches. Approach one is rather straightforward once the remeshing framework and multiresolution modeling is available. Nonetheless we primarily use this approach in our interactive modeling tool. The key idea is rather simple and can be described in one sentence:

> Take the reconstructed modified mesh as it comes out of the multiresolution modeling process and perform an interactive remeshing afterwards.

In other words the remeshing and the modeling run independently of each other. In this method the modeling starts off with the input mesh $\mathcal{D}$ and the hierarchy of meshes meshes $\mathcal{S}_i$ is constructed for the multiresolution modeling part. After that we create a copy $\mathcal{M}$ of the input mesh and establish a one-to-one link between $\mathcal{M}$ and $\mathcal{D}$. $\mathcal{M}$ will later be the final remeshed result and $\mathcal{D}$ is equal to the the fully reconstructed mesh hierarchy $\mathcal{S}_m$ before any

modeling operation is performed. The advantage of this method is twofold. On the one hand side due to the fact that the two methods run independently of each other it is rather simple to put into practice. For instance in our implementation we just transmit the altered vertex positions from the modeling part to the remeshing part of the application. We change the base mesh with respect to the new coordinates and do one remeshing iteration. On the other hand the remeshing is quite efficient since only regions where the modeling is done require the full calculation effort, the other regions remain more or less static. This way we get immediate visual feedback due to the fact that the remeshing is done incrementally.

However, the above method performs a true remeshing of the object that undergoes a modeling operation. This effect might not be desired since even in case only a small edit is done, the triangulation might change completely. For this reason we propose a slightly modified method that overcomes this shortfall but sacrifice the relaxation based on local parameterizations and pre-calculate a global parameterization of the area which is to be modified instead. This user defined area is in practice significantly smaller than the object itself and a global parameterization can be constructed. More precisely we project $\mathcal{D}$'s boundary of the modeling area to a 2D polygon. The 2D positions of $\mathcal{D}$'s vertices in the interior of this boundary are calculated with the exactly same relaxation operator that is subsequently used for the remeshing process itself. Typically the precalculation of this parameterization takes only a fraction of a second for moderately large modeling areas of 5 - 10K$\triangle$. By choosing this parameterization we ensure, that vertices of $\mathcal{M}$ do not change positions until the user performs a modification. The remaining modeling algorithm works just as in method one.

4.4.6 Discussion

In this section we have shown how to do interactive multiresolution modeling with triangle meshes of changing connectivity. The main advantage of our method is the fact that we can clearly separate the three different stages of the scheme that eventually allow the edits and that in each stage we can easily plug in different algorithms that do not interfere with the other two stages.

We have shown how to smoothly edit user defined areas on arbitrary triangle meshes. We have extended this freeform modeling to true multiresolution modeling with fixed connectivity by generating a multiresolution hierarchy of differently smooth meshes that are defined as offsets to some smooth base mesh. Eventually we have taken the mesh with the multiresolution edits we have proposed two methods to adapt the connectivity of the mesh to the users edits.

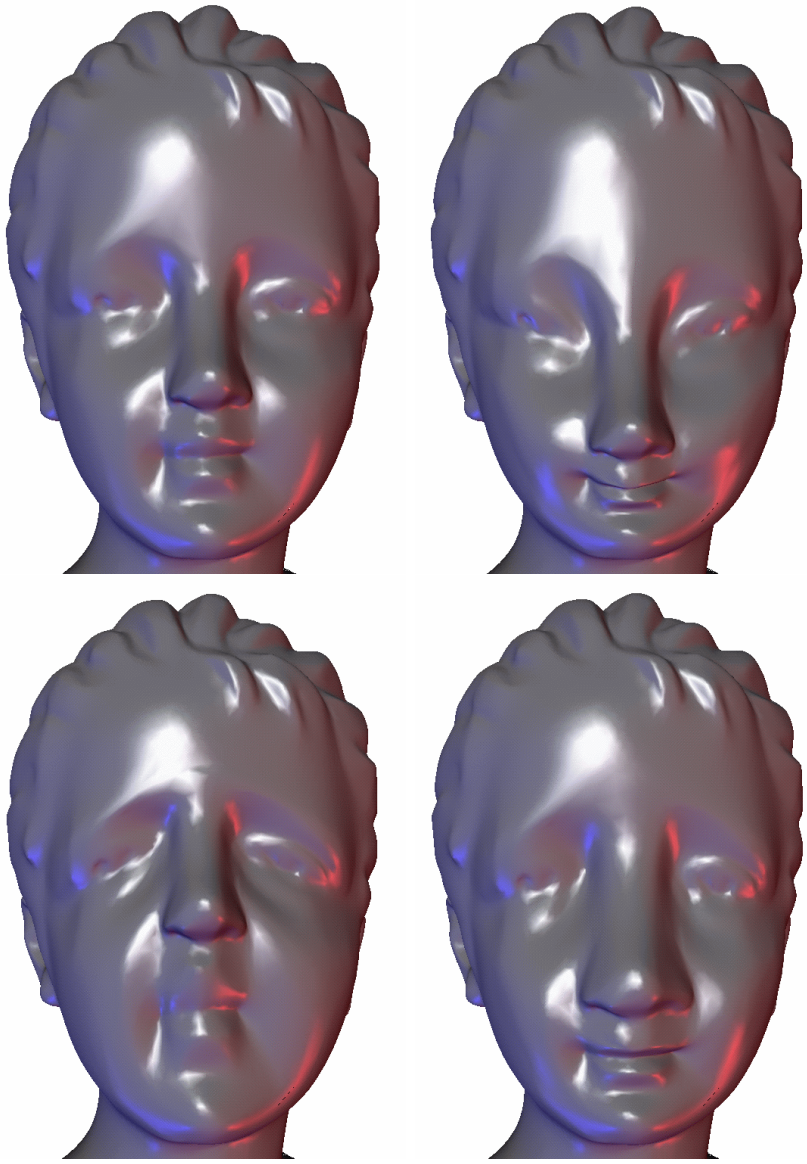

Figure 4.11: Multiresolution editing of a bust (62k triangles,left). The handle lies around the nose, the whole face is marked as the area of influence. The edits where achieved by scaling and translating the nose (from left to right).

Chapter 5

Conclusion and Future Work

In this thesis we presented a new remeshing framework for the optimization of triangular meshes. In the following we summarize our main contributions and conclude by showing directions for future research.

In the first part we presented our incremental remeshing framework. This includes the optimization of the mesh connectivity based on the DCM approach, the optimization of the vertex distribution by means of a particle system as well as a method for preserving features like sharp edges. We see the main advantage of our work in the fact that the framework is versatile in the sense that we can use it for many types of applications and that it can easily be adapted by assembling the components in different orders and ways or by plugging in different types of operators.

Our particle system requires only local parameterization of the original surface. Constructing a global parameterization can explicitly be avoided and we can thus handle topologically complex models, which are particularly challenging for traditional remeshing procedures. In particular, we developed a method that efficiently flattens the local domains to the plane and demonstrated how to incrementally update these domains whenever the connectivity of the mesh changes. Three local topological operators are at the core of our topological optimization method. The method we have presented is particularly simple but can nevertheless generate meshes of sufficient regularity and it is also capable to generate semi-regular meshes. In this context

multiresolution algorithms benefit from the fact that our framework is able to generate particularly coarse base meshes. We introduced a skeleton that a user can attach to the mesh in order to preserve certain features of it. The main advantage of this skeleton is that it is flexible and adapts itself if the underlying mesh changes its resolution. The skeleton, but also the entire framework, does not require to start with an initial remesh $\mathcal{M}$, that is mapped 1-to-1 to the domain mesh $\mathcal{D}$, but we can also start with an *arbitrary* mesh if a link, i.e., a mapping between $\mathcal{M}$ and $\mathcal{D}$ is given. In order to recapture feature lines of the original mesh, we have developed and described an algorithm that runs without any external thresholds, but fully automatically aligns vertices with respect to the feature lines.

The second part of this thesis was dedicated to applications, which we derived from our framework. In this context we mainly wanted to show that the theory can effectively be put into practice and that there is a wide range of applications that our framework covers. We also gave explicit advice how to implement the algorithms (cf. also Appendix). We started with an interactive remeshing technique that can be used to remesh only certain parts of the mesh. Our method is designed in a way that the remesh seamlessly integrates itself into the otherwise untouched rest of the mesh. We briefly showed how to use the framework for mesh simplification and based on this we presented two semi-regular remeshing techniques. Last but not least we presented a three step modeling framework. The user can perform smooth multiresolution deformations to a mesh while the connectivity adapts itself automatically. We put emphasis on the second step where we constructed a hierarchical multiresolution representation of the original mesh that is based on differently smooth approximations to the original mesh. The hierarchy is robust in a sense that one can modify one level of a mesh and the following levels can be reconstructed in a natural manner.

We see several opportunities for future research directions, that can be based on our work. On the technical side we would like further to explore different types of parameterizations and relaxation operators. A promising approach might be to construct a set of local parameterizations that can be smoothly blended into one global parameterization. Due to the fact that our smoothing operator is relatively simple we also think of a hardware implementation of

it, which could drastically improve the speed of our framework.

From an application point of view we would also like to extent our framework. Of course we would like to see more applications besides the approaches for the applications that we have already described. In order to achieve this we would like to further modularize our algorithms. On the one hand this would make implementations reusable and exchangeable. On the other hand we would have a construction kit of algorithms that a user can assemble and custom tailor to his needs. This way one could fine tune own algorithms and generate even better remeshes than it is possible at the moment.

Appendix A

A Framework to Implement Dynamic Connectivity Meshes

Implementing algorithms that are based on dynamic triangle meshes often requires updating internal data-structures as soon as the connectivity of the mesh changes. The design of a class hierarchy that is able to deal with such changes is particularly challenging if the system reaches a certain complexity.

In this chapter we propose a software design that enables the users to efficiently implement algorithms that can handle these dynamic changes while still maintaining a certain encapsulation of the single components.

Our design is based on a callback mechanism. A client can register at some `Info`-object and gets informed whenever a change of the connectivity occurs. This way the client is able to keep internal data up-to-date. Our framework enables us to write small client classes that cover just a small dedicated aspect of necessary updates related to the changing connectivity. These small components can be combined to more complex modules and can often easily be reused. Moreover, we do not have to store related "dynamic data" in one central place, e.g. the mesh, which could lead to a significant memory overhead if an application uses some modules just for a short time.

Compared to *dynamic meshes*, algorithms that are based on *static meshes* are usually easier to implement from a design point of view. Modules that realize such an algorithm just need to store static data, with respect to the

mesh, that does not change during the runtime of the application.

To give a simple example, the application might comprise one module that sets a flag whenever an edge of the mesh gets selected by a user. Since the connectivity of the mesh does not change the module can use an internal `vector-of-bools` that reflects the current status of each edge.

When it comes to meshes that change their connectivity during runtime, however, implementing algorithms that operate on them gets more involved. Data that is stored internally to some module has to be aware of these changes. This task is particularly challenging if the application reaches a certain complexity and one wants to implement components that can be reused. One way of solving such a problem is to store the data (edge-flags) outside of the module, e.g., directly along with the mesh-data-structure that obviously "knows" when its connectivity changes.

One possible way to store data within the mesh are the so called *Meshtraits* or *Meshitems* that have been effectively used in several libraries [8, 39]. This is an excellent approach, if the data that is to be stored is an "established property" that can be reused. This might for instance be a vertex-property such as the valence, a list of all adjacent triangles, texture-coordinates etc. The major advantage is the fact that multiple modules that work with the mesh have easy access to this data and the data is stored/updated in just one place. We found that in this case, the loss of data encapsulation is not a severe restriction (as long as it gets updated correctly). However, the documentation of a module that uses such a Meshitem should explicitly state that it needs a specific Meshitem and a compile-time check[1] should make sure that the appropriate Meshitems are present.

On the other hand, if a module makes use of module specific data that is used only temporarily, inflating/polluting the Meshitems with this data is problematic. In particular for more complex applications one can easily lose control over all the different components, which clearly limits the maintainability. Even worse, the data consumes memory throughout the lifetime of the application even though it might get used for a short period of time only.

For this reason we have developed a framework that enables independent modules of an application to be alerted whenever the connectivity of the mesh

changes. We ensure that the modules do not have to expose internal data to the outside, which facilitates their reusability. Our framework is based on a callback mechanism, all client classes that have to be aware of a changing connectivity supply a common interface (they derive from a common base-class).

In Section A.1 we will build up the callback mechanism that informs a custom tailored client class whenever the connectivity of the mesh changes. Section A.2 illustrates how we can make necessary information available to a client. In Section A.3 we show the concept of informing *multiple* modules of a change in the connectivity while being completely independent of each other. To clarify our concept, we will describe an example application in Section A.4 and point out some extensions of our framework in Section A.5.

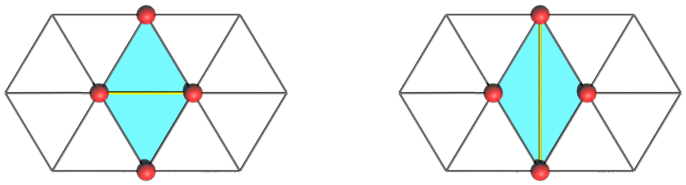

Figure A.1: Illustration of an edge-flip, the example-operator in this chapter. The common edge (red) of the triangles $\triangle ABC$ and $\triangle ACD$ (left) "flips" and forms the new triangles $\triangle ABD$ and $\triangle BCD$ (right).

A.1 The Callback Mechanism

In order to keep our explanations of the class design and the examples as simple as possible, we will restrict ourselves to one single topological operation, the edge-flip (cf. Fig A.1). A fully-fledged library would of course comprise the complete set of operators that change the connectivity of the mesh, e.g., edge-split, edge-collapse, face-split, etc. The additional operators can be integrated into our framework in a similar way as the edge-flip and are thus omitted here.

As we have illustrated in the beginning of this chapter, we will store data that is sensitive to changes of the connectivity of the mesh along with the modules that are using this data, instead of storing them in Meshitems inside the mesh. This way however we are not able to update our data inside some private method of the mesh whenever `Mesh::flipedge()` gets called. Hence, we cannot call `Mesh::flipedge` directly and therefore outsource calls that change the connectivity of the mesh to another class. It makes sure that "dynamic data" is always up-to-date and calls `Mesh::flipedge`.

So instead of calling

```
mesh->flipedge(edge_handle);
```

directly and thus risking that some modules remain clueless about the fact that the connectivity of the mesh has just changed and consequently that the data they store might be outdated, we wrap the call to `Mesh::flipedge` by two calls to methods which the users can custom tailor to their needs. Later we will show how a module can hook into these calls and thus updates its own data whenever an edge-flip occurs.

For now we will show a very simple example of this concept and will later develop a more complex class that we are using in a real-world application.

The heart and soul of our framework is called `Dynamic`, the following listing illustrates the basic form of the callback mechanism.

Listing A.1: `Dynamic::flipedge` shows the basic form of the callback mechanism a user can hook into.

```
struct Dynamic<Mesh> {

  void flipedge(EdgeHandle _edge_handle){
    if(info->preFlip()){
      mesh.flipedge(_edge_handle);
      info->postFlip();
    }
  }
 Mesh&       mesh;
```

```
 InfoBase* info;
};
```

in this example, `InfoBase` is implemented as follows:

```
class InfoBase{
 virtual bool preflip(){return true;}
 virtual void postFlip(){};
};
```

`InfoBase` is meant as a dummy base-class that does nothing but illustrate a certain interface, i.e., in its simplest form the call to `Dynamic::flipedge()` does nothing but flip an edge of the mesh just as a direct call to `Mesh::flipedge()` would have done. So what is the benefit of introducing this additional layer?

The users can derive their own `MyInfo`-class from `InfoBase` and replace `Dynamic::info` with it. This way additional functionality can be implemented and the appropriate `MyInfo::pre/postFlip`-method gets called whenever `Dynamic::flipedge` gets called.

As a simple example we implement a `MyInfo` class as in Listing A.2.

Listing A.2: A simple custom tailored MyInfo class that does nothing but print a message just before and after an edge-flip.

```
class MyInfo : public InfoBase
{
 bool preflip() {cout<<"preFlip";return true;}
 void postFlip() {cout<<"postFlip";}
};
```

Using an instance of `MyInfo` Listing A.3 illustrates how to flip the edge with the EdgeHandle 0 while getting feedback about the `MyInfo::pre/postFlip`-calls.

Listing A.3: Getting feedback about the flip of edge 0 via MyInfo.

```
int main()
{
 Dynamic dynamic;
```

```
  //read mesh & pass it to dynamic

  MyInfo  myinfo;
  dynamic.info = &myinfo;
  dynamic.flipedge(EdgeHandle(0));

  //output of the program:
     preFlip-called postFlip-called
}
```

Please note that you can prevent an edge from flipping by returning *false* in your own `MyInfo::preFlip()`-method. This way you can (in addition to executing edge-flip specific code) influence the optimization process.

Conceptually it would make sense to distinguish between the influence on the optimization process (cf. Section A.4) and the execution of edge-flip specific code. Consequently we should separate between, for example, a class `MyDataUpdateBase` which provides the `pre/postFlip`-interface and another `MyStrategyBase`-class where the users can implement different strategies to influence the optimization process. In practice however, we found it more convenient to have everything in one single `MyInfo`-object.

A.2 Passing Data to `MyInfo`

Up to this point we get informed via our own `pre/postFlip`-method whenever an edge-flip occurs. Of course, a very important fact we are interested in is where the flip actually took place. This is crucial for executing flip-specific code in `MyInfo`. It would be straightforward to pass the edge to the `pre/postFlip`-method as an argument. In practice however, we found that we often need more information about the flip that is going to take place or just took place. For this reason we pass a pointer to a whole `Data`-struct to `pre/postFlip`, which is of the form:

```
struct Data{
  EdgeHandle flip_edge_;
  ...
```

};

and is a member of `Dynamic`. Opposed to our toy-example, this struct also holds all the information that is needed for the other topological operations (cf. Section A.1). This additional information can be exploited for instance if the user needs to know which was the last edge that collapsed prior to the current edge-split etc.

We have made `Data` a member of `Dynamic` instead of storing it directly in `InfoBase` for two reasons: First, for reasons of efficiency since we can use the members of `Data` to store the current edge directly while being in `Dynamic::flipedge()` and thus do not need any additional copy operation.

Second and more importantly, in the next expansion stage of our framework we will introduce the concept of multiple `InfoBase`-objects that work independently of each other. Each of them gets notified by a special instance of an `InfoBase`-object, however, we would like to avoid multiple instances of `Data`. We will see how our framework can benefit from this and point out some implementation issues in the next section.

A.3 Distributing to Multiple Clients

With the current implementation of `MyInfo` we would have to put all code, which has to be executed in order to respond correctly to an edge-flip, into `MyInfo`.

In practice, for a more complex application, a comprehensive, custom tailored `MyInfo` class can easily become a *Blob* [10], i.e., a single class with a large number of attributes and operations. Even worse, we would not have gained much compared to the "embed-all-edge-flip-specific-code-in-the-mesh-class"-approach (cf. introduction of this chapter), meaning that if a module puts flip-specific code into `MyInfo` entails that this module cannot encapsulate and manage its own data anymore.

By using the observer/observable-pattern[25], the diagram in Figure A.2 shows how we can get around the two problems that we have just mentioned. A client can register at the Observer, in our framework we call it `Hub`, and

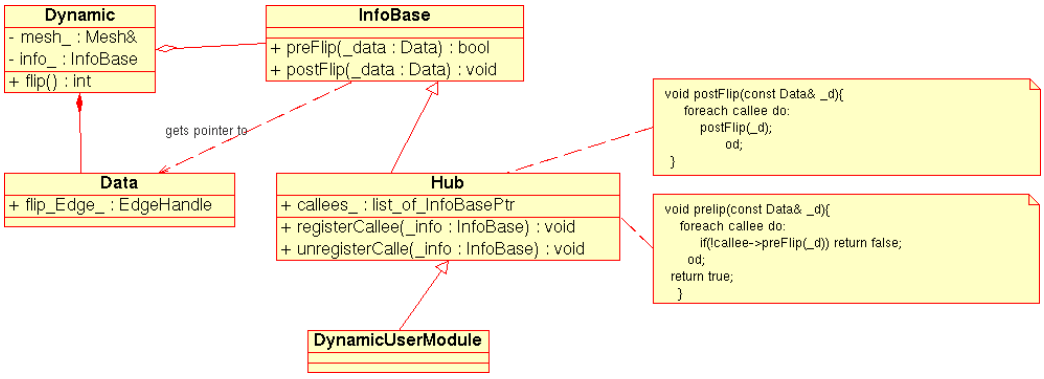

Figure A.2: UML-diagram of our framework. A client class derives from `InfoBase` and registers at the `Hub`. The `Hub` gets passed to `Dynamic` and is thus "aware" of a (scheduled) edge-flip. Additional information about that flip is stored in `Data` which is a member of `Dynamic`. A pointer to this struct is passed to the `pre/postFlip` methods and can thus be exploited by a client class.

gets called whenever an edge-flip is performed. This way we can write small, independent client classes that are aware of changes of the mesh-connectivity. The whole concept works as follows.

We create a `Hub` that is derived from `InfoBase` and let `Dynamic::info` point to it - this way the Hub gets called by `Dynamic::flipedge()`. A client class derives from `InfoBase` and hence supports the `pre/postFlip` interface. Now an object of this client class registers at the `Hub` by passing a reference to it. The `Hub` maintains a list of these client objects, the callees. Whenever `Hub::pre/postFlip` gets called, the `Hub` passes the call to all its callees. Additional information about that flip is available via `Data`. A pointer to this struct is passed to the `pre/postFlip` methods and can thus be exploited by the callees.

Again, using this approach we are able to hide client specific code and data-structures and do not have to expose it to some central instance. We found that the Hub also encourages users of our framework to write small and independent and thus reusable components.

In our current implementation the `Hub` holds a simple list of references to

callees that get called one after the other. However, if the user needs fine grained control over the order of execution of the client classes, one can easily incorporate a more sophisticated calling strategy. This could either be calls by priority, but one can also think of a calling-tree similar to a scenegraph (the `Hub` serve as nodes, the clients are the leaves).

A.4 An Example Application

In this section we will showcase a small application scenario that demonstrates how the framework can be put into practice. We just want to give an impression how the parts of our concept play together and show that the different modules form a closed entity that can be reused easily. Of course, many more applications can be realized similarly to our simple example and we hope that the pool of modules that uses our framework will grow rapidly.

Assume we are given a triangle mesh that contains vertices with high valences, i.e. many edges emanate from these vertices. A multitude of algorithms in geometric modeling prefer vertices with valence six (or at least close to six). The edge-flip is one operator that can reduce this valence-excess (cf. [44, 83] for a detailed description).

In our example we will use two client classes that register at a `Hub` (cf. Section A.3). `ValenceStore` manages the valences of all mesh-vertices - it serves as an example for a module that holds its own data and updates it if the connectivity of the mesh changes.

```
class ValenceStore : public InfoBase{

  // assume the valences are stored in valenceMap_
  // and initialized by the constructor of this class.

  //update valences after an edge_flip
  bool postFlip(const Data& _data){

    EdgeHandle e = _data.flip_edge_;
```

```
    // Update their valences of the four adjacent
    // vertices v_i of e        //      v3
    valenceMap_[v[0]] -=1;      //     / | \
    valenceMap_[v[1]] +=1;      //   v0  | v2
    valenceMap_[v[2]] -=1;      //     \ | /
    valenceMap_[v[3]] +=1;      //       v1
  }

  std::map<VertexHandle,int> valenceMap_;
};
```

Note that we do not have to enumerate all adjacent vertices of a vertex to recalculate its valence, since we know how the valences of the four vertices are affected by an edge-flip.

The second class is an example for exerting influence on the execution of edge-flips. `BalanceStrategy` is a class that calculates the valence-excess of vertices incident to an edge and indicates via `preFlip` if an edge-flip would improve it.

```
class BalanceStrategy : public InfoBase{

  // Pass a ValenceStore object to
  // this class in the constructor

  bool preFlip(const Data& _data){

    EdgeHandle e = _data.flip_edge_;

    // Now get the four adjacent vertices of e and
    //their valences val[0,...,3] from valenceStore_

    //...and calculate the valence-excess...
    current_excess = sqr(val[0]-6)+...+sqr(val[3]-6);

    // New valences under the assumption that
```

```
    // a flip has taken place
    val[0] -= 1; val[1] += 1;
    val[2] -= 1; val[3] += 1;
    flip_excess = sqr(val[0]-6)+...+sqr(val[3]-6);

    if( new_excess < current_excess ) return true;
    else return false;
  }
};
```

Eventually we can assemble the components. A sketch of the main parts of the program that balances the valence-excess of a triangle mesh is shown in Listing A.4.

Listing A.4: Reducing the valence excess

```
int main()
{
 Dynamic dynamic;
 //read mesh & pass it to dynamic

 Hub hub;

 ValenceStore vStore(mesh);
 hub.registerCallee(&vStore);

 BalanceStrategy balance(vStore);
 hub.registerCallee(&balance);

 dynamic.info = &hub;

 //now iterate...
 for( e = mesh.edges_begin(); e!= mesh.edges_end();++e)
   dynamic.flipedge(e);
}
```

A.5 Extensions

As capturing all details of our framework exceeds the scope of this chapter, we have only outlined the core concepts. However, there are many ways to extend the concept we have shown so far and we want to highlight some of them.

In our implementation we have used an *iterator* for processing multiple edges at once. So, instead of passing edges to `Dynamic::flipedge()` one-by-one, we iterate over a whole set of edges. We will illustrate the advantage of this concept by means of a small example.

Let's stick to the valence-excess example of the previous section. Assume we have a module A that wants to prevent the four blue edges (cf. Figure A.1) from flipping as soon as the red edge has flipped - the four edges remain locked as long as we have not processed every edge in the mesh. After one iteration over all edges, the status of these edges is set to "free" again. If a module B of our application processes edges one-by-one, we need an indicator when it is done with processing. The problem is that both modules might be unaware of each other. We can solve this problem by leaving the control over the edge-flips to `Dynamic` and inform the user via the `InfoBase`-interface after all the edges provided by the iterator are processed. This way all clients that have registered at the `Hub` can respond in their specific way. Of course, the iterator is not limited to iterating over all edges of a mesh , but it can feed an arbitrary set of edges to `Dynamic::flipedge()`.

Listings A.5 and A.6 show the extension we have to make in order to realize the concept.

Listing A.5: Incorporating the iterator-concept into Dynamic.

```
struct Dynamic<Mesh> {

  //canditate serves as iterator over a set of edge

  void flipedge(){
    for(candidate->init(); candidate->hasmore();
        candidate->next() ){
```

```
      if(info->preFlip(const Data& _d)){
        mesh.flipedge(candidate->get());
        info->postFlip(const Data& _d);
      }
    }//end: for all candidates

    info->endFlip();
  }

  Mesh&       mesh;
  InfoBase* info;
  Candidate* candidate;
};
```

Listing A.6: Extended version of InfoBase.

```
class InfoBase<Dynamic>{
 virtual bool preflip(const Dynamic::Data&){return true;}
 virtual void postFlip(const Dynamic::Data&){};
 virtual void endFlip(){}
};
```

Another venue for extending our framework is the design of small client classes that cover just one specific aspect of the changing connectivity. In this context we do not limit ourselves to the edge-flip, but think of the complete set of operators that change the connectivity of a mesh. The aspects can be as diverse as:

- a change of the adjacency list of a vertex.
- a notification that some triangles/edges/vertices have vanished.
- a change of normals in the vicinity of an operation.

For instance we could address the first item by designing a module that maintains a list of adjacent vertices for each vertex in a mesh. This modules registers at the `Hub` and updates its internal list with respect to a notification

it gets via the callback mechanism - this can be done efficiently, since the module "knows" how the adjacency lists have to be updated for e.g. an edge-flip. Now the module can grant a client access to these lists. The client does not need to worry if the lists are up-to-date or maintain its own list.

We can even go one step further and separate the data (the adjacency lists) from the information about the update (which list has changed in which way). Using this concept we can design a class *A* that registers at the `Hub` and just takes care of the update. Another class *B1*, which registers at *A*, can e.g. hold the adjacency lists. To carry on this thought, we also think of a class *B2* that just needs to be notified about a change of the adjacency-information, but does not hold a complete adjacency-list at all. Eventually, this concept will lead to a tree-like structure of callbacks that enable a client to register at those points that are vital for its algorithms.

A.6 Results and Conclusion

We have used the framework we have described in this chapter to implement *FSR*, a program that comprises the algorithms propose in [83, 84]. In *FSR* we register dozens of modules that inter-operate with each other, it showed that managing these modules without clearly separating between them is quite error-prone. We have also implemented a small example which is based on *OpenMesh*[8]. A tar-archive can be downloaded from our web-site [81]. Please note that the current version is just a feasibility study and is not mature enough to be used in a production environment.

Certainly, our framework comes along with some overhead compared to directly calling the member functions (edge-flip/edge-split, etc.) of the mesh. As a worst case scenario we have tested our implementation for *OpenMesh*[8]. We have passed an empty `InfoBase`-object to `Dynamic` in order to disable the callback-mechanism and have executed one single edge-flip via our iterator-interface. This setup is 2.5x slower that the direct call to `mesh.flip()`. However, after changing our *FSR*-program to the proposed concept, we not only found that it was easier to incorporate new algorithms, but we were also able to discarded many calls to redundant update routines and eventu-

ally make *FSR* significantly faster.

We have proposed a framework for efficiently handling and working with dynamic connectivity meshes from a design point of view. In particular we have shown how encapsulated modules that depend on the changing connectivity of a mesh can keep internal data-structures up-to-date. Our callback-mechanism facilitates implementing new algorithms which base on dynamic meshes and shows how to add this new functionality to complex applications that make use of our framework.

Of course, there are many venues for further extensions and improvements. We expect, that a rich pool of small and reusable clients, which cover one specific aspect of the changing connectivity, will significantly speed up the development of algorithms that depend on dynamic meshes.

Appendix B

Calculating Phong-Detail

This section gives some background on encoding a vertex in a local coordinate frame that is given by one triangle and a Phong-type normal field. The normal field is determined by three prescribed normals located at the triangle's vertices and gets blended in-between. We stumbled across this seemingly simple geometric problem when encoding high frequency geometric detail levels on top of a lower frequency domain (cf. 4.4.3). Assume we are given a point $\mathbf{p}$ in a global 3D coordinate system and a triangle $\mathbf{t}$ with vertices $\mathbf{a}, \mathbf{b}$ and $\mathbf{c}$ and corresponding vertex normals $N_{\mathbf{a}}, N_{\mathbf{b}}$ and $N_{\mathbf{c}}$.

As it is depicted in Figure B.1 we are looking for that point $\mathbf{q}$ inside $\mathbf{t}$ and a scalar offset h such that

$$\mathbf{p} = \mathbf{q} + hN_{\mathbf{q}} \tag{B.1}$$

where

$$\begin{aligned} \mathbf{q} &= \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} \\ N_{\mathbf{q}} &= \alpha N_{\mathbf{a}} + \beta N_{\mathbf{b}} + \gamma N_{\mathbf{c}} \\ 1 &= \alpha + \beta + \gamma \end{aligned}$$

Once we have α, β and h calculated, we can express $\mathbf{p}$ with respect to the given triangle and the normal-field. This way, whenever $\mathbf{t}$ is transformed, $\mathbf{p}$ changes it's position accordingly. The straightforward approach, i.e., plugging in the definitions for $\mathbf{q}$ and $N_{\mathbf{q}}$ leads to the equation:

$$\det\big(\begin{bmatrix} \mathbf{a}-\mathbf{p} & \mathbf{b}-\mathbf{p} & \mathbf{c}-\mathbf{p} \end{bmatrix} + \lambda \begin{bmatrix} N_{\mathbf{a}} & N_{\mathbf{b}} & N_{\mathbf{c}} \end{bmatrix}\big) = 0$$

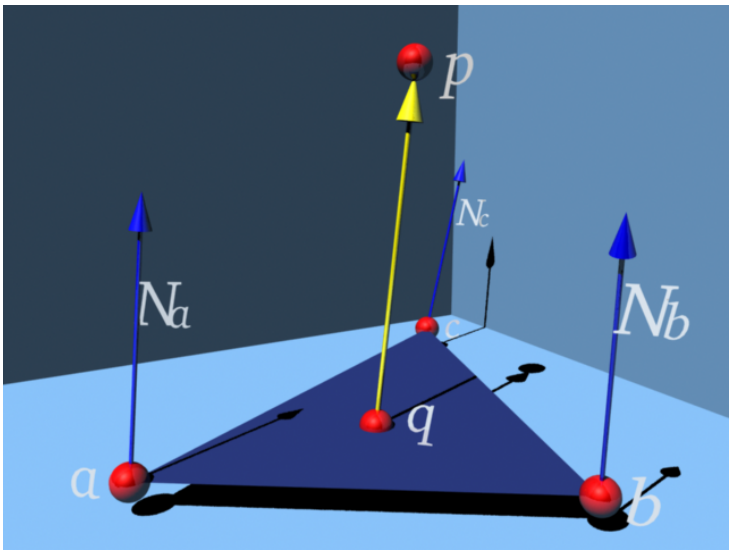

Figure B.1: The given vertex-normals N_i create a smooth Phong-type normal field. The objective is to calculate the point q in a way that the blended normal at q points in the direction of p.

This equation of third degree can be solved with Cardano's Formula [15]. In a second step α and β can be obtained as the solution of the remaining over determined linear system. Since our initial problem is a time critical operation in our interactive environment, we favor a faster approach that we want to discuss in more detail within the scope of this thesis.

Instead of calculating the whole detail vector in one single step, we first calculate the base point $\mathbf{q}$ as the solution of

$$(\mathbf{p} - \mathbf{q}) \times N_{\mathbf{q}} = 0$$

In a second step we can simply calculate the offset h by computing the distance between $\mathbf{p}$ and $\mathbf{q}$. Again, by plugging in the definition of $\mathbf{q}$ and $N_{\mathbf{q}}$ we obtain the bivariate quadratic function

$$\begin{array}{rcl} F : \mathbf{R}^2 & \rightarrow & \mathbf{R}^3 \\ (u,v) & \mapsto & (\mathbf{p} - u\mathbf{a} - v\mathbf{b} - (1-u-v)\,\mathbf{c}) \times (uN_{\mathbf{a}} + vN_{\mathbf{b}} + (1-u-v)\,N_{\mathbf{c}}) \end{array}$$

and we have to find a tuple (α, β) such that $F(\alpha, \beta) = (0,0,0)^T$. We numerically solve this equation with Newton's multidimensional method since it is

known to converge quickly. In the remainder, we determine the iteration rule for one step of Newton's method and present a reasonable starting value.

Newton's iteration rule in its general form for a mapping

$$F \,:\, \mathbf{R}^n \,\rightarrow\, \mathbf{R}^n \,,\, x \mapsto F(x)$$

is of the form:

$$x_{k+1} = x_k - J_F^{-1}(x_k)F(x_k) \;\;, k = 0, 1, \ldots$$

where J_F^{-1} denotes the inverse of the Jacobian of F. In our context it is more convenient to rewrite this formula in the following way:

$$J_F(x_k)\,(x_{k+1} - x_k) = -F(x_k)$$

Notice that F can be interpreted as a quadratic surface patch in $\mathbf{R}^3$, which passes through the origin, and that we can solve the system of equations in a least squares sense only, i.e., we solve for $dx := x_{k+1} - x_k$ in the following equation.

$$J_F^t J_F dx = -J_F^t F$$

Hence, in our context we get the difference $(\triangle\alpha, \triangle\beta)$ between two iteration steps as the solution of the linear system.

$$\begin{pmatrix} F_u^T F_u & F_u^T F_v \\ F_u^T F_v & F_v^T F_v \end{pmatrix} \begin{pmatrix} \triangle\alpha \\ \triangle\beta \end{pmatrix} = \begin{pmatrix} -F_u^T F \\ -F_v^T F \end{pmatrix} \tag{B.2}$$

where the partial derivatives F_u and F_v of F can explicitly be given by doing a Taylor expansion of F about the origin.

$$\begin{aligned} F &= (\mathbf{p} - \mathbf{c}) \times N_{\mathbf{c}} \\ F_u &= (2\mathbf{c} - \mathbf{a} - \mathbf{p}) \times N_{\mathbf{c}} + (\mathbf{p} - \mathbf{c}) \times N_{\mathbf{a}} \\ F_v &= (2\mathbf{c} - \mathbf{b} - \mathbf{p}) \times N_{\mathbf{c}} + (\mathbf{p} - \mathbf{c}) \times N_{\mathbf{b}} \end{aligned}$$

In our application scenario, i.e., encoding detail information with respect to a geometrically smooth base domain, we are typically not facing geometric degeneracies. For this reason we simply apply Cramer's rule to solve the

linear system. This way the update rules for two consecutive Newton steps can explicitly be given.

$$\begin{aligned} \triangle\alpha &= (F_u^T F_v \cdot F_v^T F - F_v^T F_v \cdot F_u^T F)/s \\ \triangle\beta &= (F_u^T F_v \cdot F_u^T F - F_u^T F_u \cdot F_v^T F)/s \end{aligned}$$

with $s = F_u^T F_u \cdot F_v^T F_v - (F_u F_v)^2$.

As a good starting value (α_0, β_0) we can simply project p into the plane that is spanned by $\mathbf{t}$. Since our application requires positive values for α and β only, we found that the barycenter $(\alpha_0, \beta_0) = (\frac{1}{3}, \frac{1}{3})$ is a sufficient starting value and we typically do not need more than 5-7 iterations until the error falls below some reasonable threshold.

As a side remark: In case one of the barycentric coordinates of the resulting point $\mathbf{q}$ is negative, we continue the search for a base point in the corresponding neighboring triangle of our domain mesh. Since the Phong normal field is globally continuous we always find a base point with positive barycentric coordinates.

List of Figures

1.1 Two different triangulations of a geometric object 4

2.1 Basic edge based topological operators on triangle meshes . . . 12
2.2 The 1-to-3-split of a triangle 13
2.3 Divide a 1–to–4-split into three edge-splits and one edge-flip . 14
2.4 A Semi-regular mesh at different resolutions 18

3.1 Shift vector: The three situations 33
3.2 Tweety's tail: Projection causes artifacts 34
3.3 Controlling the approximation error 35
3.4 The topological operators used in DCM 37
3.5 Remeshing of Tweety, a geometrically more complex model . . 41
3.6 Remeshing of the Elch model, a geometrically and topologically complex model 42
3.7 Aliasing artifacts at sharp features 43
3.8 1D sketch of filtered curvature field 48
3.9 Hierarchical curvature field on mesh with sharp features 49
3.10 Shift-vector restriction by kernel 50
3.11 Aligning edges to features by edge-flips 52
3.12 Feature snapping with restricted snap-directions 53
3.13 Feature reconstruction by edge-snapping 55
3.14 Skeleton on the fan-disk model 56

3.15 How to insert new bone-vertices 58
3.16 Skeleton preservation at different resolutions 60

4.1 Interactive Remeshing of Max Planck's ear 62
4.2 $\sqrt{3}$-remeshing of a tooth model 68
4.3 Freeform modification of a semi-regular mesh 70
4.4 Freeform modification of an unstructured mesh 71
4.5 Multiresolution modeling metaphor 73
4.6 Different scales of detail on Max Planck's head 75
4.7 Comparison of detail encoding with respect to different normal fields 79
4.8 Comparison of modeling results wrt. different normal fields . . 81
4.9 Single vs. multiple detail levels 82
4.10 3D comparison of single vs. multiple detail levels 82
4.11 Multiresolution Modeling shown on the bust's face. 87

A.1 Illustration of an edge-flip 95
A.2 UML-diagram of the dynamic remeshing framework 100

B.1 Detail encoding with respect to Phong-type normal field. . . . 110

Bibliography

[1] A. Alexandrescu. *Modern C++ Design: Generic Programming and Design Patterns Applied*. Addison-Wesley, 2001.

[2] Pierre Alliez, David Cohen-Steiner, Olivier Devillers, Bruno Lévy, and Mathieu Desbrun. Anisotropic polygonal remeshing. In *SIGGRAPH 2003 Conference Proceedings*, pages 485–493, 2003.

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

[4] Pierre Alliez, Mark Meyer, and Mathieu Desbrun. Interactive geometry remeshing. In *SIGGRAPH 2002 Conference Proceedings*, pages 347–354, 2002.

[5] Alan H. Barr, Mark Meyer, Mathieu Desbrun, and Peter Schröder. Discrete differential-geometry operators for triangulated 2-manifolds, 2001.

[6] Henning Biermann, Ioana Martin, Fausto Bernardini, and Denis Zorin. Cut-and-Paste editing of multiresolution surfaces. In *SIGGRAPH 2002 Conference Proceedings*, pages 312–321, 2002.

[7] M. Botsch and L. Kobbelt. Multiresolution surface representation based on displacement volumes. In *Computer Graphics Forum (Eurographics 2003)*, 2003.

[8] M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt. Openmesh - a generic and efficient polygon mesh data structure. In *OpenSGPlus Symposium*, 2002.

[9] P. J. C. Brown and C. T. Faigle. A robust efficient algorithm for point location in triangulations. Technical report, Cambridge University, 1996.

[10] William J. et al. Brown. *Antipatterns*. John Wiley and Sons, 2000.

[11] James Davis, Steven R. Marschner, Matt Garr, and Marc Levoy. Filling holes in complex surfaces using volumetric diffusion. In *Proceedings of the 1st International Symposium on 3D Data Processing Visualization and Transmission (3DPVT-02)*, pages 428–438, Los Alamitos, CA, 2002.

[12] T. DeRose, M. Lounsbery, and J. Warren. Multiresolution analysis for sufaces of arbitrary topological type. Technical Report 93–10–05, Department of Computer Science and Engineering, University of Washington, 1993.

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

[14] Mathieu Desbrun, Mark Meyer, and Pierre Alliez. Intrinsic parameterizations of surface meshes. In *Computer Graphics Forum (Eurographics 2002)*, pages 209–218, 2002.

[15] W. Dunham. *Journey Through Genius: The Great Theorems of Mathematics, Cardano and the Solution of the Cubic*. Wiley, 1990.

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

[17] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *SIGGRAPH 1995 Conference Proceedings*, pages 173–182, 1995.

[18] D. Zorin et. al. Subdivision for modeling and animation. In *SIGGRAPH 2000 Course Notes*, 2000.

[19] Gerald Farin. *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Guide*. Academic Press, New York, NY, USA, fourth edition, 1997.

[20] M. S. Floater and K. Hormann. Surface parameterization: a tutorial and survey. In *Advances in Multiresolution for Geometric Modelling*, Mathematics and Visualization, pages 157–186. Springer, 2005.

[21] Michael S. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3):231–250, 1997. ISSN 0167-8396.

[22] Foley, van Dam, Feiner, and Hughes. *Computer Graphics*. Addison Wesley, 1990.

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

[24] David R. Forsey and Richard H. Bartels. Hierarchical B-spline refinement. In *SIGGRAPH 1988 Conference Proceedings*, pages 205–212, 1988.

[25] Gamma, Helm, Johnson, and Vlissides. *Design Patterns*. Addison-Wesley, 1995.

[26] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH 1997 Conference Proceedings*, pages 209–216, 1997.

[27] Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe. Geometry images. *ACM Transactions on Graphics*, 21(3):355–361, July 2002.

[28] I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. In *SIGGRAPH 1999 Conference Proceedings*, pages 325–334, 1999.

[29] I. Guskov, K. Vidimce, W. Sweldens, and P. Schröder. Normal meshes. In *SIGGRAPH 2000 Conference Proceedings*, 2000.

[30] Igor Guskov and Zoë Wood. Topological noise removal. In *Proceedings of Graphics Interface 2001*, pages 19–26, 2001.

[31] W. Hackbusch. *Multi-Grid Methods and Applications*. Springer-Verlang, 1986.

[32] H. Hoppe. Progressive meshes. In *SIGGRAPH 1996 Conference Proceedings*, pages 99–108, 1996.

[33] K. Hormann. *Theory and Applications of Parameterizing Triangulations*. PhD thesis, Department of Computer Science, University of Erlangen, November 2001.

[34] K. Hormann and G. Greiner. MIPS: An efficient global parametrization method. In *Curve and Surface Design: Saint-Malo 1999*, pages 153–162. 2000.

[35] A. Hubeli and M. Gross. Multiresolution feature extraction from unstructured meshes. In *IEEE Visualization 2001 Conference Proceedings*, pages 287–294, 2001.

[36] Martin Isenburg and Stefan Gumhold. Out-of-core compression for gigantic polygon meshes. In *SIGGRAPH 2003 Conference Proceedings*, 2003.

[37] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. In *SIGGRAPH 2002 Conference Proceedings*, pages 339–346, 2002.

[38] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. In *Interational Journal of Computer Visision*, pages 321–331, 1987.

[39] L. Kettner. Using generic programming for designing a data structure for polyhedral surfaces. *CGTA: Computational Geometry: Theory and Applications*, 13, 1999.

[40] A. Khodakovsky, Nathan Litke, and P. Schröder. Globally smooth parameterizations with low distortion. In *SIGGRAPH 2003 Conference Proceedings*, 2003.

[41] A. Khodakovsky, P. Schröder, and W. Sweldens. Progressive geometry compression. In *SIGGRAPH 2000 Conference Proceedings*, 2000.

[42] L. Kobbelt. Discrete fairing. In *Proceedings of the Seventh IMA Conference on the Mathematics of Surfaces*, pages 101–131, 1996.

[43] L. Kobbelt. $\sqrt{3}$–subdivision. In *SIGGRAPH 2000 Conference Proceedings*, 2000.

[44] L. Kobbelt, T. Bareuther, and H.-P. Seidel. Multiresolution shape deformations for meshes with dynamic vertex connectivity. In *Computer Graphics Forum (Eurographics 2000)*, volume 19(3), pages 249–260, 2000.

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

[46] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive multi–resolution modeling on arbitrary meshes. In *SIGGRAPH 1998 Conference Proceedings*, pages 105–114, 1998.

[47] L. Kobbelt, J. Vorsatz, U. Labsik, and H.-P. Seidel. A shrink wrapping approach to remeshing polygonal surfaces. *Computer Graphics Forum (Eurographics 1999)*, 18(3):119–130, 1999.

[48] L. Kobbelt, J. Vorsatz, and H.-P. Seidel. Multiresolution hierarchies on unstructured triangle meshes. *Computational Geometry: Theory and Applications*, 14, 1999.

[49] Leif Kobbelt, Stephan Bischoff, Mario Botsch, Kolja Kähler, Christian Rössl, Robert Schneider, and Jens Vorsatz. Geometric modeling based on polygonal meshes. In *Tutorial Notes (Eurographics 2000)*, pages 1–47. European Association for Computer Graphics, Eurographics, August 2000.

[50] V. Krishnamurthy and M. Levoy. Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH 1996 Conference Proceedings*, pages 313–324, 1996.

[51] Ulf Labsik and Günther Greiner. Interpolatory sqrt(3)-subdivision. *Comput. Graph. Forum*, 19(3), 2000.

[52] Ulf Labsik, Kai Hormann, and Günther Greiner. Using most isometric parametrizations for remeshing polygonal surfaces. In *GMP*, pages 220–228, 2000.

[53] A. Lee, H. Moreton, and H. Hoppe. Displaced subdivision surfaces. In *SIGGRAPH 2000 Conference Proceedings*, 2000.

[54] A. W. F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin. MAPS: Multiresolution adaptive parameterization of surfaces. In *SIGGRAPH 1998 Conference Proceedings*, pages 95–104, 1998.

[55] S. Lee. Interactive multiresolution editing of arbitrary meshes. *Computer Graphics Forum (Eurographics 1999)*, 18(3):73–82, 1999.

[56] Y. Lee and S. Lee. Geometric snakes for triangular meshes. *Computer Graphics Forum (Eurographics 2002)*, pages 229–238, 2002.

[57] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least squares conformal maps for automatic texture atlas generation. In *SIGGRAPH 2002 Conference Proceedings*, pages 362–371, 2002.

[58] Peter Liepa. Filling holes in meshes. In *Eurographics Symposium on Geometry Processing*, pages 200–20, 2003.

[59] P. Lindstroem. Out-of-core construction and visualization of multiresolution surfaces. In *SIGGRAPH 2003 Conference Proceedings*, 2003.

[60] C. T. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, Department of Mathematics, 1987.

[61] M. Lounsbery, T. DeRose, and J. Warren. Multiresolution Analysis for Surfaces of Arbitrary Topological Type. *ACM Transactions on Graphics*, 16(1):34–73, January 1997.

[62] David Luebke and Carl Erikson. View-Dependent Simplification of Arbitrary Polygonal Environments. In *SIGGRAPH 1997 Conference Proceedings*, pages 199–208, 1997.

[63] H. P. Moreton and C. H. Séquin. Functional optimization for fair surface design. In *SIGGRAPH 1992 Conference Proceedings*, volume 26, pages 167–176, 1992.

[64] Hartmut Prautzsch, Wolfgang Boehm, and Marco Paluszny. *Bezier and B-spline techniques*. Springer, 2002.

[65] Jarek Rossignac. Simplification and Compression of 3D Scenes. In *Tutorial Notes (Eurographics 1997)*, 1997.

[66] Jarek Rossignac and Paul Borrel. Multi-Resolution 3D Approximations for Rendering Complex Scenes. In *Modeling in Computer Graphics*, pages 455–465. Springer, 1993.

[67] Christian Rössl, Leif Kobbelt, and Hans-Peter Seidel. Recovering structural information from triangulated surfaces. In *Mathematical Methods for Curves and Surfaces: Oslo 2000*, pages 423–432, 2000.

[68] P. Sander, Z. Wood, S. Gortler, J. Snyder, and H. Hoppe. Multi-chart geometry images. In *Eurographics Symposium on Geometry Processing*, pages 157–166, 2003.

[69] Pedro V. Sander, Steven J. Gortler, John Snyder, and Hugues Hoppe. Signal-specialized parametrization. In *Eurographics Rendering Workshop*. Eurographics Association, 2002.

[70] N. Sapidis. *Designing fair curves and surfaces: shape quality in geometric modeling and computer-aided design*. 1994.

[71] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. In *SIGGRAPH 1992 Conference Proceedings*, pages 65–70, 1992.

[72] Oren Sifri, Alla Sheffer, and Craig Gotsman. Geodesic-based surface remeshing. In *12th International Meshing Roundtable*, 2003.

[73] Olga Sorkine, Daniel Cohen-Or, Rony Goldenthal, and Dani Lischinski. Bounded-distortion piecewise mesh parameterization. In *IEEE Visualization 2002 Conference Proceedings*, pages 355–362, 2002.

[74] Edwin H. Spanier. *Algebraic Topology.* Tata McGraw-Hill, Bombay, 1966.

[75] Vitaly Surazhsky and Craig Gotsman. Explicit surface remeshing. In *Eurographics Symposium on Geometry Processing*, pages 020–030. Eurographics Association, 2003.

[76] G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral. In *Proc. International Conference on Computer Vision*, pages 902–907, 1995.

[77] G. Turk. Re–tiling polygonal surfaces. In *SIGGRAPH 1992 Conference Proceedings*, pages 55–64, 1992.

[78] Greg Turk and Marc Levoy. Zippered Polygon Meshes from Range Images. In *SIGGRAPH 1994 Conference Proceedings*, pages 311–318, 1994.

[79] T. Várady and P. Benko. Reverse engineering B-rep models from multiple point clouds. In *Proc. of Geometric Modeling and Processing 2000*, pages 3–12, 2000.

[80] Luiz Velho. Stellar subdivision grammers. In *Eurographics Symposium on Geometry Processing*, pages 201–212, 2003.

[81] J. Vorsatz. www.mpi-sb.mpg.de/ vorsatz.

[82] J. Vorsatz. Interaktives multi-resolution modellieren mit polygonalen netzen beliebiger topologie. Master's thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 1998.

[83] J. Vorsatz, Ch. Rössl, L. Kobbelt, and H.-P. Seidel. Feature sensitive remeshing. In *Computer Graphics Forum (Eurographics 2001)*, pages 393–401, 2001.

[84] J. Vorsatz, Ch. Rössl, and H.-P. Seidel. Dynamic remeshing and applications. In *ACM Symposium on Solid Modeling and Applications*, pages 167–175, 2003.

[85] J. Vorsatz and H.-P. Seidel. A framework for dynamic connectivity meshes. In *OpenSG Symposium*, 2003.

[86] D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In *SIGGRAPH 1997 Conference Proceedings*, pages 259–268, 1997.

Jens Vorsatz

Persönliche Angaben

Geburtsdatum/-ort	13.01.1971 in Schwerte
Staatsangehörigkeit	deutsch
Familienstand	ledig
Kontakt	Schinkelstr. 26, 80805 München +49-176-20068493 mail@jensvorsatz.de

Werdegang

12/2005 – heute	Unternehmensberater bei TNG Technology Consulting in Unterföhring
10/2003 – 7/2005	Unternehmensberater bei McKinsey & Company, Inc. in München
4/1999 – 9/2003	Promotionsstudium als wissenschaftlicher Mitarbeiter am Max-Planck-Institut für Informatik in Saarbrücken
8/1998 – 3/1999	Promotionsstudium als wissenschaftlicher Mitarbeiter am Lehrstuhl für Graphische-Datenverarbeitung der Friedrich-Alexander-Universität in Erlangen-Nürnberg (FAU)
10/1992 – 7/1998	Studium der Mathematik an der FAU
	Abschluß: Diplom-Mathematiker Univ. (Note: sehr-gut)
10/1990 – 9/1992	Wehrdienst in der Sportfördergruppe Sonthofen
5/1990	Abitur am Friedrich-Bährens-Gymnasium Schwerte (Note: 1,7)