Constructing Boundary and Interface Parametrizations for Finite Element Solvers

Diploma Thesis in Computer Science

presented by

Oliver Sander

born October 20, 1975, in Hanau

written at the

Zuse Institute Berlin (ZIB)
Scientific Visualization Department

presented at the

Institute of Computer Science
Freie Universität Berlin

Supervising Professor: Stefan Felsner
Second Supervisor: Hans-Christian Hege

End of Work: November 2001

Acknowledgments

I feel indebted to Prof. Dr. Dr. h.c. Peter Deuflhard and Hans-Christian Hege who made it possible for me to write my thesis in such an inspiring environment as is the Zuse Institute Berlin (ZIB). I am grateful to Dr. Stefan Felsner for supervising my thesis and for introducing me to the wide field of computational geometry.
I'd like to thank all employees at the department for scientific visualization of the ZIB, in particular Dr. Martin Seebaß for his help on hyperthermia treatment and model generation, and Dr. Detlev Stalling and Steffen Prohaska for their help with Amira and all kinds of related topics.
My special thanks go to Prof. Dr. Ralf Kornhuber and Dr. Rolf Krause for their private lessons on numerical mathematics and finite elements, and for their general support. Thanks also to Stefan Zachow for supplying the human skull model.
It goes without saying that I feel infinitely grateful to my family for all they did for me throughout my entire life.

Contents

1 Introduction

Modern physics abounds with problems that depend, in some way or another, on a position in space as well as on time. From quantum mechanics to general relativity, a great deal of questions can be formulated in terms of 'compute the value f at the position x'. The value f can, for example, be a probability density in quantum mechanics or the stress tensor in elasticity.

Problems of that kind are usually formulated as *partial differential equations* (*PDEs*). Those equations, in use for a considerable amount of time, still pose formidable difficulties to mathematicians. It is generally considered a success if the existence or the uniqueness of the solution of a particular equation can be proven, not to mention actually computing a solution[1].

With the advent of modern computer technology, those problems have become tractable. Disposing of machines capable of billions of floation point operations per second enables researchers to use algorithms for computing approximate solutions that were simply unthinkable at the time of hand-written computations.

However, solving partial differential equations is still far from being easy. The demand on numerical precision continues to rise faster than hardware speed. Numerical simulations of general relativity effects, for example, can still take weeks on today's most powerful machines [3]. Therefore, a lot of research goes into the development of *efficient algorithms* for the numerical solution of PDEs.

One of the tools used to tackle partial differential equations is the *finite element method*, or *FEM* for short. Having been developed in the middle of the 20th century, it has quickly become one of the most commonly used approaches, mainly because of its superior flexibility when compared to alternative methods such as *finite differences* [15].

Finite element programs depend on the decomposition of the computational domain into small *elements*. Few conditions are imposed on the structure of such a *grid* other than that the elements cover the entire domain. In two-dimensional problems, triangles and quadrilaterals are most frequently used, whereas in three-dimensional space tetrahedra and hexahedra are predominant. Note that the faces of the elements need not be aligned with the coordinate axes, making very irregular domains possible. Even mixing different element types poses no fundamental difficulty.

The quality of this space decomposition, or *mesh*, as it is also called, is crucial for several reasons. First of all, the numerical error of the solution depends directly on

[1] Proving the existence of solutions of the Navier-Stokes equation, a PDE describing the behaviour of a viscous, incompressible fluid, has recently been declared one of the seven *Millenium Problems* of modern mathematics.

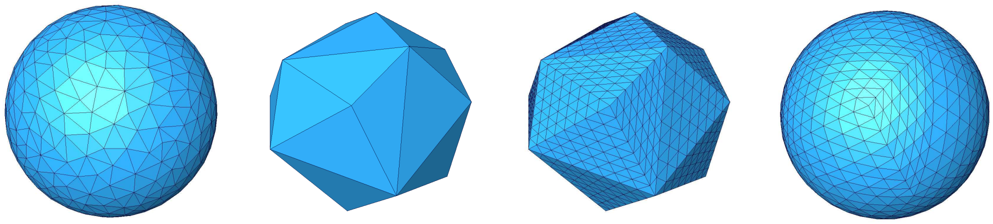

Figure 1.1: *Using parametrized surfaces increases the quality of local refinements. i) Original surface; ii) Simplified down to a twenty-vertex model; iii) Each triangle refined three times using a standard refinement scheme; iv) Same as iii), but taking into account a parametrization of the original surface.*

geometrical features of the mesh, such as the size of the elements and their interior angles. Secondly, the time needed for computing the solution is obviously a function of the mesh size. Therefore, one wants a mesh to be as coarse as possible, in order to ensure a fast computation, but on the other hand the mesh needs to be fine enough to yield a low-error result.

Modern day FE algorithms use *adaptive mesh refinement techniques* to overcome this problem. They do most part of the work on a rather coarse grid for increased speed, but this grid is refined wherever local error estimates show that it is necessary. This approach works particularly well with *elliptic* equations, where it can be shown that local errors always have a local cause [41].

These refinement techniques are particularly important in parts where elements with different properties meet. For example, when computing the mechanical behavior of a bridge, resistance to external forces is a property of the materials used, and, as bridges are usually built out of several materials, it is a function of position. The interfaces between elements with different properties can be described by polygonal surfaces and merit a special attention.

When meshes are refined using standard bisection techniques, their geometry doesn't change (see Fig. 1.1). This is unfortunate near material interfaces, since they are frequently known in far higher resolution than what is shown in the coarse grid. Therefore, *multiresolution representations* of such surfaces have been developed, which allow to work with a low-resolution surface but to retrieve detailed information wherever it's necessary.

One way of managing such surfaces with locally choosable resolution is to store the detailed surface as a function defined on a coarse version of the same surface. Such a function is called a *parametrization* of the surface. The coarse surface can be used for computation, and whenever increased accuracy is called for, the function on each face describes its true shape in the highest level of detail available.

In this thesis, we describe a general way of constructing such coarse surfaces along with the parametrization of the original surface. We pay particular attention to the quality of the parametrization and give several ways how to measure and improve it. In more detail, this thesis proceeds as follows.

In Chapter 2, we motivate our work. We present an exemplary problem from medical physics, namely *hyperthermia treatment planning*. We show that the physical problems arising can be modeled by partial differential equations, and we briefly demonstrate how those can be solved by means of finite elements. This leads to the introduction of *computational grids*. We describe the grid generation process and explain the demands that are put on those grids. Finally, we show that the *adaptive local refinement* of grids can lead to performance benefits without sacrificing numerical accuracy.

Chapter 3 deals with the task of automatic surface simplification. We formally introduce triangulated surfaces as *abstract simplicial complexes* together with an embedding into euclidean space. We describe an abstract framework for surface simplification algorithms. Within this framework, algorithms can use different *geometrical primitives* to reduce surface complexity. Secondly, they can consult an *oracle* to choose the order in which to apply the primitives. We present the *point-removal scheme*, the primitive we used in our implementation, and briefly discuss a few others. We then discuss several quality measures that can be included in the implementation of an oracle. Most of them are direct consequences of the quality demands put on computational grids in Chapter 2. We further explain how to deal with non-manifold surfaces. Finally, we show how to prevent topology changes and self-intersections when simplifying surfaces.

In Chapter 4, we introduce surface parametrizations. After formally defining the term for arbitrary surfaces, we present three algorithms that deal with the important case of parametrizing a topological disk over a compact subset of the plane. We pay particular attention to *piecewise linear* surfaces. We then describe our implementation of an algorithm that automatically constructs parametrizations of triangulated surfaces. It works by simplifying the surfaces to be parametrized and by propagating a valid parametrization at each step. We show how the *relative condition number* of a certain matrix can be used to measure the quality of a parametrization at each triangle. We finally present several ways to improve this quality.

In Chapter 5 we explain the most common way of refining tetrahedral grids. It is shown that the refinement of three-dimensional grids induces two-dimensional refinements of the boundary and interface surfaces. We show how the use of parametrized surfaces increases the geometric accuracy of such refined surfaces. We finally present *Loop-smoothing*, an advanced surface refinement scheme which is expected to lead to considerable improvements of the quality of the refined surface.

In Chapter 6 we use our algorithm in a real-life setting. We first use it to simplify a complicated surface obtained from the segmentation of computer tomography images of a hyperthermia patient. Repeated refinement of the simplified surface is shown to lead to an approximation of the original surface. We then take the model of a human skull, simplify it, and export it into the finite element package UG. There we numerically compute its behavior when being subjected to boundary forces using an adaptive nonlinear multigrid algorithm. The use of the parametrized surface leads to a considerable increase in geometric accuracy.

We conclude with Chapter 7. After recapitulating the essentials of this thesis, we mention two possible areas of further research. We finally point out the versatility of the parametrized surface data structures by briefly describing a few other applications they have.

2 Solving Problems in Physics

2.1 Hyperthermia Treatment Planning

Regional hyperthermia is a cancer treatment that is primarily applied to deep-seated tumors in the pelvic region. It consists of heating parts of the body to a *therapeutic temperature*, i.e., to around 43°C. Cancerous tissues that are heated become more susceptible to classical radiation- and chemotherapy. The heating is usually accomplished using *phased arrays* of radio-wave emitting antennae surrounding the patient. Single antennae or groups of antennae are fed by separate power generators. They emit sine waves of a fixed frequency in the radio wave range (around 100MHz) with controllable phase and amplitude. By proper selection of these parameters the interference patterns of the antenna fields can be steered in order to create a therapeutically favourable temperature distribution in the body [67].

The most common phased array system is the Sigma-60 applicator of BSD Medical Systems, Salt Lake City, Utah. It consists of a single ring of eight antennae surrounding the patient. The antennae are connected to form four pairs that are driven by one signal generator each. Clinical experiments have shown, however, that there are certain locations which this system cannot heat properly. Therefore, several studies have been made to evaluate the use of applicators consisting of more antennae on more than a single ring [62, 78]. They have shown that a 3D control of the interference pattern can be achieved if the antennae are arranged in more than one ring. Partly in view of this, the Sigma-Eye applicator was introduced by BSD Medical Systems in 1996. This applicator contains three rings of four antenna pairs each.

One of the keys to success in hyperthermia treatment is a proper *treatment planning*. It is virtually impossible for the physician to obtain acceptable results by manually steering the signal generators. The task of creating the interference patterns such that as much heat as possible is absorbed within the tumor and on the other hand avoiding to overheat healthy tissue is far too complex for the human mind. Also, the thermal effect of the external application of radio waves depends closely on the particular patient, since the human body displays a very heterogeneous dielectric behaviour. Thus, the use of computers to control the antennae is inavoidable.

Treatment planning begins with the construction of a geometric computer model of the patient. Using a computer tomography (CT) scan of the body region involved, a *tetrahedral grid* suitable for finite-element computations is generated (see Sec. 2.4 for a detailed description of this important mesh generation process). With a starting set of antenna parameters, this model is then used to compute first the electromagnetic field and then the temperature distribution within the patient's body. The information thus obtained can be used to optimize the antenna parameters using a variety of different *objective functions*.

2.2 Physics of Hyperthermia

The ultimate goal of hyperthermia treatment planning is the computation of the temperature distribution within the patient's body given the amplitudes and phases of the signals emitted by the antennae and the geometry of the patient together with the applicator. While we show how to acquire such geometries in Sec. 2.4, we will explain here briefly the physical problems that have to be solved.

The first step is the computation of the electromagnetic field created by the antennae in the patient's body. It is governed by the well-known *Maxwell equations* [39]

$$\operatorname{div} B = 0, \qquad \operatorname{curl} E + \frac{1}{c}\frac{\partial B}{\partial t} = 0, \tag{2.1}$$

$$\operatorname{div} D = 4\pi\rho, \qquad \operatorname{curl} H - \frac{1}{c}\frac{\partial D}{\partial t} = \frac{4\pi}{c} J. \tag{2.2}$$

Here, E and H are the electric and magnetic fields, respectively, D denotes the electric *displacement field* and B the *magnetic induction*. The scalar function $\rho = \rho(x)$ is the charge density, J the current density and c the speed of light.

These equations are coupled in linear, isotropic media by the relations

$$D = \epsilon E \quad \text{and} \quad B = \mu H,$$

where ϵ and μ denote dielectricity and magnetic permeability, respectively. Both ϵ and μ denote properties of materials and are therefore functions of position.

Due to the finite conductivity of the tissues involved, energy is deposited in the tissue by the electric field E. We use the energy

$$Q_e = \frac{1}{2}\sigma |E|^2$$

to define the *specific absorption rate (SAR)*

$$\mathrm{SAR} = \frac{Q_e}{\rho} = \frac{\sigma}{2\rho}|E|^2,$$

with σ the conductivity [Sm^{-1}] and ρ the density [kgm^{-3}] of the material under consideration (see [74]).

The SAR is already a good indicator of the effect the irradiation has on the patient. However, what's truly interesting is the distribution of temperature. This turns out to be rather difficult to model mathematically. The problem is that within the patient's body, heat is not only transported via the regular thermal conductivity. *Blood perfusion* has a major impact since blood flow is always trying to even out temperature differences in the human body and is thus counteracting any exterior thermal influence. The basic model commonly used is the instationary bio-heat-transfer equation for the temperature T proposed by Pennes [63]

$$\rho c \frac{\partial T}{\partial t} = \operatorname{div}(\kappa \operatorname{grad} T) - c_b W(T - T_b) + Q_e,$$

where ρ is the tissue density, c and c_b are the specific heats of tissue and blood, κ is the thermal conductivity; T_b is the blood temperature; W is the mass flow rate of blood per unit volume of tissue. The standard linear version of this equation assumes W to be a material constant. However, clinical experiments have shown that blood flood is strongly temperature dependent [69]. For example, blood flow *increases* with rising temperature in healthy tissue, whereas it *decreases* in the tumor. Consequently, Lang *et al.* [50] have modeled the temperature distribution using a *nonlinear* version of the above equation, where W_{tissue} is replaced by phenomenological functions $W_{\text{tissue}}(T)$ reflecting the desired properties.

2.3 Finite Element Methods

The physical problems described above are modeled mathematically using instances of a large family of equations called *partial differential equations (PDEs).* They occur in many areas of physics, and their analytical treatment is usually very difficult. Therefore, most PDEs have to be treated numerically.

Several general strategies exist for this. The *finite difference methods* consist of replacing derivatives with finite differences, giving rise to an approximation of the solution on a fixed regular grid. These methods are rather straightforward to implement. In hyperthermia planning, they are used with some astute extensions by, e.g., van de Kamer [74, 60]. However, since hyperthermia planning deals with very complex geometries, being limited to a regular grid is a severe disadvantage. *Finite element methods* remedy this problem.

Finite element methods, or FEM for short, are mainly in use for a subclass of the PDEs called the *elliptic* ones. Elliptic partial differential equations for a function $u : \mathbb{R}^d \supset \Omega \to \mathbb{R}^l$ on a domain Ω can be written as

$$Lu = f \quad \text{in } \Omega \tag{2.3}$$

with suitable boundary conditions prescribed on $\partial\Omega$. L describes a differential operator in *divergence form.* If L is *of second order*, it can be written as

$$Lu := -\sum_{i,k=1}^{n} \partial_i(a_{ik}\partial_k u) + a_0 u.$$

FE methods now use the fact that solving this equation is equivalent to solving the minimization problem

$$u \in H : \quad J(u) \leq J(v) \quad \forall v \in H$$

for the functional

$$J(v) := \frac{1}{2}a(v,v) - \langle f, v\rangle$$

in a suitable solution space H. Here, $a(\cdot,\cdot) : H \times H \to \mathbb{R}$ denotes a symmetric positive bilinear form and $\langle f, \cdot\rangle : H \to \mathbb{R}$ a linear functional [15]. As solution spaces H, one chooses *Sobolev-spaces*, which are subspaces of L_2, the space of square-integrable functions. Sobolev-spaces are large enough to ensure the existence of a solution. In particular, electrodynamic fields need not be continuous. Therefore, H has to contain functions which are not in $\mathcal{C}(\Omega)$. The set of PDEs that comply with (2.3) includes all equations that are of interest in the context of hyperthermia treatment.

Minimizing the functional J is equivalent to determining u such that

$$a(u,v) = \langle f, v\rangle \quad \forall v \in H. \tag{2.4}$$

However, solving this in the space of nondenumerable dimension H isn't any easier than solving the original equation. Therefore, instead of minimizing J in H, we content ourselves with finding the minimum in a suitable, finite-dimensional subspace. This subspace is taken from the class of *finite-element subspaces* $S^p_{\mathcal{T}}$. Let $\mathcal{T} = \{T_i\}$ be a decomposition of Ω into tetrahedral or hexagonal parts, then $S^p_{\mathcal{T}}$ is defined as

$$S^p_{\mathcal{T}} = \{v \mid \quad v|_{T_i} \in \mathcal{P}_p \quad \forall T_i \in \mathcal{T}\}, \tag{2.5}$$

$\mathcal{P}_p$ being the space of all polynomials of degree not exceeding p. Equation (2.4) is therewith changed to its finite-dimensional form

$$a(u_{\mathcal{T}}, v) = \langle f, v\rangle \quad \forall v \in S^{k,p}_{\mathcal{T}}.$$

Choosing a basis $\{\psi_i\}$ of $S^p_{\mathcal{T}}$, this is nothing more than the linear system

$$Ax = b, \tag{2.6}$$

with $A_{ij} = a(\psi_j, \psi_i)$ and $b_i = \langle f, \psi_i\rangle$.

It is obvious from (2.5) that if a function $v \in S^p_{\mathcal{T}}$ has discontinuities, those will only occur on the boundary between different T_i. Thus, if discontinuities in the true solution are expected, as is the case in the simulation of electromagnetic fields across material interfaces, those interfaces have to appear in the grid $\mathcal{T}$.

A short note on computational complexity should be adequate here. The largest part of the computer time needed to solve partial differential equations by the means of the finite element method is spent assembling the *stiffness matrix* A and solving the resulting system (2.6). By choosing basis functions ψ_i with compact support, we can have A be a *sparse* matrix. For that reason, the *nodal basis* is frequently used. Letting n denote the number of degrees of freedom of our discretizations, i.e., the number of rows and columns of A, we can solve (2.6) in $O(nd^2)$ time using appropriate sparse solvers. Here, d denotes the *bandwidth* of the matrix which, in our case, is of the order of $\sqrt{n}$. Modern multigrid algorithms can even solve systems of linear equations in optimal $O(n)$ time. A lot of research is currently spent on these interesting methods (see, e.g., [15]).

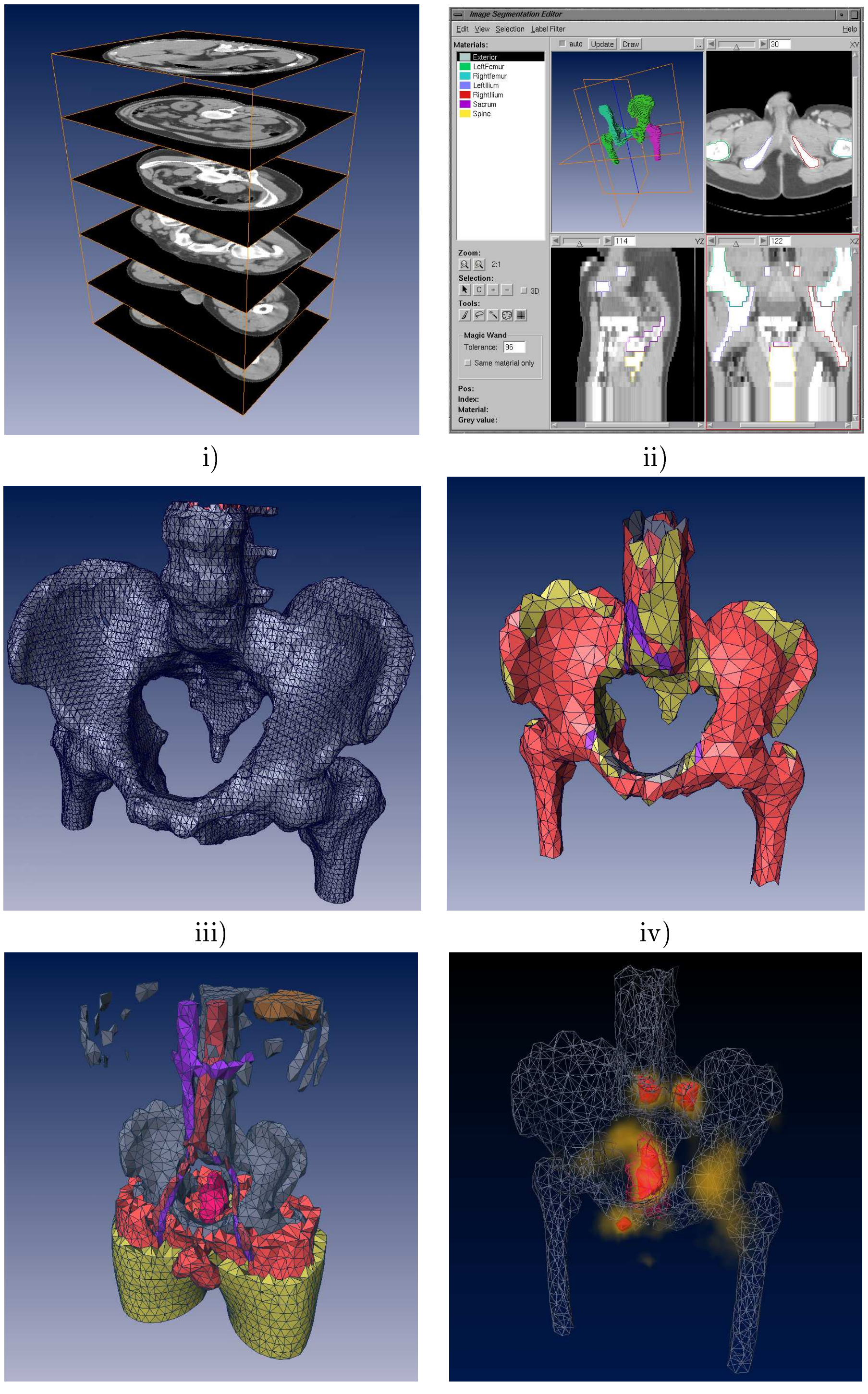

Figure 2.1: *The different steps of mesh generation: i) Slice images from a computer tomograph; ii) A snapshot from the* AMIRA *segmentation editor (GI); iii) Part of the output of the* generalized marching cubes (GMC) *algorithm, notice the high resolution; iv) Simplified surface, the different colors encode the neighboring materials; v) An insight into the volumetric model; vi) A numerically obtained temperature distribution.*

2.4 Generation of a Patient Model

The generation of individual patient models is the part requiring the most work in hyperthermia treatment planning. The reason is that one important substep, namely *image segmentation*, has as of yet not been fully automated. Even though many semi-automatic tools exist, the bulk of the work has still to be done manually.

We describe the model building process here the way it is done in the planning system HYPERPLAN [70], which is based on the general scientific visualization software package AMIRA [5]. HYPERPLAN uses tetrahedral meshes exclusively and no true multiresolution techniques are used so far. The model generation consists of five steps.

Image Acquisition

The first step is a CT-scan of the patient in question. Computer tomography (CT) is an advanced X-ray technique that allows to record axial slice images of a human body. These images are stored in the computer as graylevel bitmaps using sixteen bits per pixel. The whole set of images gives a stack of slices which represents a 3D volume. The image resolution can be quite high in each slice, up to 512×512 at the moment. Compared to that, the resolution on the third axis is rather low, on the order of less than a hundred slices, since taking more slices than absolutely necessary means an extra radiation dose of potentially harmful X-rays.

Image Segmentation

After their acquisition, the CT images are *segmented*. Segmentation is the process of labeling each voxel with the appropriate material, such as bone or muscle tissue. Depending on the quality of the images and the complexity of the scene, the segmentation problem ranges from fairly easy to nearly impossible. For this reason, no fully automated segmentation algorithms for complete CT images are available yet. Since segmentation has far more applications than just hyperthermia treatment, a lot of research is put into it.

HYPERPLAN contains many segmentation tools such as standard drawing utilities but also complex algorithms like *active contours* [77] and *region growing algorithms* [51]. Using those tools, a skilled person needs about four hours to segment a set of fourty slice images. Fig. 2.1 shows a view of AMIRA's segmentation editor GI.

Surface Extraction

The set of labeled slice images is used to create a triangulated surface model of the patient's geometry. This is done using an improved version of the well-known *marching cubes* algorithm [57] for surface extraction. This *generalized marching cubes (GMC)* algorithm [38] is able to handle non-binary classification, i.e., the volume can contain an arbitrary number of materials. The result is a surface that models all boundaries between different materials in the input data. The surface consists of *patches*, each patch being the set of triangles that divide a compartment of material A from a compartment of material B. Due to the high resolution of the CT-images, the extracted surface contains a very high number of triangles, generally of the order of several hundred thousands.

Simplification

The surfaces resulting from the last step are far too detailed to be used in mesh generation. Creating a volumetric model from a pure GMC surface would take a very long time and would result in a grid containing several millions of tetrahedra. At the time being, models of that size cannot be handled in a reasonable amount of time on a desktop workstation. We have to renounce some of the detail given and work with a simplified surface.

For this purpose, HyperPlan incorporates a *surface simplifier*. It is based on the approach of Garland and Heckbert ([33], see also Chapter 3), but contains some special features such as local error and edge-length control. The GMC surfaces are usually reduced by a factor ten in this application.

Building a Volume Model

In order to do finite-element computations on the model, it has to be expanded to a true volumetric representation. For that, a great number of additional vertices are added inside the boundary surface. Using these vertices and the known boundaries, the domain is then completely covered by nonintersecting tetrahedra. These tetrahedra form the partition of space necessary to define finite elements.

The high number of degrees of freedom (vertex positions and type of tetrahedrization) make grid generation a very difficult problem (see e.g., [34, 54]). Amira uses an advancing front approach based on work by Löhner and Parikh [55].

2.5 The Role of Mesh Quality

It is a well-known fact that the accuracy of the computed solution depends crucially on the 'quality' of the grid used in the finite element simulation. However, that notion of grid quality has always remained somewhat vague. In the literature, only very few results can be found that directly show the influence of the geometric properties of the grid on the numerical error.

Many introductory texts content themselves with saying that a valid triangulation of the computation domain should be *uniform*, or at least *quasi-uniform* [15]. The stronger condition of *uniformity* is fullfilled if there is a number $\kappa > 0$ such that every element T of $\mathcal{T}_h$ contains a circle with radius $\rho_T \geq h/\kappa$. These two conditions suffice to prove the convergence of the solution for $h \to 0$. The book by Hackbusch [37] is a bit more specific. The error bounds given there contain constants that depend on the smallest interior angle α_0 of the triangles. In order for the bounds to be reasonable, it is required that α_0 is bounded from below. This last condition is known as the *minimum angle condition.* It is widely used and has been elaborated upon in many papers even though it has been shown by Babuška and Aziz [6] that what is really essential is not the *minimum*, but the weaker *maximum angle condition*, which states that the largest angle in the triangulation should be bounded from above.

Some very detailed results on the influence of angles on the error bounds obtained are given by Acosta and Durán [1]. Working on the Stokes equation in two dimensions using the Raviart-Thomás (RT) element, they first prove an error bound for the normal direction components of the RT interpolation $\|(u - RTu) \cdot \nu_i\|_{0,T}$ that is independent of the shape of the triangle T. They then proceed to prove the following

Lemma 2.1 *Let T be a triangle with maximum angle θ_0. Let $H^1(T)$ be the first order Sobolev-space on T with the norm $\|\cdot\|_{0,T}$. Then there exists a constant C independent of T such that, for any $u \in H^1(T)$,*

$$\|u - RTu\|_{0,T} \leq \frac{C}{\sin\theta_0} \sum_{i=1}^{2} \|(u - RTu) \cdot \nu_i\|_{0,T},$$

where ν_1 and ν_2 are the normals to the edges sharing the vertex corresponding to the maximum angle.

It is immediately visible that as θ_0 tends to π, the leading fraction on the right side of the inequation blows up and the error bound becomes less and less strong.

A recent development has also been to define mesh quality not only in terms of geometry but also considering the behaviour of the expected solution (see [10] for a survey).

2.6 Criteria for 'good' Meshes

From the rather abstract discussion on grid quality above we derive some more practical consequences for tetrahedral grid generation, and from there for triangulated interface surfaces. We'll briefly discuss them here.

Geometric Approximation Error

The first quality characteristic for a mesh is the *geometric approximation error*. This simply means that the tetrahedral model should render the actual geometry as faithfully as possible within the limits of the piecewise linear approximation. It is a requirement on the segmentation, where it states that all voxels be labeled correctly, but it's much more important in surface simplification. Good simplification algorithms pay special attention to the geometric deviation of the simplified model to the original surface. Different approaches have been developed for measuring the geometric approximation error, such as *error quadrics* and different forms of the set-theoretic *Hausdorff distance*. They will be covered in more detail in Sec. 3.3.1 and 3.3.2.

Angles

As could be seen in the discussion in Sec. 2.5, most requirements on the cell quality are stated in terms of angles. The prototype is the *maximum angle condition* [6] which states that no angle should be too close to 180°. While Babuška and Aziz proved this in the planar setting, the generalization to three dimensions leads to a variety of different angle types that need to be considered. Interesting are, e.g., the interior angles of the triangular faces of the tetrahedra, and the *dihedral angles*, i.e., the angles formed by adjacent triangular faces. Considering the former immediately yields a criterion for interface surfaces, namely the original 2D maximum angle condition.

Element Aspect Ratio

With their proliferation in tetrahedral grids, angles become somewhat difficult to deal with. Many people therefore opt to use the *tetrahedral aspect ratio* as a simplified measure of element quality. It is defined as the ratio

$$\rho = \frac{R}{r}$$

between the radii of circumsphere R and insphere r, respectively, and captures the intuitive notion of 'well-formedness' rather nicely. The tetrahedral aspect ratio, too, implies a condition for boundary surfaces, requiring that the *triangular aspect ratio* remain bounded from above, a criterion that we will incorporate into our surface simplification algorithm later on.

2.7 Local Mesh Refinement

Partial differential equations are usually classified into several groups. One of them, namely the group of *elliptic equations*, to which belong among others the static heat equation and the diffusion equation, have the desirable property that *local errors have local causes.* This is what makes local adaptive mesh refinement works.

For the prototype of all elliptic equations, called *Poisson's equation*,

$$-\Delta u = f,$$

given with suitable boundary conditions, we have the following

Theorem 2.1 (Representation Theorem) *Let $u \in C^2(\overline{\Omega})$ be a classical solution of $-\Delta u = f$ on the open, bounded set Ω with known boundary conditions. Then for every point $\xi \in \Omega$*

$$u(\xi) = \int_\Omega K(x,\xi)\,\Delta u\,dx - \int_{\partial\Omega} \left(K(x,\xi)\frac{\partial u(x)}{\partial n} - u(x)\frac{\partial K(x,\xi)}{\partial n} \right) dS$$

holds.

Here $\partial/\partial n$ denotes differentiation in the direction of the *exterior* normal to $\partial\Omega$. The theorem shows the influence of the right hand side f and of the boundary conditions on the solution in a given point $\xi \in \Omega$. This influence is transported via 'weight functions'

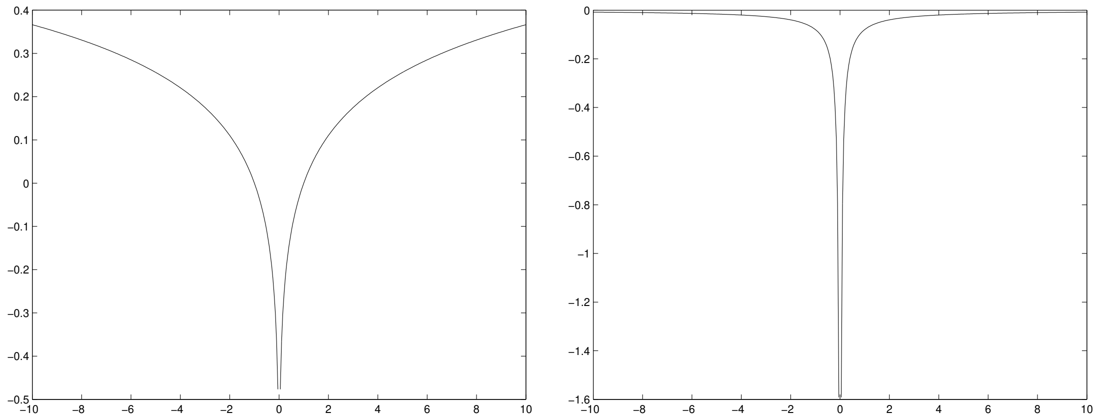

Figure 2.2: *The Green's functions $K(x, \xi)$ in two and three dimensions for $\xi = 0$.*

$K(x, \xi)$. These functions are called *Green's functions*. In n-dimensional space, they are defined for $x \neq \xi$ as

$$K(x,\xi) = \begin{cases} \dfrac{|x-\xi|^{2-n}}{(2-n)\omega_n}, & \text{for} \quad n \neq 2, \\[2ex] \dfrac{\log|x-\xi|}{2\pi}, & \text{for} \quad n = 2, \end{cases}$$

with ω_n the surface area of the unit sphere in $\mathbb{R}^n$.

Technically, Green's functions have infinite support. However, it is clear from looking at Fig. 2.2, that they have values close to zero everywhere except in a small neighborhood around ξ. Thus, handwavingly, one can say that the value of the solution u at the point ξ really only depends on a part of the the right hand side f *close to* ξ, and therefore, if an unacceptably large error is detected during the computation of $u(\xi)$, the cause for that error must have propagated with the Green's function as well and must have causes near ξ. It is therefore sufficient to repeat the computation on a grid which has a higher resolution only in the neighborhood of the error, considerable economizing on computer time. This does *not* necessarily work for non-elliptic PDEs where errors can and do propagate across the whole domain.

3 Simplification of Triangulated Surfaces

The quality of the simplified surface is of high importance for numerical applications and multiresolution representations. There are two reasons for this. First of all, the biggest part of the computation will most likely be performed directly on the simplified grid. In view of mesh quality, there's not much use in aggressively simplifying a surface only to refine it again globally before any calculations are done. That would just add one more processing step that makes triangle quality yet a little harder to control. On the contrary, one would want to keep as much of the base grid as possible, doing local refinements only where the approximation error is high. Therefore, any simplification algorithm that is intended to be used with an FE-solver has to pay the utmost attention to all the special criteria of surface quality set forth in Sec. 2.6. In particular, quality criteria that measure the suitability for high-quality grid generation, such as the triangle aspect ratio, may be more important than geometric approximation. Just because a model is geometrically accurate doesn't necessarily mean that it is suitable for a low-error computation. This makes many common grid simplification tools inapplicable for numerical computing.

Secondly, the quality of remeshed triangles depends directly on the quality of their base-grid 'ancestors'. In Chapter 4, we will describe how to construct surface parametrizations, i.e., continuous functions from a surface S into $\mathbb{R}^3$, that are as distortion-free as possible. We will define exactly what we mean by *distortion* in Sec. 4.4. Assuming such a parametrization, by subdividing a base-grid triangle, we get four new triangles that are almost congruent to their father (see Fig. 3.1). Therefore, a base-grid triangle with a poor aspect ratio will most likely yield four sons of similarly poor quality.

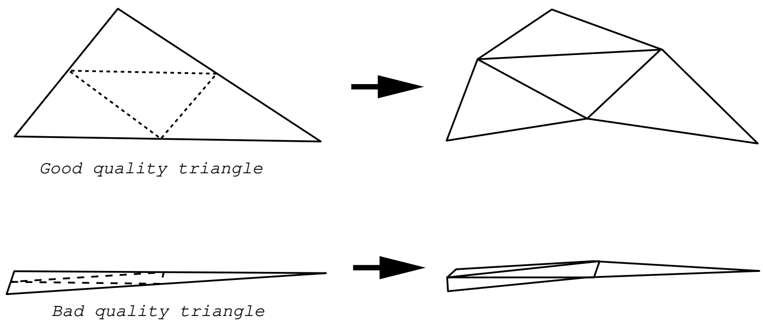

Figure 3.1: *When using a parametrization with little distortion, refined triangles have a quality similar to their ancestors'.*

Kobbelt [47] has described an abstract framework for surface simplification algorithms that we will present here in some detail. He sees simplification as the repeated application of a geometric primitive that performs a local, precise reduction step on the number of surface elements. Examples of such primitives are the contraction of an edge, or the removal of a single point with subsequent retriangulation of the resulting hole. Those and a few others will be described in further detail in Section 3.2.

The elements that the geometric primitive is based on (i.e., vertices, edges, or possibly triangles) are weighed using an appropriately chosen cost-function. This function measures how much the removal of a corresponding geometrical primitive would lessen the quality of the surface. After each application of a geometric primitive, the cost function is updated for those elements that were affected by the change. Except for pathological cases, those are only a very small number. The elements are kept in a priority queue that always gives immediate access to the element with the lowest error.

Kobbelt's description subsumes a lot of the simplification algorithms used today. Many follow the framework described above and differ only in the choice of reduction primitive and cost function. Kobbelt's own approach, for example, presented in [47], uses half-edge contractions as the primitive and a Hausdorff-like distance as the error measure. Garland and Heckbert [33] prefered pair-collapses ranked by the quadric error estimator described in Section 3.3.2. As a third example, the simplification algorithm used in the MAPS algorithm in [52] reduces the surfaces by removing single points and retriangulating the resulting holes. The points are weighed using a convex combination of local curvature and the area of the triangles surrounding each point. As an additional feature, the vertices are grouped into maximally independent sets, which assures that the removed vertices are spread out evenly across the surface.

This chapter will proceed as follows: In Sec. 3.1 we will start by formally defining the combinatorial and geometric objects we will be dealing with. Sections 3.2 and 3.3 explain in more detail the two constituent parts of Kobbelt's algorithm, namely the geometric primitives and different ways of estimating the 'cost' of surface elements. We will pay particular attention to the *point removal primitive*, since this is the one we used in our own implementation. Section 3.4 describes how that primitive can be extended to handle non-manifold surfaces. In Sec. 3.5 we give a simple rule which makes sure that a point removal step doesn't change the topology of a surface. We prove the correctness of this rule for manifold surfaces. Finally, Section 3.6 presents the techniques we used to make sure that the simplified surface doesn't self-intersect.

3.1 Surfaces and Graphs

In this chapter, we describe in detail the simplification process of triangulated surfaces. We will therefore start by giving a precise definition of such surfaces. Also, we'll briefly introduce a few basic concepts from graph theory, in order to have them at hand when needed. Since surfaces in computer graphics and numerical modeling are represented by piecewise linear objects, graph theory is the natural language for many problems that arise in these fields.

Definition 3.1 *An* **abstract simplicial complex** $\mathcal{K}$ *is an ordered pair of disjoint, finite sets* (V, F)*, such that* F *is a set of subsets of* V*. For the set* F*, the following condition must hold: If* $\alpha \in F$ *and* $\beta \subseteq \alpha$*, then* $\beta \in F$*. We call a set* $\alpha \in F$ *a* **simplex** *of dimension* $\dim \alpha = \operatorname{card} \alpha - 1$*. The dimension of the abstract simplicial complex is defined as the dimension of its highest-dimensional simplex*

$$\dim \mathcal{K} = \max_{\alpha \in F} \dim \alpha.$$

Intuitively, the set V is a set of *vertices*, and F a set of simplices on the vertices V. F is called the set of *faces*. The condition then asserts that for each simplex α in F, its lower-dimensional faces are also part of the complex.

A triangulated surface is a special kind of an abstract simplicial complex, together with a geometric structure.

Definition 3.2 *A* **triangulated surface** $\mathcal{S} = (V, F)$ *is an abstract simplicial complex of dimension two such that*

$$\forall \alpha \in F, \quad \operatorname{card} \alpha < 3 \quad \exists \beta \in F \quad \textit{with} \quad \alpha \subsetneq \beta$$

together with a function

$$\varphi : V \to \mathbb{R}^3.$$

An alternative way of describing a triangulated surface is as a triple of sets $\mathcal{K} = (V, E, F)$, where V is a set of vertex positions in $\mathbb{R}^3$, a set of *one-faces* E, and a set of *two-faces* F. We call the one-faces *edges*, the two-faces *triangles* and require that every edge should be the subface of at least one triangle and that every vertex should be incident to at least one edge. Sometimes, we'll denote edges $e = \{v_i, v_j\} \in E$ by $\overline{v_i v_j}$.

The above definition neatly separates the combinatorial and geometric aspects of triangulated surfaces. All information about adjacency is contained in the abstract simplicial complex, which in turn does not depend on surrounding space. If we consider the union of the convex hulls

$$\bigcup_{\alpha \in F} \operatorname{conv} \alpha$$

of all faces of F, we arrive at a surface in the classical sense, which we call the *straight-line geometric realization.* Throughout this thesis, we will sometimes use the term 'triangulated surface' when in fact the straight line geometric realization of a triangulated surface is meant. The exact meaning will be clear from the context.

Definition 3.3 *Given an abstract simplicial complex* $K = (V, F)$*, the* **star** $\operatorname{St} v$ *of a vertex* $v \in V$ *is the set of all faces* $f \in F$ *that contain* v*. The* **link** $\operatorname{Lk} v$ *of* v *is the set of all subfaces of the star of* v *that do* not *contain* v *[28].*

Thus, the term *star* formalizes the notion of the open neighborhood of a vertex v. Equivalently, the *link* of a vertex can be seen as the boundary of that neighborhood (Fig. 3.2). Next, we define what is meant by a *path* on a surface.

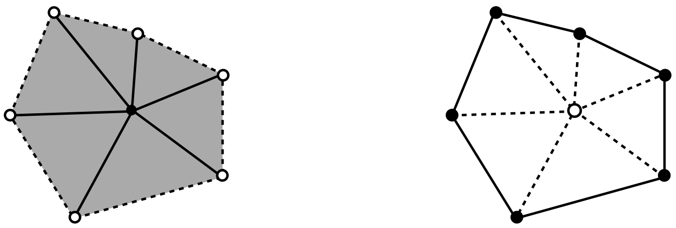

Figure 3.2: *The star and the link of a vertex.*

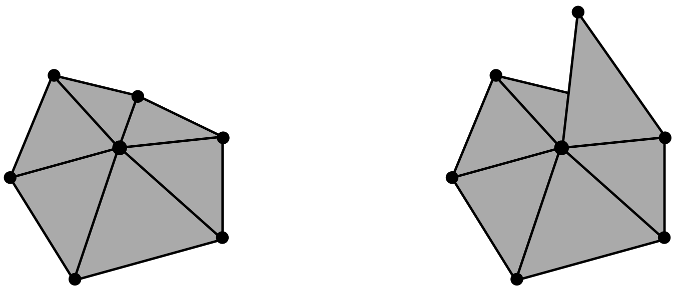

Figure 3.3: *The link of the center vertex on the left is a simple cycle. The vertex is therefore a manifold vertex. The center vertex on the right is a non-manifold vertex.*

Definition 3.4 *A* **path** *P on a surface G is an alternating sequence of vertices and edges, say $x_0, e_1, x_1, e_2, \ldots, x_l$, where $e_i = \overline{x_{i-1}x_i}$. The* length *of P is l, the number of edges in the path. If $x_i \neq x_j \quad \forall i \neq j$ we call P a* **simple** *path. If $x_0 = x_l$, we call the path a* cycle*, and if in addition all its edges and all vertices $x_1, \ldots, x_{l-1}$ are distinct, we call it a* **simple** *cycle.*

Now we have all concepts needed to introduce *triangulated manifold surfaces.*

Definition 3.5 *A vertex v is called a* **manifold vertex***, if its link forms a simple cycle. The vertex v is called a* **boundary vertex***, if its link forms a simple path. In all other cases, v is called a* **non-manifold vertex***.*

If a triangulated surface $\mathcal{S}$ contains only manifold vertices, it is called a **manifold surface***, or simply a* **manifold***. If it contains boundary vertices as well as manifold vertices, it is called a* **manifold with boundary***. In all other cases, it is called a* **non-manifold***.*

We now turn to graph theory. As it turns out, a graph is simply a one-dimensional abstract simplicial complex.

Definition 3.6 *A* **graph** *G is an ordered pair of disjoint, finite sets (V, E), such that E is a subset of the set of unordered pairs of V.*

The set V is called the set of *vertices* and E is the set of *edges*. We will sometimes also use the term *node* to refer to an element of V. An edge $\{x, y\}$ is said to *join* the vertices

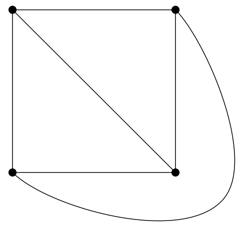

Figure 3.4: K_4, *the complete graph on four vertices.*

x and y and is denoted by $\overline{xy}$. If $\overline{xy} \in E$ then x and y are *adjacent* or *neighboring* vertices of G, and $\overline{xy}$ is *incident* with x and y. The number of edges incident with a vertex x is called the *degree* of x an is denoted d_x.

Despite their definition in terms of sets, graphs are usually interpreted in a geometric way. Thus we do not think of a graph as an ordered pair but as a collection of vertices some of which are connected by edges. It is the easy to draw a picture of a graph. Fig. 3.4 shows a drawing of K_4, the *complete graph* on four vertices. Graphs are termed *complete* if they contain all possible edges.

Given two graphs G and G', they are called *isomorphic* if there exists a correspondence between their vertex sets that preserves adjacency. Thus $G = (V, E)$ is isomorphic to $G' = (V', E')$ if there is a bijection $\phi : V \to V'$ such that $\overline{xy} \in E \iff \overline{\phi(x)\phi(y)} \in E'$. Clearly isomorphic graphs have the same number of vertices and edges.

A concept that we will use frequently is the *planar graph.* A graph is called *planar* if it is possible to represent it in the plane in such a way that the vertices correspond to distinct points and the edges to simple Jordan curves connecting the points of its endvertices in such a way that every two curves are either disjoint or meet only at a common endpoint. Such a representation is called a *plane graph.*

From Jordan's curve theorem [2] it follows that every plane graph divides the plane into distinct domains, called *faces.* The number of vertices, edges and faces of a plane graph are related via the following

Theorem 3.1 (Euler's formula) *If a connected plane graph G consists of n vertices, m edges, and f faces, then*

$$n - m + f = 2.$$

If all faces of a plane graph G, except for possibly the unbounded one, are triangles, then G is called a *triangulation.*

A final term that will be used a few times is the *edge graph* of a surface. Given a triangulated surface $\mathcal{S} = (V, E, T)$, the edge graph of $\mathcal{S}$ is simply the set of vertices V and the set of edges E of $\mathcal{S}$, considered as a graph.

A profound introduction to graph theory can be found in the monograph by Bollobás [13], from which the previous definitions are taken.

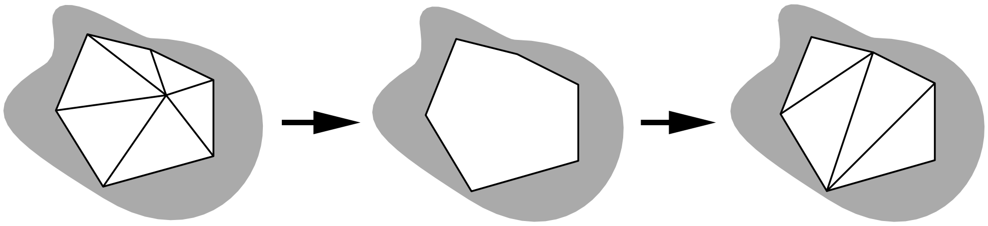

Figure 3.5: *The point removal primitive.*

3.2 Geometrical Primitives for Surface Simplification

In this section we will briefly describe several common simplification primitives. Each of them is a well-defined procedure that performs a strictly local surface reduction step.

3.2.1 Point Removal

This reduction scheme has been widely used, for example in [52, 66] and our own implementation. It is applicable wherever the surface is locally homeomorphic to a disk. It consists of removing a single vertex V from the surface, together with its star $\mathrm{St}\, v$, i.e., the set of all triangles that incident to it. If v was a manifold vertex, this leaves a circular hole bounded by the link of v. We will explain how we deal with non-manifold vertices in Sec. 3.4. The hole is then retriangulated without using vertex v (See Fig. 3.5).

The retriangulation is done in two steps. The straight line geometric realization of $\mathrm{Lk}\, v$ is first mapped to the plane. The resulting polygon is triangulated and the new triangles are inserted into the surface. For the triangulation, a *constrained Delaunay triangulation (CDT)* [34] is used.

The main problem is finding a suitable injective mapping of the link of the removed vertex into the plane. Cohen *et al.* [19] suggest finding a plane that allows the complete star to be projected orthogonally onto it without overlapping triangles. However, finding such a plane, which doesn't even necessarily exist involves linear programming and is therefore rather expensive. Also, since the triangles undergo a possibly severe distortion in this projection, the advantage of the Delaunay triangulation to produce 'nice' triangles gets lost in the process.

It is therefore preferable to use a different map which is known in differential geometry as the *polar map* [25]. It is defined as follows: Let $\mathbf{p}$ be the vertex to be removed and let $\mathbf{p}_k, 1 \leq k \leq n$, be its neighbors in any cyclic order. We take θ_k to be the angle of the triangle T_k at the vertex $\mathbf{p}$ and we compute the *accumulated angles*

$$\Theta_1 = 0, \quad \Theta_k = \sum_{j=1}^{k-1} \theta_j, \quad \text{and} \quad \Theta = \sum_{j=1}^{n} \theta_j.$$

We use those to define the map $\phi_p : \mathbb{R}^3 \to \mathbb{C}$ as

$$\phi_{\mathbf{p}}(\mathbf{p}) = 0 \quad \text{and} \quad \phi_{\mathbf{p}}(\mathbf{p}_k) = (\|\mathbf{p}_k - \mathbf{p}\| e^{i\Theta_k})^a, \quad a = 2\pi/\Theta \tag{3.1}$$

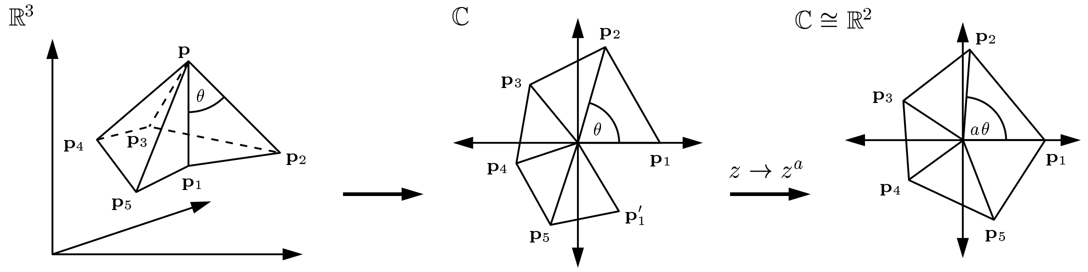

Figure 3.6: *The polar map: The star is first cut along $\overline{\mathbf{p}\,\mathbf{p}_1}$ and flattened out into the complex plane. The conformal map $z \to z^a$, $\quad a = 2\pi/\Theta$, then stretches it out so that it covers one full circle.*

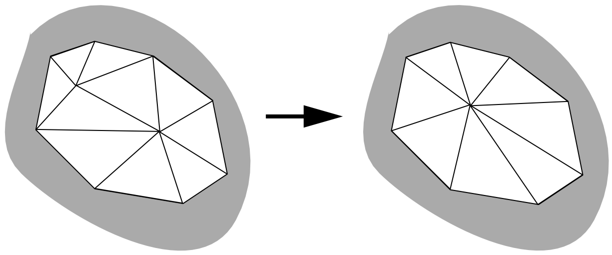

Figure 3.7: *Edge Contraction.*

which, due to the exponent a, maps the star of $\mathbf{p}$ with the total angle Θ to a full circle in the complex plane (Fig. 3.6). Identifying $\mathbb{C}$ with $\mathbb{R}^2$, we obtain the formula

$$\phi_{\mathbf{p}}(\mathbf{p}_k) = \|\mathbf{p}_k - \mathbf{p}\|^a \cdot \begin{pmatrix} \cos a\Theta_k \\ \sin a\Theta_k \end{pmatrix}.$$

3.2.2 Edge and Half Edge Contractions

Edge contraction is a simpler way of simplifying a surface than point removal, since the complicated part of retriangulating a hole is avoided. When doing edge contraction,

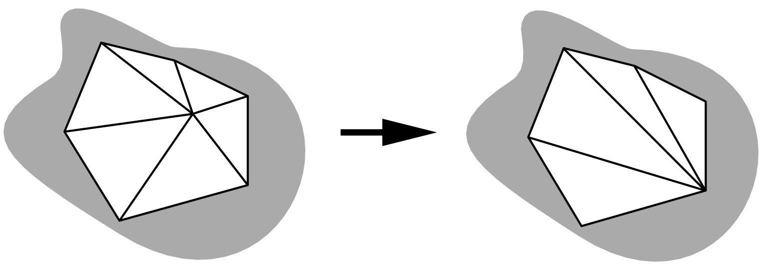

Figure 3.8: *Half Edge Contraction.*

errors are computed *per edge*, and a reduction step consists of contracting an edge into a single vertex (Fig. 3.7). Thus, when contracting two manifold vertices, we end up with one vertex, three edges, and two triangles less. There is some choice about where to put the new vertex. One can simply put it on one of the endpoints, or the midpoint of the contracted edge. Some error measures also allow to choose an optimal new position (see Sec. 3.3.2). This position is not guaranteed to be on the original surface, however.

Half edge contraction differs from true edge contraction in that the position of the new vertex is fixed. An edge is simply contracted by moving one endpoint into the other one and correspondingly removing the surrounding triangles (See Fig. 3.8). It can also be regarded as a point removal step with a particular retriangulation, namely the one that has all newly inserted edges emanate from a single vertex on the polygon boundary.

Since triangle quality is not taken into account, except when choosing the best edge to contract, one should expect the quality of the output surface to be inferior to a surface simplified using point removal. However, Kobbelt, for example, claims in [47] that the choice of reduction primitive has hardly any impact on surface quality anyways. In that case, half edge contraction is the preferable method since it is obviously the fastest one of them all.

3.2.3 Pair and Cluster Collapse

Pair collapse is a generalization of edge contraction. The difference is that now pairs of vertices can be merged that aren't necessarily connected by an edge. Thus, separate components of a surface can become connected. That is particularly useful when dealing with surfaces that consist of many small objects, such as, for example, the bones in a hand or foot.

Cluster collapse goes even further than that. Here, vertices are grouped into *clusters*, for example by partitioning space into a regular grid, and treating the content of each grid cell as a cluster. Each complete cluster can then be merged into a single vertex. Triangles with more than one vertex in a particular cluster disappear in the process. Since both pair and cluster collapse primitives merge vertices based on geometric reasoning alone, the topology of the surfaces changes. Therefore, they cannot be used to simplify boundary surfaces of computational domains, where topology preservation is crucial. On the other hand, cluster collapse can be very fast. These features have been exploited in *level-of-detail rendering* [58].

3.3 Cost Functions

3.3.1 Hausdorff Distance

The Hausdorff distance is a common way to define a distance between sets. If $(X, d(\cdot,\cdot))$ is a metric space (in our case $\mathbb{R}^3$, together with the euclidean distance) and $\mathcal{A}$ the set of all non-empty, compact subsets of X, we define for $a \in X, B \in \mathcal{A}$,

$$\operatorname{dist}(a, B) := \inf\{d(a,b),\ b \in B\}$$

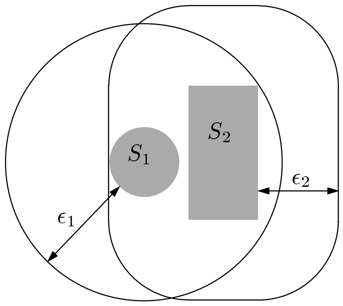

Figure 3.9: *An alternative way of defining the Hausdorff distance: The smallest ϵ that the two sets have to be enlarged by in order to contain each other [4]. Here it is ϵ_1, ϵ_2 being too small.*

the shortest possible distance from a to any point in B, and

$$d_H(A, B) := \max \left\{ \sup_{a \in A} \operatorname{dist}(a, B),\ \sup_{b \in B} \operatorname{dist}(b, A) \right\}. \tag{3.2}$$

This defines a metric on $\mathcal{A}$ [4]. In particular, we have

$$d_H(A, B) = 0 \iff A = B.$$

This definition captures the intuitive notion of distance of two sets rather well (Fig. 3.9). By regarding the Hausdorff distance between the simplified surface and its original, one is able to construct algorithms that obey global bounds for the geometric approximation error [44].

Globally considering the Hausdorff distance at all times, however, is impossible due to the prohibitive running time needed to compute it. A straightforward implementation of the formula (3.2) is infeasible for its asymptotic complexity. Given two surfaces having n_1 (resp. n_2) vertices and t_1 (resp. t_2) triangles, using a hierarchical data structure for the computation of $\operatorname{dist}(a, B)$ and replacing the sup over all points with the maximum over all vertices yields a running time of the order of $O(n_1 \log t_2 + n_2 \log t_1)$ with rather large constants. Thus, computing $d_H(S_{\text{Orig}}, S_{\text{Simp}})$ at each simplification step is out of the question.

We have found to be sufficient, however, to take into account only that part of the surface that changes during a simplification step. Therefore, in practice, we compute the Hausdorff error of a vertex v as follows. We hypothetically remove v and retriangulate the hole. We then define the Hausdorff error $\epsilon_H(v)$ of v to be the Hausdorff distance between that part of the original surface which has been simplified into the star of v and the retriangulation. This is easy, since the image of the parametrization function ϕ, which is defined on the surface at all times, gives us exactly the part of the original surface that we are interested in (the function ϕ will be introduced formally in Sec. 4.1). Also, in our implementation, we replace the unsimplified surface by the set of its vertices and and compute only the Hausdorff distance of this discrete subset to the retriangulation.

3.3.2 Quadric Error Metrics

Quadrics are an alternative way of measuring the geometric deviation of the simplified model to the original surface. They are used as the only quality measure in the algorithm of Garland and Heckbert [33]. There, each vertex $\mathbf{v}$ is associated with a set of planes $\mathbf{p}(\mathbf{v})$, and the error of the vertex $\mathbf{v} = [v_x\, v_y\, v_z\, 1]^T \in \mathbb{R}^4$ with respect to this set is defined as the sum of squared distances to its planes:

$$\Delta(\mathbf{v}) = \sum_{\mathbf{p}\in\mathbf{p}(\mathbf{v})} (\mathbf{p}^T\mathbf{v})^2,$$

with $\mathbf{p} = [a\, b\, c\, d]^T$ representing the plane with the equation $ax + by + cz + d = 0$ where $a^2 + b^2 + c^2 = 1$. The set of planes for each vertex is initialized with those planes spanned by the triangles that meet at that particular vertex. Therefore, the initial error at each vertex is zero.

The error metric given above can be rewritten as the quadratic form

$$\begin{aligned}
\Delta(\mathbf{v}) &= \sum_{\mathbf{p}\in\mathbf{p}(\mathbf{v})} (\mathbf{v}^T\mathbf{p})(\mathbf{p}^T\mathbf{v}) \\
&= \sum_{\mathbf{p}\in\mathbf{p}(\mathbf{v})} \mathbf{v}^T(\mathbf{p}\mathbf{p}^T)\mathbf{v} \\
&= \mathbf{v}^T \left(\sum_{\mathbf{p}\in\mathbf{p}(\mathbf{v})} \mathbf{K}_\mathbf{p} \right) \mathbf{v}.
\end{aligned}$$

This allows the definition of the *fundamental error quadric*

$$\mathbf{K}_\mathbf{p} = \mathbf{p}\mathbf{p}^T = \begin{pmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{pmatrix}$$

which permits to compute the squared distance of any point in space to the plane defined by $\mathbf{p}$. The matrices $\mathbf{K}_\mathbf{p}$ can be added to give a matrix $\mathbf{Q}$ representing the sum of the squared distances to all planes in the set. Each vertex keeps its own $\mathbf{Q}$, which, being symmetric, needs ten *floats* of memory space.

For simplifying the surface, Garland and Heckbert use *pair contraction*. Pairs of suitable vertices $(\mathbf{v}_1, \mathbf{v}_2)$ that are not necessarily connected by an edge are merged into a single new one $\bar{\mathbf{v}}$, which has a position other than $\mathbf{v}_1$'s and $\mathbf{v}_2$'s. When performing the contraction $(\mathbf{v}_1, \mathbf{v}_2) \rightarrow \bar{\mathbf{v}}$, the matrix $\bar{\mathbf{Q}} = \mathbf{Q}_1 + \mathbf{Q}_2$ is chosen to approximate the error at $\bar{\mathbf{v}}$. That way, $\bar{\mathbf{v}}$ 'inherits' the information about all the planes associated with $\mathbf{v}_1$ and $\mathbf{v}_2$. Strictly speaking, this simple additive rule overemphasizes the planes associated with *both* $\mathbf{v}_1$ and $\mathbf{v}_2$, and a set union following the rule $\bar{\mathbf{Q}} = \mathbf{Q}_1 + \mathbf{Q}_2 - \sum_{\mathbf{p}\in\mathbf{p}(\mathbf{v}_1)\cap\mathbf{p}(\mathbf{v}_2)} \mathbf{K}_\mathbf{p}$ should be preferable. However, this alternative is considerably more expensive and is therefore disregarded.

The error quadrics $\mathbf{Q}$ are now used for two purposes: Firstly, they determine the optimal position for new vertices. When contracting the pair $(\mathbf{v}_1, \mathbf{v}_2)$, it is obvious to choose the position of the new vertex $\bar{v}$ such that it minimizes the error form

$$\Delta(\bar{\mathbf{v}}) = \bar{\mathbf{v}}^T \bar{\mathbf{Q}} \bar{\mathbf{v}} = \bar{\mathbf{v}}^T (\mathbf{Q_1} + \mathbf{Q_2}) \bar{\mathbf{v}}.$$

Since this form is quadratic, finding its minimum is a linear problem. Setting the partial derivates

$$\frac{\partial \Delta}{\partial x} = \frac{\partial \Delta}{\partial y} = \frac{\partial \Delta}{\partial z} = 0$$

equal to zero leads to the system

$$\begin{pmatrix} \bar{q}_{11} & \bar{q}_{12} & \bar{q}_{13} & \bar{q}_{14} \\ \bar{q}_{12} & \bar{q}_{22} & \bar{q}_{23} & \bar{q}_{24} \\ \bar{q}_{13} & \bar{q}_{23} & \bar{q}_{33} & \bar{q}_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \bar{\mathbf{v}} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

The matrix on the left may not be invertible. In that case, Garland and Heckbert propose to pick the optimal vertex on the segment from $\mathbf{v}_1$ to $\mathbf{v}_2$, or if even that fails, to choose between $\mathbf{v}_1$ and $\mathbf{v}_2$ themselves, or their midpoint.

Secondly, the error of the vertex that is chosen that way can be used as the cost of the contraction that would produce it. This yields a quality measure for edges: the cost of the contraction of an edge is the error of the optimal vertex that results from that contraction. This allows to keep the *edges* in a priority queue and to contract them one after the other.

When geometric approximation is important, the approach of Garland and Heckbert usually yields very good results. Not restricting new vertices to lie on the original surface leads to simplified surfaces that are closer to the original on an average than other schemes. It is, however, exactly this feature, which makes it unsuitable for the construction of coarse surfaces that carry a parametrization. If the original surface is to be approximated by repeated refinement of the coarse surface *without* moving the base grid vertices, these have to be on the original surface, since they also belong to the refined surface. We have therefore decided not to choose this approach.

3.3.3 Triangle Aspect Ratio

It has been explained in Sec. 2.5 that the numerical accuracy of finite element computations depends directly on the well-formedness of the tetrahedral elements that make up the computation domain. One good measure for this well-formedness is the *element aspect ratio*, defined as the ratio $\rho = R/r$ between circumsphere radius R and insphere radius r. This leads directly to a criterion for the numerical suitability of triangulated boundary surfaces, requiring that the *triangle aspect ratio* should stay below a reasonable limit. This latter ratio, defined analogously to the tetrahedral case as the ratio $\rho_\Delta = R_\Delta / r_\Delta$, can be expressed easily in terms of the three triangle edge lengths a, b, c,

$$\rho_\Delta = \frac{2abc}{(-a+b+c)(a-b+c)(a+b-c)}. \tag{3.3}$$

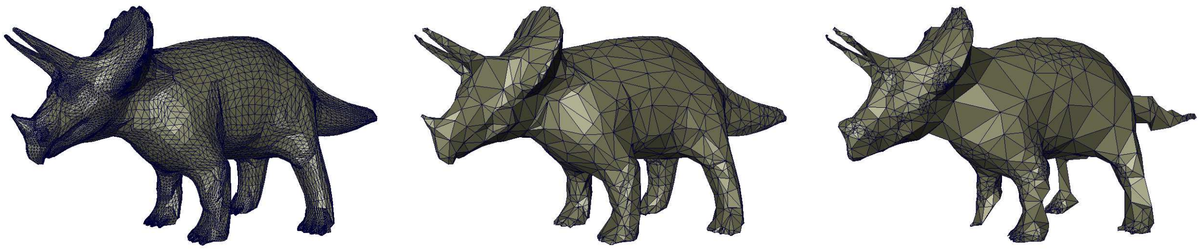

Figure 3.10: *A comparison of different error criteria: One the left, a model of a fierce dinosaur made of 22638 triangles. In the center, the dinosaur has been simplified down to 2638 triangles, using only the one-sided Hausdorff distance as the error function. The worst triangle has an aspect ratio of 20.3. In contrast to that, the third model has been constructed paying attention only to the triangle aspect ratio. It is not as good a geometric approximation as the second one, yet its triangle aspect ratios are bounded from above by 4.63. A simplification algorithm usable for finite element grids should obviously combine both approaches.*

The aspect ratio $\rho_\triangle$ takes on a value of 2 for the 'best' triangle, namely the equilateral one and goes to infinity with increasing deformation. If the three points forming the triangle are actually collinear the denominator of the formula above gets zero. Be warned, though, that on a computer, the expression (3.3) might yield *negative* values when finite-precision arithmetic is used.

The triangle aspect ratio is a criterion that is mainly in use for numerical reasons. There are hardly any reasons why to use it in a strictly geometrical context. It subsumes the restrictions on the interior angles of the triangles that have been described in Sec. 2.5, is very easy to implement, and is fast to compute. Note, though, that it is not enough to favour surface reduction operations that create triangles with a low aspect ratio. At the same time one also has to try to remove triangles with bad aspect ratio in order to obtain satisfying results.

3.3.4 Local Curvature and Triangle Area

The local curvature and triangle area are two additional geometric quality criteria. They have been used together in the surface simplification algorithm of Lee *et al.* [52]. The main idea is that the triangle regions most suitable for simplification are those that consist of small triangles and are relatively flat.

While the computation of triangle areas is quite evident, the local curvature deserves a few words. Classical differential geometry defines a scalar curvature only for surfaces that are twice differentiable [45]. However, the triangulated surfaces we're dealing with are only piecewise linear, which means they are not necessarily $\mathcal{C}^1$. One therefore computes an approximative *discrete local curvature.* That is done by fitting a two-dimensional quadratic polynomial through the star of each vertex and taking the curvature thereof (see Kobbelt's article [46] for a more detailed account).

Considering the discrete curvature is a somewhat costly approach, which is not guaranteed to lead to good results, since the information about the original surface is continuously discarded. Only the characteristics of the current surface are taken into account when choosing a next spot to simplify.

3.3.5 Combining Several Criteria

A word has to be said about using several different criteria together. Our own implementation currently incorporates the triangle aspect ratio and the one-sided Hausdorff distance, however, more criteria can be inserted without difficulty. The question arises as to how the errors given by the different quality measures can be combined to yield a behaviour that can be controlled intuitively by the user. We currently use the following rule. For n different quality criteria we let the user choose n weights $\lambda_1, \ldots, \lambda_n$, each between zero and one. We then use a convex combination of the individual errors $\epsilon_1, \ldots, \epsilon_n$ as the total error ϵ

$$\epsilon = \frac{\sum_{i=1}^{n} \epsilon_i \lambda_i}{\sum_{i=1}^{n} \lambda_i}. \tag{3.4}$$

The problem is that some measures, such as the triangle aspect ratio, are scale-invariant, whereas others, like the Hausdorff distance, are not. The choice of a suitable set of λ_i will therefore depend on the overall dimensions of the model. Unless some suitable normalizing is implemented for each criterion, the user is required to anticipate this effect.

3.4 Manifold and Non-Manifold Triangulated Surfaces

Many current simplification algorithms implicitly assume that only manifold surfaces (possibly with a boundary) need to be dealt with [52, 66]. This is sufficient for a wide range of applications. However, the material interface surfaces arising in PDE problems with discontinuously varying coefficients are of a more general type and need a more careful handling.

Remember that a triangulated surface $\mathcal{S} = (V, E, F)$ is called a *manifold*, if for each vertex $v \in V$, $\operatorname{Lk} v$ is a simple loop. A triangulated surface is called manifold *with a boundary*, if there are *boundary vertices* whose links form simple paths.

In order to handle non-manifold vertices, we adopt the approach taken by Lee *et al.* [52] for dealing with *feature lines*. First, we classify surface edges and vertices according to their local topology. We call an edge an *n-fold feature edge*, $n \geq 3$, if it borders exactly n triangles. By consequence, an edge that borders only a single triangle is called *boundary edge* or a *1-fold feature edge*. If the edge under consideration is a neighbor of exactly two triangles, we call it a *regular edge*. This classification induces a similar one

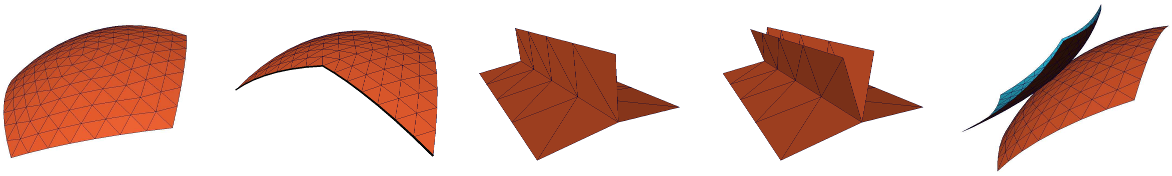

Figure 3.11: *i) A topological disk; ii) a boundary; iii) a t-junction; iv) an n-junction of order four; v) two sheets touching in a single point.*

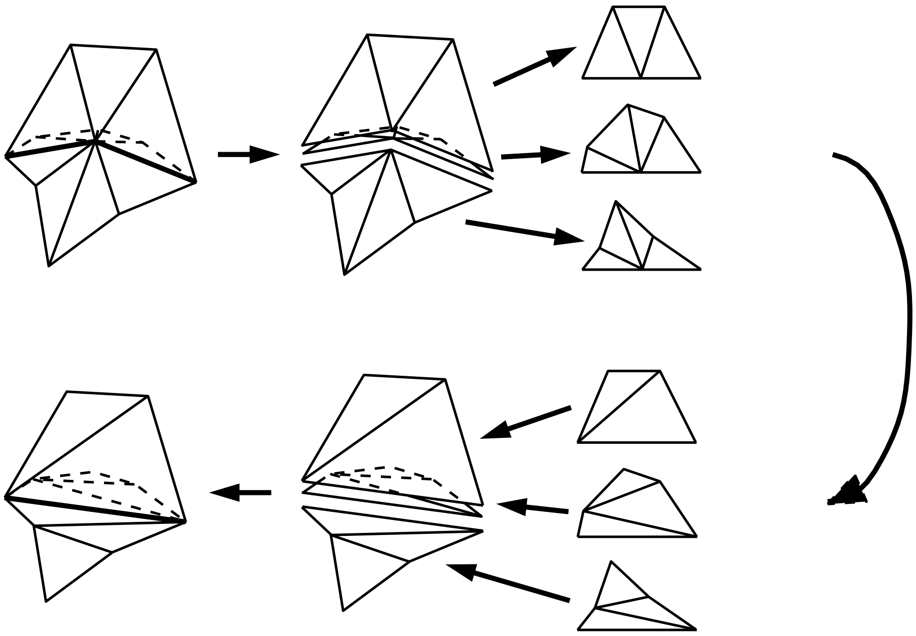

Figure 3.12: *Removal of a vertex on a feature line of third order: The three adjacent half-stars are separately mapped to the plane and retriangulated. The result still contains the feature line.*

for vertices, which we describe with the corresponding handling by our point-removal simplification algorithm:

- **Regular vertices** are vertices that do not touch any feature edges and that are not points of incidence of several surface sheets. Those points are manifold vertices as defined in Sec. 3.1. Both priority computation and the actual point removal can be performed as described in Sec. 3.2.1.

- **Vertices on a feature line** are those vertices that touch exactly two feature edges of the same order n and whose link is connected. The neighborhood of those vertices is equivalent to a collection of n half-disks which are glued together at the two feature edges (see Fig. 3.11, *iv)*). All quality measures described in Sec. 3.3 are therefore computed *per half-disk*, and the arithmetic mean of all of them is taken as the priority of the vertex. Instead of computing a surface curvature, we take the discrete curvature of the path described by the two incident feature edges. The point removal follows along the same lines. We map each half-disk onto the positive half-plane using a modified polar map, and retriangulate each polygon separately. The result is a consistent retriangulation of the neighborhood of the removed vertex, with the two feature edges now replaced by a single one (Fig. 3.12).

- **Feature Vertices** are all those vertices that do not fall under one of the two groups above. They occur for example when two surface sheets touch in a single vertex as in Fig. 3.11, *v)*, or when four materials meet in a point. Even more cases are envisagable if we drop the restriction that the surface should bound well-defined compartments. The shark-fin constellation in Fig. 3.3, right, shows such a vertex, which cannot occur in surfaces describing computational domains but which should nevertheless be handled in order to ensure a robust algorithm. A theoretically sound treatment should formulate proper treatment for all such cases. However, in practice, they do not occur very frequently. A surface describing a hyperthermia patient consisting of half a million vertices contains generally around five hundred feature vertices. For all practical purposes, it is therefore safe to omit these vertices and simply never remove them.

3.5 Topology Changes

A word of warning is due here. The simplification primitives described in Sec. 3.2 cannot be used without some precaution. A simplification algorithm that is intended for mesh generation purposes has to ward off any *topology changes* and *self-intersections*. These seemingly related topics are in fact strictly distinct.

Surface self-intersections are a geometric property. They occur when the abstract simplicial complex which describes the surface, together with prescribed vertex positions, does not have an injective straight-line embedding in $\mathbb{R}^3$. We give a description of how the implementation of our simplification algorithm deals with self-intersections in Sec. 3.6.

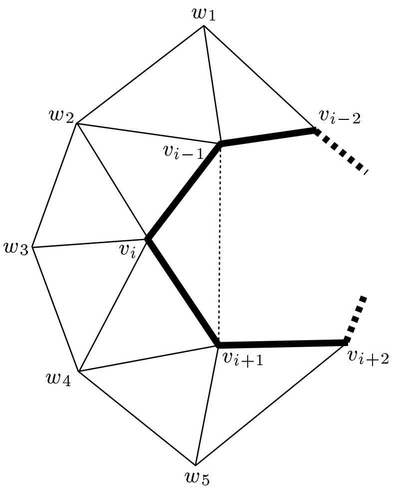

Figure 3.13: *Edge* $\overline{v_{i-1}v_{i+1}}$ *does not already exist in the surface. The link of* v_i *is* v_{i-1}, w_2, w_3, w_4, v_{i+1}. $\mathrm{Lk}\, v_{i-1}$ *is* v_{i-2}, w_1, w_2, v_i *and* $\mathrm{Lk}\, v_{i+1}$ *is* v_i, w_4, w_5, v_{i+2}. *If* $\overline{v_{i-1}v_{i+1}}$ *is added to the surface, the new link of* v_i *is the cycle* $v_{i-1}, w_2, w_3, w_4, v_{i+1}, v_{i-1}$. $\mathrm{Lk}\, v_{i-1}$ *is* $v_{i-2}, w_1, w_2, v_i, v_{i+1}$ *and* $\mathrm{Lk}\, v_{i+1}$ *is* $v_{i-1}, v_i, w_4, w_5, v_{i+2}$. *Thus, the remaining polygon can be triangulated by induction.*

In contrast, the topology of a triangulated surface is a strictly combinatorial property. No surrounding space is necessary to define the topology of such a surface. Hence, in order to prevent topology changes it is sufficient to argue in terms of the underlying abstract simplicial complex exclusively.

The decision of whether or not two two-dimensional simplicial complexes have the same topological type is as hard as deciding whether or not two graphs are isomorphic [61]. No polynomial time algorithm for this is known, but it is also unknown whether the graph isomorphism problem is NP-complete [32]. Edelsbrunner [28] gives sufficient local conditions for the preservation of the topological type when simplifying surfaces by contracting edges. Since our implementation works with the point removal primitive, we need different conditions. We found the following condition to work well in practice. We prove it for the case of a manifold surface.

Lemma 3.1 *Let* $\mathcal{K} = (V, F)$ *be a two-dimensional abstract simplicial complex, and assume that* $\mathcal{K}$ *is a* manifold. *Remove a vertex* $v \in V$ *of degree* n *together with its star* $\mathrm{St}\, v$ *from* $\mathcal{K}$ *and choose a retriangulation of the resulting hole. That retriangulation will add* $n-3$ *one-faces to the complex which are not part of* $\mathrm{Lk}\, v$. *Call the set of those one-faces* $E = \{e_1, \ldots, e_{n-3}\}$. *Then the following holds:*

If no $e_j \in E$, $\quad j = 1, \ldots, n-3$ *is a one-face in* $F \setminus \mathrm{St}\, v$ *then* $\mathcal{K}' = (V \setminus \{v\}, F \setminus \mathrm{St}\, v)$ *together with the retriangulation is a manifold.*

Proof: Since $\mathcal{K}$ is a manifold, for each vertex $v \in V$, $\mathrm{Lk}\, v$ is a simple closed cycle. Assume that a point removal step has removed vertex v of degree n and that a retriangu-

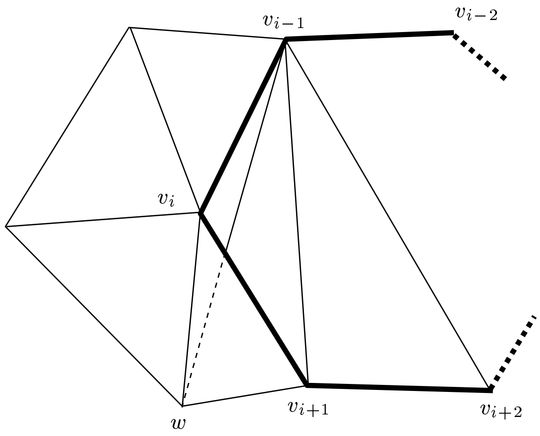

Figure 3.14: *A constellation which leads to a topology change. The link of v_{i+1} consists of v_{i+2}, v_{i-1}, w, and v_i. Adding the edge $\overline{v_{i-1}v_{i+1}}$ would prolong it to v_{i+2}, v_{i-1}, w, v_i, v_{i-1}, which contains a cycle.*

lation of the resulting hole is given. This retriangulation contains a set of one-faces. We denote the subset of those one-faces (we shall call them edges in the following), which were not part of $\mathrm{Lk}\, v$, as $E = \{e_1, \ldots, e_{n-3}\}$. We assume that no edge $e_j \in E$ exists already in $\mathcal{K}$.

Since the manifold property of a simplicial complex is defined using only the link of each vertex, it is sufficient to show that the vertices that have v in their link remain manifold vertices. Call those vertices $v_1, \ldots, v_n$ in any cyclic order. We prove the lemma by induction over n. Let first $n = 3$. Then the hole is a triangle and can be triangulated without introducing new edges and the assertion follows. Assume now that the lemma is proved for all $k < n$. There exists an $e_j \in E$ such that $e_j = \{v_{i-1}, v_{i+1}\}$ for some v_i. If the edge $e_j = \{v_{i-1}, v_{i+1}\}$ does not already exist in $\mathcal{K}$, the links $\mathrm{Lk}\, v_{i-1}$, $\mathrm{Lk}\, v_i$, and $\mathrm{Lk}\, v_{i+1}$ each form simple paths. Insertion of e_j together with the triangle $\{v_{i-1}, v_i, v_{i+1}\}$ then completes $\mathrm{Lk}\, v_i$ to a simple cycle (see also Figs. 3.13 and 3.14). $\mathrm{Lk}\, v_{i-1}$ and $\mathrm{Lk}\, v_{i+1}$ still are simple paths, the links of all other vertices remain unchanged. The hole is reduced to $n-1$ vertices and can be triangulated without topology changes by induction.□

We empirically found the condition set forth in Lemma 3.1 to prevent topology changes in all cases. We therefore state its generalization as a conjecture.

Conjecture 3.1 *Let $\mathcal{K} = (V, F)$ be a two-dimensional abstract simplicial complex. Remove a vertex v of degree n together with its star $\mathrm{St}\, v$ from $\mathcal{K}$ and choose a retriangulation of the resulting hole. That retriangulation will add one-faces to the complex which are not part of $\mathrm{Lk}\, v$. Call the set of those one-faces $E = \{e_1, \ldots, e_{n-3}\}$. Then the following holds:*

If no $e_j \in E$ is a one-face in $F \setminus \mathrm{St}\, v$ then $\mathcal{K}' = (V \setminus \{v\}, F \setminus \mathrm{St}\, v)$ together with the retriangulation is of the same topological type as $\mathcal{K}$.

3.6 Self-Intersections

When working with surfaces that are intended to be used in mesh generation, it is of great importance that they do not self-intersect. Whereas surfaces that are only judged by their rendered, visual quality may very well contain triangles that intersect others, triangulated surfaces that serve as input for a 3D grid generator cannot have parts that pervade other parts of the same surface. This is because those surfaces are interpreted as *interface surfaces* between areas with different physical properties, for example, different tissue types in the hyperthermia application. Therefore, the surfaces are expected to partition space into well-defined regions. This obviously excludes self-intersecting surfaces.

Unfortunately, full intersection tests are very expensive. Given a triangulated surface with N triangles, the brute force strategy of testing all triangle against all others has quadratic complexity, which can be reduced to $O(N \log N)$ by using a suitable spatial data structure such as an octree. Looking for self-intersections induced by a single simplification step necessitates the test of all new triangles against the complete surface. As we are using the point removal primitive, the new triangles are those that are introduced by the retriangulation of the hole left by the removal of a vertex. If that vertex was a manifold vertex of degree n, then $n-3$ triangles are needed to retriangulate the hole. The number of new triangles at each step is therefore bounded by the degree of the removed vertex. With the use of Euler's formula (Theorem (3.1)) it can be shown that in a planar graph, vertices have an average degree of six [13]. Since the edge graphs of triangulated surfaces occuring *in practice* generally consist to a large extend of planar graphs, one can expect the average degrees of vertices in triangulated boundaries to be bounded by a similar constant as well. Therefore, the average number of triangles incident to a vertex is independent of the overall size of the surface. Considering all this, testing whether the application of the point removal primitive has introduced any self-intersections has a running time that is linear in the number of triangles of the surface, respectively $O(\log N)$ when using the octree.

Several strategies exist for reducing the computational effort. The AMIRA surface simplifier, for example, implements several variants of testing newly inserted triangles not against the full surfaces but only against the topological neighborhood of the reduction. This reduces running time considerably, since this neighborhood is chosen to be small compared to the complete model. DeRose *et al.* [21] use suitable *two-dimensional* hierarchical data structures that partition the parameter domains of their spline- and subdivision surfaces.

Neither of those approaches is truly applicable to our purpose. The parameter partition strategy works well with manifold surfaces used for animation, since the data hierarchy can stay the same for all time steps. It is, however, unclear how to use their parameter domain approach with our more general surfaces. The AMIRA approach, on the other side cannot guarantee to find all self-intersections. If the new triangles created during a reduction step happen to intersect others that are topologically far away this will go unnoticed. These cases do happen with quite some regularity when dealing with patient models, as those sometimes contain long, sheet-like structures such as human skin.

In order to ensure the absence of self-intersections, we have therefore chosen to implement the full intersection test. That means that before applying a surface reduction primitive, we check whether it would lead to intersecting triangles, and we do not apply it in the affirmative case. Using an octree to store the surface, this algorithm has a running time of $O(m \log N)$ if m vertices are removed from a surface with N triangles. However, since reliably testing for intersections between triangles is somewhat involved, the constant hidden in the asymptotic notation is high. It can be reduced by considering that when simplifying an intersection-free surface using only point-removal operations, a reduction step that introduces an intersection will always have a new triangle be pierced by an edge of the surrounding surface. The inverse case cannot occur. It is therefore sufficient to only test newly inserted triangles against all *edges* of the surface. Since triangle-edge intersections are far easier to compute than triangle-triangle ones, this reduces the asymptotic constant considerably.

In some cases it does also happen that the few triangles that form the retriangulation of a hole intersect *each other*. This type of intersection is of course not detected by testing all new triangles against the *remaining* surface. It therefore has to be handled separately. Since, as has been shown above, the number of triangles involved in a retriangulation is constant on an average, this does not increase the asymptotic running time of the algorithm.

4 Parametrizations

This chapter is the core of this thesis. As explained in Chapter 2, we need a surface simplification, but additionally, we want to keep the original surface as a parametrization over its simplified descendant. That way, every time the FE solver tells us to subdivide a boundary triangle, we can access this information and have the newly inserted triangles approximate the original surface as closely as possible.

Surface parametrizations have been studied intensively both in pure and applied mathematics. The bulk of the work has been done on parametrizations of topological disks over subsets of $\mathbb{R}^2$. Since the patient models we create, however, are non-manifold surfaces, i.e., they are not even locally homeomorphic to a disk, special algorithms have to be implemented to use specific parametrizations on such surfaces. We will first give mathematically sound descriptions of several standard parametrizations over compact subsets of the real plane and then we'll show heuristics of how to use them on non-manifolds.

4.1 Surface Parametrizations

We start by giving an exact definition of the term *surface parametrization.*

Definition 4.1 *Let $\mathcal{S}$ and $\mathcal{S}'$ be two surfaces in $\mathbb{R}^3$. Let $\phi : \mathcal{S} \to \mathbb{R}^3$ be a single-valued function defined on $\mathcal{S}$ such that $\phi(\mathcal{S}) = \mathcal{S}'$ and such that its inverse ϕ^{-1} exists on $\mathcal{S}'$. Then ϕ is called a* **parametrization** *of $\mathcal{S}'$ over $\mathcal{S}$. Its inverse ϕ^{-1} is called a* **projection** *of $\mathcal{S}'$ onto $\mathcal{S}$.*

It is a basic result in differential geometry that such a function ϕ does not always exist. If $\mathcal{S}$ and $\mathcal{S}'$ are manifolds, a necessary condition for a ϕ to exist is that $\mathcal{S}$ and $\mathcal{S}'$ have to have isomorphic homology groups. Unfortunately, this condition is not sufficient [31].

It is also quite evident that the function ϕ is not unique. In fact, we will spend a considerable effort on the task of finding a 'good' function, with the exact definition of 'good' to be given in Sec. 4.4.

In order to work with functions defined on a surface $\mathcal{S}$, we have to introduce coordinates on $\mathcal{S}$. Except for a few trivial cases, it is not possible to use one global coordinate system for the whole surface. One therefore works with a finite set of local coordinate systems, each valid for a subset U_i of $\mathcal{S}$. Of course, the union $U = \bigcup_i U$ of those subsets has to cover all of $\mathcal{S}$. Such a set $\mathcal{U}$ of subsets U_i, together with a local coordinate system on each U_i, is called an *atlas* of $\mathcal{S}$ [40].

The choice of a suitable atlas is made easy when $\mathcal{S}$ is the geometric realization of a triangulated surface (cf. Sec. 3.1). For the set $\mathcal{U}$ of coordinate patches we choose the set of the convex hulls of the two-dimensional faces. The union of those convex hulls obviously covers $\mathcal{S}$. For each triangular face $f_i \in F$, we arbitrarily number its three vertices, and use the barycentric coordinates with respect to those vertices as the coordinate system on U_i. This allows us to specify each point $\mathbf{p} \in \mathcal{S}$ by giving a face number i and a pair of two coordinates (p_x, p_y).

4.2 Parametrizations of Topological Disks

After having introduced surface parametrizations in general, we will talk now about an important subclass of such parametrizations. We call a *topological disk* $\mathcal{D}$ a surface that can be mapped diffeomorphically onto the unit circle in $\mathbb{R}^2$. Parametrizations of topological disks over compact, simply connected subsets of $\mathbb{R}^2$ are far easier to handle mathematically than parametrizations between general surfaces. At the same time, algorithms for the construction of parametrizations for topological disks can serve as building blocks for parametrization algorithms for more general surfaces, as will be shown in Sec. 4.5.

Many algorithms for the parametrization of disk-like surfaces have been proposed. We'll discuss several alternatives, with a special interest in *piecewise linear functions*, since we're dealing with triangulated surfaces. Note that the three algorithms we present have in common that the boundary of the parameter domain $U \subset \mathbb{R}^2$ is fixed in advance. This is an unnecessary restriction for some purposes. The interested reader is refered to [49, 68].

4.2.1 Harmonic Functions

Harmonic functions are among the best-studied functions in mathematics. That is because they have many desirable properties. They are defined as those functions $f : \mathbb{R}^n \to \mathbb{R}$ that fulfill *Laplace's Equation*

$$\Delta f = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i^2} = 0. \tag{4.1}$$

This equation, together with its cousin the *Poisson Equation*, is ubiquitious in modern physics, which makes harmonic functions very interesting.

Let $\mathcal{M}$ be a compact Riemann surface with boundary $\partial\mathcal{M}$. Suppose that $\mathcal{M}$ is diffeomorphic to a disk and let $U \subset \mathbb{R}^2$ be a compact, convex region with boundary ∂U. It is shown in [25] that for a piecewise smooth boundary function

$$F_\partial : \partial\mathcal{M} \to \partial U$$

there is a homeomorphism

$$F : \mathcal{M} \to U$$

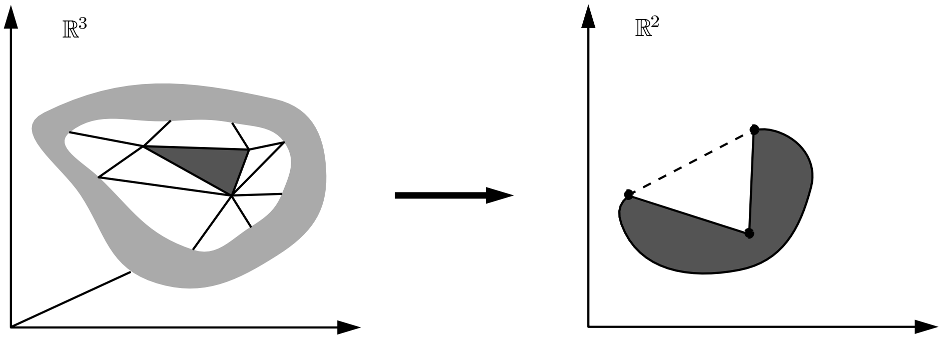

Figure 4.1: *Even though harmonic functions are bijections, they don't map straight lines to straight lines. Therefore, piecewise linear harmonic functions can contain triangles that are flipped over.*

with

$$F|_{\partial\mathcal{M}} = F_\partial$$

which is a diffeomorphism on the interior of $\mathcal{M}$ and whose coordinate functions f_1, f_2 fulfill the Laplace equation (4.1). That function F is called a *harmonic map*.

What makes harmonic maps interesting for our purposes is that, intuitively spoken, they 'minimize distortion'. If g is a Riemannian metric on $\mathcal{M}$, then in [25] it is shown that in the space of smooth functions $f : \mathcal{M} \to \mathbb{R}$ with fixed boundary values $f_\partial : \partial\mathcal{M} \to \mathbb{R}$, the harmonic functions minimize the *harmonic energy functional*

$$E_g[f] = \frac{1}{2}\int_{\mathcal{M}} |df|_g^2 \, dA_g,$$

which intuitively measures the stretching energy. Also, harmonic maps are *conformal*, i.e., they preserve angles.

When working with triangulated surfaces, the situation is less favorable. If the surface consists of piecewise affine sets bounded by straight lines, its harmonic mapping to the parameter plane will still be a bijection. However, straight lines do *not* get mapped to straight lines. Therefore, if the function is only evaluated at the surface vertices and completed via linear interpolation, the resulting map is not injective anymore. The image of the surface is 'folded over' (Fig. 4.1). As the resolution of the triangulated surface goes to infinity, the non-injectivity eventually disappears, however, this result has hardly any practical value.

4.2.2 The Barycentric Parametrization

The lack of injectivity of piecewise linear harmonic functions mentioned in the last section more or less probihits their use in actual applications. If the projection ϕ^{-1} is not injective, then its inverse, the parametrization function ϕ will be multi-valued, and therefore, when querying a point in the parameter domain for its position in real space, there will not be a unique answer. One can evade the problem by using some local triangle-unflipping schemes (see, e.g., [52]), but those obviously destroy the nice properties of the harmonic functions.

A general method for obtaining parametrizations of triangulated surfaces that are guaranteed to be bijections comes from graph theory. Note that finding a parametrization for a given triangulated surface $\mathcal{S}$ over a compact subset $U \subset \mathbb{R}^2$ is equivalent to finding a planar graph with straight line edges embedded into U and isomorphic to the edge graph of $\mathcal{S}$. Tutte [72, 73] gave a constructive proof employing a method known as the *barycentric mapping* for finding such a planar triangulation.

The barycentric mapping is based on the condition that in a fold-over free planar embedding of a graph, each vertex should be a *convex combination* of its neighbors. Let $G = (V, E)$ be the edge graph of $\mathcal{S}$ with $|V| = N$. We label a subset of $V^+ \subset V, |V^+| = n$ the *interior vertices* and we label all vertices in $V \setminus V^+$ as *boundary vertices*. For the sake of convenience, we order the vertices $\mathbf{v} \in V$ such that $\mathbf{v}_1, \dots, \mathbf{v}_n$ denote the interior vertices and that $\mathbf{v}_{n+1}, \dots, \mathbf{v}_N$ denote the boundary vertices.

Let then $\mathbf{u}_i \in \mathbb{R}^2$ denote the parameter position of vertex $\mathbf{v}_i$. We require

$$\mathbf{u}_i = \sum_{j=1}^{N} \lambda_{ij} \mathbf{u}_j \tag{4.2}$$

for all $1 \le i \le n$ with $\lambda_{ij} > 0$ if the vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ are connected by an edge and $\lambda_{ij} = 0$ if they are not. Additionally, convexity requires that

$$\sum_{j=1}^{N} \lambda_{ij} = 1, \quad 1 \le i \le n. \tag{4.3}$$

If we assign fixed parameter positions $\mathbf{u}_i$ to the boundary nodes $\mathbf{v}_i, \quad n+1 \le i \le N$, equation (4.2) can be rewritten as

$$\mathbf{u}_i - \sum_{j=1}^{n} \lambda_{ij} \mathbf{u}_j = \sum_{j=n+1}^{N} \lambda_{ij} \mathbf{u}_j, \quad 1 \le i \le n.$$

By considering the two components $u_i^{(x)}$ and $u_i^{(y)}$ of $\mathbf{u}_i$ separately, this amounts to two sparse matrix equations

$$AU^{(x)} = B^{(x)} \quad \text{and} \quad AU^{(y)} = B^{(y)}. \tag{4.4}$$

Here, $U^{(x)}$ and $U^{(y)}$ are the column vectors $(u_1^{(x)}, \dots, u_n^{(x)})^T$ and $(u_1^{(y)}, \dots, u_n^{(y)})^T$ respectively. The matrix A is $n \times n$ having elements

$$a_{ij} = \begin{cases} 1 & \text{if} \quad i = j, \\ -\lambda_{ij} & \text{else.} \end{cases}$$

Floater [30] shows that A is nonsingular, i.e., that (4.4) always has a unique solution. It can be computed efficiently using appropriate iterative sparse solvers such as BiCGSTAB [75].

Tutte himself, in his work on graph drawing, chose the case $\lambda_{ij} = 1/d_i$, d_i the degree of $\mathbf{v}_i$, which fulfills the conditions (4.2) and (4.3). That way, each vertex is located in the *barycenter* of its neighbors, hence the name *barycentric mapping.*

4.2.3 The Shape-Preserving Parametrization

The barycentric mapping guarantees a fold-over-free embedding of the surface edge graph into the parameter domain. However, no geometric information about the original surface enters the computation of the parameter positions at all. All edges in the original surface contribute an equal weight to the computation, regardless of their lengths. This leads to an approximately uniform distribution of the points in the parameter plane, independently of whether they actually are distributed evenly on the surface. Thus, the result might be arbitrarily far from being conformal or isometric.

For most surfaces diffeomorphic to a disk it is impossible to find an isometry that maps them onto the unit disk in $\mathbb{R}^2$. This follows from the following

Theorem 4.1 (Gauss's Theorema Egregium [31]) *Let $\mathcal{S} \subset \mathbb{R}^3$ be a surface and for any point $\mathbf{x} \in S$ let κ_1, κ_2 denote the principal curvatures. Then the Gauss curvature*

$$K = \kappa_1 \kappa_2$$

is an isometry invariant.

Since the Gauss curvature of a plane is zero everywhere, it follows that isometric mappings to the plane can only exist for surfaces whose Gauss curvature K is likewise zero everywhere. This is the case only for the very restricted class of surfaces that have at least one principal curvature equal to zero at all points. An example is the cylinder.

As a compromise, Floater [30] suggested to choose the weights λ_{ij} in such a way as to have the resulting map $\mathcal{P} : \mathcal{S} \to U$ fulfill a certain *reproduction property.* He calls a mapping $\mathcal{P}$ *shape-preserving* if it fulfills the following condition:

Definition 4.2 *Let $\mathcal{S}$ be a triangulated surface with n interior nodes $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and $N - n$ boundary nodes $\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N$ and let $\mathcal{P}$ be a parametrization of $\mathcal{S}$ over a convex polygon U with the boundary points $\mathbf{u}_{n+1}, \ldots, \mathbf{u}_N$ such that $\mathcal{P}(\mathbf{x}_i) = \mathbf{u}_i, \quad n+1 \le i \le N$. Then $\mathcal{P}$ is called* **shape-preserving** *if the following condition holds:*

If $\mathcal{S}$ is planar and if the mapping from $\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N$ to $\mathbf{u}_{n+1}, \ldots, \mathbf{u}_N$ is an affine one, then $\mathcal{P}$ is an affine mapping of $\mathcal{S}$.

This type of mapping is widely used in surface parametrization algorithms [35, 36].

In order to compute the new weights λ_{ij}, we construct a shape-preserving parametrization for the closed neighborhood $\operatorname{St} v \cup \operatorname{Lk} v$ of each vertex v and then accumulate the resulting weights. For each vertex $\mathbf{v}_i$ on the surface, let d_i be the number of its neighbors and let $\mathbf{x}_i$ be its position in space. Further let $\mathbf{x}_{j_1}, \ldots, \mathbf{x}_{j_{d_i}}$ be the positions of the

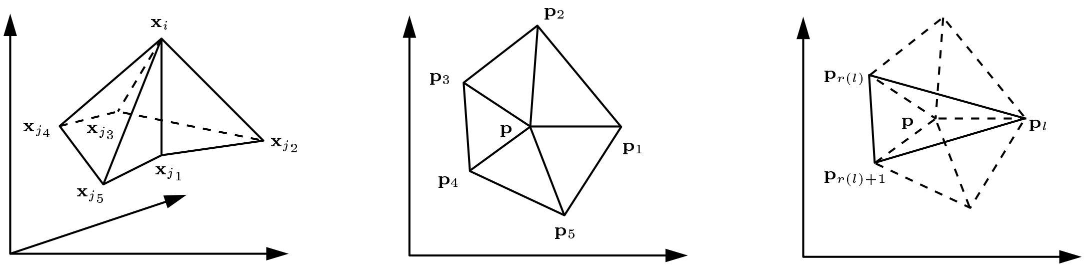

Figure 4.2: *Constructing the local weights for a vertex* $\mathbf{v}_i$ *at position* $\mathbf{x}_i$*. The* star *of the vertex is mapped to the plane; for each of its neighbors* $\mathbf{p}_l$*, two further neighbors* $\mathbf{p}_{r(l)}$ *and* $\mathbf{p}_{r(l)+1}$ *are chosen such that* $\mathbf{p}$ *lies in the triangle* $\Delta\mathbf{p}_l\mathbf{p}_{r(l)}\mathbf{p}_{r(l)+1}$.

neighbors of vertex $\mathbf{v}_i$, in any cyclic order. We use the *polar map* defined in Sec. 3.2.1 to map $\operatorname{St} v \cup \operatorname{Lk} v$ to the parameter plane and obtain a plane image with $\mathbf{v}_i$ being mapped to a parameter position $\mathbf{p} = 0$. The images of the neighbors form a star-shaped polygon $\mathbf{p}_1, \ldots, \mathbf{p}_{d_i}$.

We now wish to find suitable $\lambda_{ij_k} \geq 0$ satisfying

$$0 = \mathbf{p} = \sum_{k=1}^{d_i} \lambda_{ij_k} \mathbf{p}_k \quad \text{and} \quad \sum_{k=1}^{d_i} \lambda_{ij_k} = 1. \tag{4.5}$$

If the polygon forms a triangle $\Delta\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3$, we choose the well-known *barycentric coordinates* of $\mathbf{p}_i$ with respect to $\Delta\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3$:

$$\lambda_{ij_1} = \frac{\text{area}(\Delta\mathbf{p}\mathbf{p}_2\mathbf{p}_3)}{\text{area}(\Delta\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3)}, \quad \lambda_{ij_2} = \frac{\text{area}(\Delta\mathbf{p}_1\mathbf{p}\mathbf{p}_3)}{\text{area}(\Delta\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3)}, \quad \lambda_{ij_3} = \frac{\text{area}(\Delta\mathbf{p}_1\mathbf{p}_2\mathbf{p})}{\text{area}(\Delta\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3)}.$$

For the general case there is some choice. Floater suggested the following construction: Since the polygon is star-shaped, for each polygon vertex $\mathbf{p}_l$ there is a unique pair of polygon vertices $\mathbf{p}_{r(l)}$ and $\mathbf{p}_{r(l)+1}$ such that $\mathbf{p}$ lies in the triangle $\Delta\mathbf{p}_l\mathbf{p}_{r(l)}\mathbf{p}_{r(l)+1}$ (Fig. 4.2 *iii)*). If $\mathbf{p}$ lies on the line segment between two non-adjacent points $\mathbf{p}_l$ and $\mathbf{p}_r$ choose either the triangle $\Delta\mathbf{p}_l\mathbf{p}_{r-1}\mathbf{p}_r$ or $\Delta\mathbf{p}_l\mathbf{p}_r\mathbf{p}_{r+1}$. Let $\delta_1, \delta_2, \delta_3$ denote the barycentric coordinates of $\mathbf{p}$ with respect to this triangle, i.e.,

$$\mathbf{p} = \delta_1\mathbf{p}_l + \delta_2\mathbf{p}_{r(l)} + \delta_3\mathbf{p}_{r(l)+1}.$$

We attribute each δ_r to its vertex by introducing μ_{kl}, $k = 1, \ldots, d_i$, and setting $\mu_{l,l} = \delta_1$, $\mu_{r(l),l} = \delta_2$, $\mu_{r(l)+1,l} = \delta_3$, and $\mu_{kl} = 0$ otherwise. Then for each l we find

$$\mathbf{p} = \sum_{k=1}^{d_i} \mu_{kl}\mathbf{p}_k \quad \text{and} \quad \sum_{k=1}^{d_i} \mu_{kl} = 1.$$

Summing this over all neighbors l, we obtain

$$\lambda_{ij_k} = \frac{1}{d_i} \sum_{l}^{d_i} \mu_{kl}, \quad k = 1, \ldots, d_i,$$

which fulfills the requirements (4.5). For a proof for this, as well as for a proof for the fact that these λ_{ij_k} yield a parametrization that is in fact shape-preserving, we refer the reader to [30].

4.3 Constructing a Parametrization while Simplifying a Surface

The following sections are of primarily technical nature. We describe our algorithm for constructing the parametrization of a high-resolution surface of a coarse copy of itself and discuss some implementational aspects. Following the strategy used by Guskov *et al.* [36], we divide the task into two parts: we first simplify the surface maintaining a valid parametrization at each moment. This approach, which is identical to the one described by Lee *et al.* [52], yields a preliminary surface parametrization. We then try to apply any of the parametrization types that we have described above for the planar case to the general surface. In Section 4.4, finally, we describe how the different choices affect the final result, and how parametrization quality can be measured.

The implementation has been done in C++ as part of the modular visualization system Amira [5]. This gave access to a wide range of base classes, utility classes and prefabricated visualization tools. The main contributions of this thesis to Amira are a data class for parametrized triangulated surfaces, a tool for their visualization, and modules that create those parametrizations by simplifying standard triangulated surfaces and apply different types of parametrizations.

4.3.1 Data Structures

In this section we describe the data structures used to handle parametrized surfaces. As we're working with the object-oriented programming language C++, data types are represented as *classes*. A class is a collection of *data elements* which can be of different *data types*. Together with the data elements, a class contains a list of operations, which can be applied to the data elements. These operations are called *member functions*. Instances of classes, which are called *objects*, can only be accessed through well-defined interfaces [53]. This implements the concepts of *abstraction* and *data-encapsulation*, which are at the base of object-oriented programming. Classes and objects of classes should not be confused. We nevertheless notate both class names and object names in Sans Serif Font, unless it is unclear from the context which kind of entity is designated.

A parametrized surface is a triangulated surface, (the *base grid* or *domain grid*), with each triangle implementing a piecewise linear function. We constructed our base grid class as a regular pointer-based surface data structure, i.e., three doubly-linked lists of

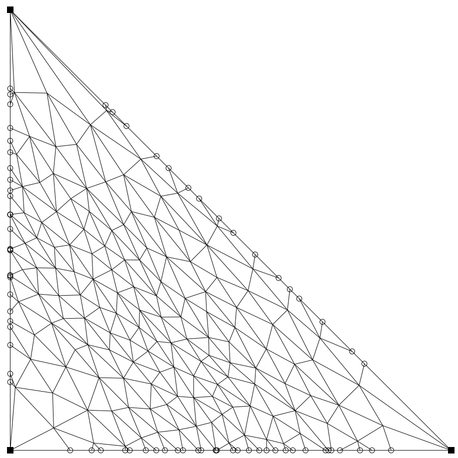

Figure 4.3: *Example of a* DomainTriangle *carrying a plane graph representing a piecewise linear function. In this and all subsequent figures, interior nodes are not marked at all, corner nodes are marked as little filled squares, intersection nodes as little circles and touching nodes as triangles. Touching nodes, which are rare, do not occur in this figure at all.*

vertices, edges, and triangles, respectively. Vertices, edges, and triangles are instances of the classes DomainVertex, DomainEdge, and DomainTriangle, again respectively. Only the vertices contain their coordinates in space, but all elements contain pointers to their neighboring elements. The reason why we have chosen to work with pointer-based surfaces rather than with index-based ones even though they use more memory (especially on 64-bit machines), is that it is a more 'natural' approach in an object-oriented language. That way, most data manipulations can be implemented more elegantly, which makes the code more readable and easier to maintain.

The main building block of the base grid surface is the class DomainTriangle. Its functionality is divided into two parts. First of all, it plays the role of a surface element by keeping pointers to the neighboring vertices and edges. In addition to this topological information, each triangle is carrier of the parametrization of a part of the original surface. That is, the class DomainTriangle has a method map() that implements the parametrization function ϕ. For each point on the triangle (given in barycentric coordinates), map() returns the unique position of the corresponding point of the high-resolution surface in $\mathbb{R}^3$. The data structures necessary to implement ϕ are encapsulated in a base class of DomainTriangle called PlaneParam.

In terms of differential geometry, each DomainTriangle T represents a coordinate patch on the domain surface S. The image of ϕ on this patch is a part of the high-resolution surface S'. Since S' consists entirely of triangles, ϕ restricted on T is a piecewise linear function. It can therefore be implemented as a plane graph G on T. This plane graph is the preimage of the edge graph of that part of S' that $\phi|_T$ parametrizes. Since S' is a *triangulated surface*, the faces of G are triangles or parts of triangles, i.e., polygons. Before evaluating the parametrization function ϕ, we triangulate each of these polygons.

An evaluation of ϕ at any point on T then consists of a point location query in G. This query yields the vertices of a triangular face of G. We compute the barycentric coordinates of the query point with respect to the three vertices of that face and obtain the function value by linear interpolation. For the point location, we use the randomized variant of the algorithm proposed by Brown and Faigle [16]. This algorithm finds the correct triangle by starting at an arbitrary edge and trying to 'walk' towards the query point. Even though its asymptotic complexity is worse than that of other known algorithms ($O(\sqrt{N})$ vs. $O(\log N)$ for the algorithm by Kirkpatrick [43]), it is preferable for actual application since it has a very low constant, it is robust to numerical errors and can easily be extended to handle triangulations with a boundary.

The plane graph is stored as an array of Nodes, with each Node containing an array of references to its neighbors. No objects that model graph edges are used. However, suitably defined *iterators* allow to write program loops just as if an array of all edges existed.

To capture the geometric information, each Node stores a 2D parameter position and a 3D image position. Finally, each Node is of one of four types: *interior nodes, corner nodes, touching nodes, or intersection nodes.* Most nodes are *interior nodes.* Those are the nodes whose parameter position is not on one of the edges of the base grid triangle. All nodes that are positioned on one of the edges are either *corner nodes*, *intersection nodes*, or *touching nodes.* Corner nodes are the nodes on the corners of the base grid triangle. For simplicity's sake, we require the corners of the DomainTriangles to be mapped to vertices of the original surface S'. The remaining two types, *intersection nodes* and *touching nodes*, come into existence when a parametrization-carrying polygon (namely the flattened star of a point to be removed) is cut into separate triangles: If a graph edge is cut in the process, the new endpoints of the two half-edges are labeled *intersection nodes.* When, while cutting, an *interior node* happens to be exactly on the cut, a copy is made for each side of the cut and each copy is labeled *touching node.* Additionally, each DomainTriangle stores three arrays keeping references to all Nodes on the three triangle edges in sequential order.

A close relative to the DomainTriangle is the DomainPolygon. Just as the DomainTriangle, it is derived from the class PlaneParam and thus implements a piecewise linear function, yet its boundary is not restricted to three vertices. It is only used as a temporary object by the simplification algorithm and is not intended to be inserted into a surface structure. Hence, it does not contain any further topological information like neighboring edges.

The three data types discussed above can be seen in a tabular view in Fig. 4.4.

DomainTriangle		
Array[0...2] of Pointers to DomainVertex	:	vertices
Array[0...2] of Pointers to DomainEdge	:	edges
Dynamic Array of Nodes	:	nodes
Array[0...2] of dynamic Arrays of integers	:	edgeNodes
Integer	:	patch
Pointers to DomainTriangles	:	pred, succ

Figure 4.4: *Overview of the* DomainTriangle *data type. The* Nodes *array is inherited from the base class* PlaneParam*.*

DomainPolygon		
Dynamic Array of Pointers to DomainVertex	:	boundaryVertices
Dynamic Array of Nodes	:	nodes
Dynamic Array of dyn. Arrays of integers	:	edgeNodes

Figure 4.5: *Overview of the* DomainPolygon *data type. The* Nodes *array is inherited from the base class* PlaneParam*.*

Node		
2D Vector	:	paramPos
3D Vector	:	globalPos
Dynamic Array of Integers	:	neighbors
Dynamic Array of Bool	:	triangular closure
Enum	:	type
Enum	:	location

Figure 4.6: *The class representing a graph node. The* triangular closure *array is actually fused into the* neighbors *array, each* bool *taking up one bit from the corresponding integer.*

4.3.2 The Surface Reduction Scheme

In this section we describe in detail the algorithmic issues of simplifying a triangulated surface on which the parametrization of a high-resolution version of itself is defined. We start with the *identity parametrization*, i.e., the base grid which is identical to the original surface, and whose triangles all bear the identity function. Obviously, the plane graph representing the identity function on a triangular domain consists of a single triangle.

Our reduction scheme is based on the point removal paradigm explained in Sec. 3.2.1. We start by associating an *error value* with each vertex of the base grid. This error value states how much the quality of the surface would lessen if we removed that vertex and retriangulated the hole. In our implementation, it is computed as a convex combination of the Hausdorff distance and the triangle aspect ratio as described in Secs. 3.3.1 and 3.3.3, respectively. In addition, depending on user choices, vertices can get blocked if their removal would introduce surface self-intersections or edges that exceed a given length. A vertex does definitely get blocked if its removal would lead to a topology change. The program is structured such that it can easily be extended to include further error criteria.

Care has to be taken when computing the errors for non-manifold vertices. Remember from Sec. 3.4 that all points that are non-manifold are either ineligible for removal or they are points on an n-fold *feature line.* We interpret each such vertex as being touched by n half-stars. This includes the case of boundary vertices ($n = 1$) and manifold vertices ($n = 2$). We then compute the error *per half-star* and take the arithmetic mean.

Unlike Lee *et al.*, who remove weighted maximally independent sets of points [52], we place the vertices in a priority queue and iteratively remove the vertex with the lowest error from the surface. We implemented the queue as a simple binary heap (see [20]). As is well known, such a heap can be built in linear time. Heaps provide the operations extractMin(), which yields the element with the lowest error and removes it afterwards, and reposition(), which assigns a new error value to an element of the heap. Both operations have a worst-case complexity of $O(\log N)$.

After having decided on a vertex v to remove, the first thing the algorithm does is to check whether the removal of v would lead to a topology change. This is done as has been described in Sec. 3.5. The test cannot be performed in advance, because the error value for each vertex is updated only when a vertex in its link gets removed. However, a constellation that makes a topology change possible can get introduced by removing a vertex that is topologically farther away from v than its link. It is therefore necessary to do the topology check right before the removal of v. If v is found to induce a topology change, it is marked as *blocked* and reentered into the queue. The simplification algorithm terminates if either the requested number of vertices has been removed from the surfaces or if the result of extractMin() is *blocked.*

We will now describe how a manifold vertex v is removed from the surface. Remember that since v is manifold, its link forms a simple cycle. Therefore, the straight line geometric realization of its star $\operatorname{St} v$ is homeomorphic to the unit disk in $\mathbb{R}^2$. As has been described, such a vertex v is deleted by removing its star from the surface. We then map $\operatorname{St} v$ into a plane polygon using the *polar map* and retriangulate that polygon

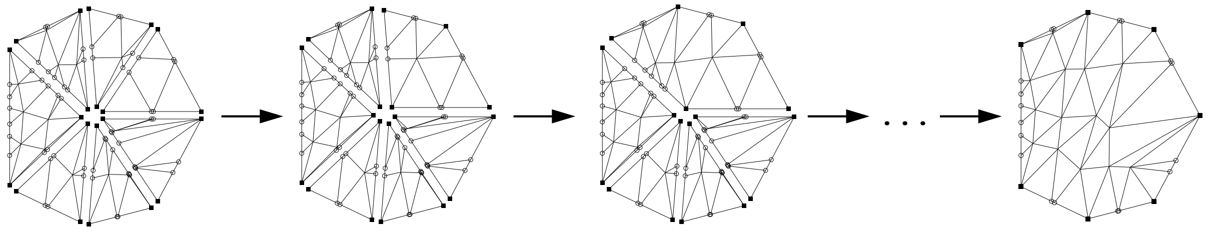

Figure 4.7: *A* DomainPolygon *is assembled by pasting together several* DomainTriangles*. Edges that were distributed across two or more* DomainTriangles *are merged and replaced by straight lines.*

with a *constrained Delaunay triangulation*. During that process, we have to make sure that the parametrization defined on the removed triangles is correctly transferred to the new triangles that replace them. For that, we use a simple approach proposed in Lee *et al.* [52]. When flattening the star of the current vertex into the 2D plane, we flatten the plane graph describing the parametrization along with it, and the retriangulation of that plane polygon will show exactly which parts of the plane graph will go with which new triangle.

In practice, we use the data type DomainPolygon, which is similar to a DomainTriangle, except that it is not restricted to three vertices and does not have the facilities to be entered into a surface (refer to Sec. 4.3.1 for details). After having computed the plane coordinates of all triangles incident to v, we merge them together one by one to form one star-shaped DomainPolygon that carries the same parametrization as the set of triangles did. We initialize the DomainPolygon with one triangle from $\mathrm{St}\, v$. We then loop around the star and merge the DomainPolygon with an adjacent triangle in each iteration. This process, which is straightforward to understand, yet somewhat finicky to implement, is demonstrated in Fig. 4.7.

The complete polygon is then retriangulated. For this, we use a constrained Delaunay triangulation. If four vertices lay on a circle, such that the Delaunay triangulation is not unique anymore, we try both possible diagonal edges and choose the one that yields triangles with a lower aspect ratio when projected back into the surface. The output of the routine is a set of edges in the polygon. These edges are then used to recut the DomainPolygon into new DomainTriangles which can be reinserted into the base grid. The process is visualized in Fig. 4.8.

Two main technical problems appear when implementing this method. Firstly, since the polar map used to flatten out the star of a vertex v is *piecewise linear harmonic*, the parametrization function ϕ from the parameter domain into $\mathbb{R}^3$ might become multi-valued. Differently put, the plane graph on the DomainPolygon might contain self-overlappings. One possible remedy to this is to apply the barycentric parametrization (Sec. 4.2.2) to the DomainPolygon after it has been assembled. This requires solving a

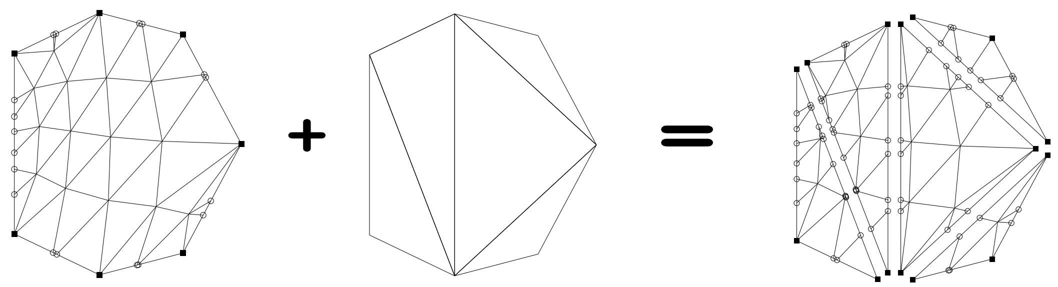

Figure 4.8: *A* DomainPolygon *is cut along a given triangulation*

sparse system of linear equations and thus has a running time quadratic in the number of graph vertices. One can save time by testing each polygon for overlaps first (which takes only linear time), and only applying the parametrization when necessary.

Secondly, the plane polygon given by the polar map routine is not necessarily convex. It therefore happens in rare cases that the plane graph is not entirely contained within the prescribed boundary polygon. This makes cutting the DomainPolygon into DomainTriangles impossible. In order to ensure a stable algorithm, we do not use the boundary polygon determined by the polar map in those cases, but map the star of v to the unit circle instead.

Handling feature line vertices is equally straightforward. We interpret a *vertex on a feature line* as being touched by a number of half-stars. Handling each of those half-stars separately, we can therefore follow the procedure laid out above. In detail, we iteratively remove each of the half-stars. After being removed, we map each of them to the plane using the polar map. However, we restrict the resulting polygon to the positive half-plane by choosing $a = \pi/\Theta$ in Equation (3.1). As in the case of a regular vertex, we assemble the flat DomainTriangles into one DomainPolygon and use a CDT to obtain a new triangulation without the old center vertex. Once this is done for all half-stars, one is left with a topologically equivalent surface equipped with a consistent parametrization where the two feature edges that touched the removed vertex are now replaced by a single edge of the same order.

One final step has to be taken before being able to use the DomainTriangles as piecewise linear maps. The Brown-Faigle algorithm [16] for point location needed for the evaluation of the parametrization function ϕ requires the plane graph to be a *triangulation* of the parameter domain. However, through the repeated merging and recutting of neighboring domains, this requirement is not entirely met any more. Preimages of triangles of the original surfaces might be distributed over several adjacent DomainTriangles, which is equivalent to saying that the images of the edges of the base grid under the parametrization do not follow edges in the high-resolution surface but cut across triangles. This leaves the border area of the DomainTriangles littered with quadrilaterals, stumps of triangles whose tip is mapped somewhere else. To remedy this problem, after having simplified the surface, we apply the *triangular closure*, i.e., the set of additional edges needed to turn our plane graph into a true triangulation. Obviously, if further simplification is needed, the triangular closure should be removed in order to avoid the proliferation of vertices and triangles that do not carry any geometric information. For every adjacency reference that a Node stores, it keeps an additional flag. This flag shows whether the adjacency, i.e., the edge it stands for, is a true part of the parametrization or whether is has only been added as part of the triangular closure. Thus, the triangular closure can be removed easily by deleting all adjacency entries with the flag set. Also, remember that adjacency references are actually Integers pointing into the array of Nodes. We found it reasonable to expect the number of Nodes per DomainTriangle not to exceed $2^{31} - 1 \approx$ two billions. Therefore, we used the 32nd bit to keep the flag, thus preventing additional memory requirements.

4.4 Measuring and Visualizing the Distortion of a Parametrization

As is clear from the discussion in Section 4.2, there is a large number of ways of mapping a piecewise linear surface patch to a convex polygon. Harmonic, barycentric, or shape-preserving mappings are the most common ones, but any intersection-free embedding of the edge graph of the surface into a convex planar region does in fact induce an eligible mapping.

It should come as no surprise that not all of those different mappings are equally suitable to our purposes. The demands on triangle quality discussed in Section 2.6 not only apply to surfaces obtained by simplification, but also to triangles that are created using *refinement* strategies. While the subject of mesh refinement will be spoken about in more detail in Chapter 5, we will here try to develop a heuristic that measures the *quality* of a given parametrization. Visualizing and analyzing this measure will then allow us to compare different parametrization strategies and to decide upon their suitability for finite element mesh generation purposes.

As the measure of quality here we use the distortion that each triangle undergoes while being mapped by the projection function ϕ^{-1} onto the parameter plane. This measure is of course a local one, which allows also to detect problems that arise due to specific geometric features of the surface such as peaks or creases.

We define the distortion of a triangle Δ_1 in three-space being mapped to another triangle Δ_2 in the plane as follows: Consider both Δ_1 and Δ_2 as triangles in $\mathbb{R}^2$. Then there is an affine function

$$f : \mathbb{R}^2 \to \mathbb{R}^2$$

that maps Δ_1 onto Δ_2. The linear part of f can be represented by a matrix $A \in M^{2\times 2}$, where $M^{2\times 2}$ is the space of all 2×2 matrices. Then, as a measure of 'distortion', we use the relative condition number κ_A of A. For a linear mapping described by a matrix A,

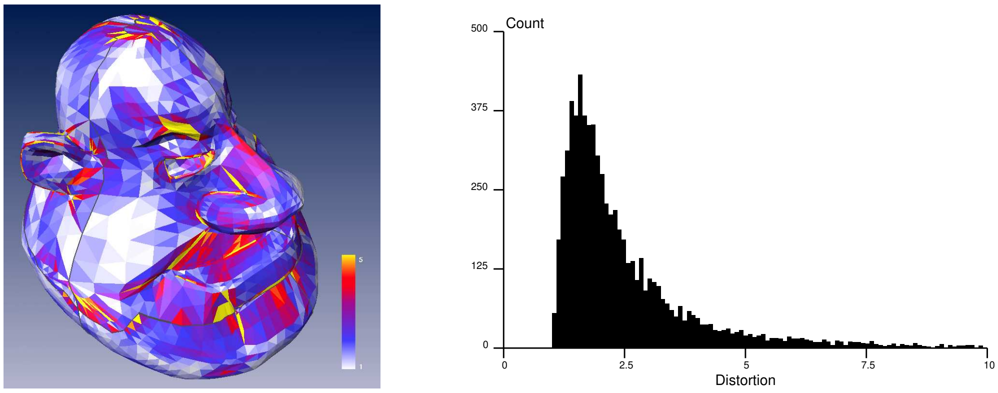

Figure 4.9: *Visualizing the quality of the barycentric parametrization using the triangle condition number κ_A (see the text). On the left, a color encoding on the triangles. The whiter the triangle, the lesser it is distorted. On the right, the distribution of the distortion values in histogram form.*

the relative condition number κ_A is defined as [22]

$$\kappa_A = \|A\|_2 \|A^{-1}\|_2.$$

The condition number κ_A can be computed easily for a 2×2 matrix by computing the two singular values $\sigma_1 \geq \sigma_2$ and using

$$\kappa_A = \frac{\sigma_1}{\sigma_2}.$$

To see that our definition is reasonable, note that the matrix condition is a real number greater than or equal to one. It is exactly one for the identity mapping as well as for all orthonal mappings, i.e., rotations and reflections, which are both mappings that do not distort a triangle. It has also the value of one for all matrices that represent a pure scaling, which is still reasonable. At the other end of the scale, a matrix of rank $r < 2$, i.e., one that maps a triangle to a line, or even a point, will yield a condition number $\kappa = \infty$. Consider also that $\kappa_A = \kappa_{A^{-1}}$.

We have used the condition number κ_A as a measure to compare the behaviour of the barycentric mapping and Floater's shape-preserving mapping from Sec. 4.2. We took a surface representation of a cartoon figure head and manually partitioned the surface into eight triangular patches. Correspondingly, we constructed a base grid consisting of eight triangles forming an octahedron. We then used the barycentric and shape-preserving parametrizations respectively to map each triangular surface patch to its corresponding base grid triangle. The results can be seen in Fig. 4.9 and 4.10. The distortion is color-encoded on each triangle using a color-map ranging from white over blue and red to yellow. It is quite obvious that using Floater's mapping yields a parametrization that is far less distorted than the barycentric approach. This is not surprising since the former uses geometric information which the latter totally neglects. In fact, the barycentric

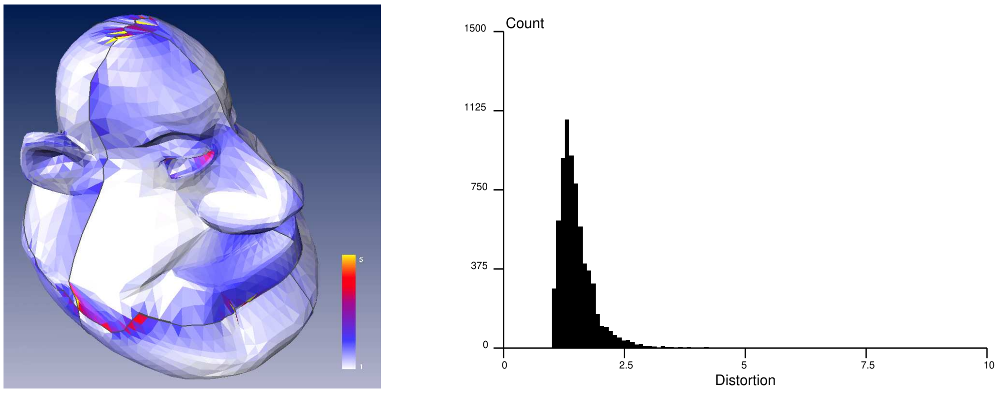

Figure 4.10: *The same as Fig. 4.10 for Floater's shape-preserving parametrization.*

parametrization is so inferior to Floater's approach that we have not seen any reason for providing the possibility to use the barycentric mapping in the final software package.

4.5 Applying a Parametrization Globally to a Non-Manifold Surface

Consider again Fig. 4.10 which shows the cartoon figure head parametrized over an octahedron using the shape-preserving parametrization on each base domain triangle. The parametrization quality is high in the interior of each base grid triangle. However, near the edges and, in particular, near the vertices of the base domain, large values for the triangle condition number occur. This comes as no surprise. It has been explained in Sec. 4.2.3 that Floater's algorithm for surface parametrizations tries to embed the edge graph of the original surface into a given parameter domain, using a mapping that is constructed to be 'as affine as possible'. It is, however, hindered by the imposed boundary conditions. That is to say, intuitively speaking, Floater's algorithms tries to place each graph node on the parameter domain such as to minimize distortion. Near the boundaries it can't however, because the positions of the boundary nodes are fixed, and those posititions might not be optimal.

This leads to the obvious conclusion that the overall quality of the parametrization of a non-trivial surface can be improved by reducing the number of boundary nodes. The simplest way to do this is to use fewer DomainTriangles, i.e., a coarser base grid. Unfortunately, there's a lower limit to this, as every surface needs a base grid of at least a certain number of base grid triangles, depending on the topological type of the surface.

4.5.1 Relaxation across Edges

In order to improve the quality of a given parametrization, we propose an iterative solution strategy that we will explain in detail in this and in the following section. Roughly speaking, we group several base grid triangles together to form one large parameter do-

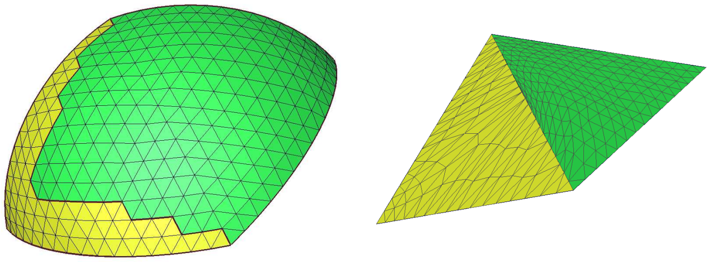

Figure 4.11: *A part of a surface that is to be parametrized over a base grid consisting of only two triangles. The run of the dividing curve is very badly chosen (left). As a result, the most part of the surface is mapped to the right triangle, leading to high distortion (right).*

main. We then apply Floater's scheme on the large domain and recut it afterwards. The implementationally simpler variant, which we call *relaxation across edges*, combines two triangles to form a quadrangle. It will be described in this section. In the subsequent section, we combine the whole star of a base grid vertex.

Consider two adjacent base grid triangles carrying each a part of a high-resolution surface, as for example in Fig. 4.11, right. It can be seen clearly from both parts of Fig. 4.11 that the partition of the original surface onto the two triangles has been done in a rather unfortunate way. Even though the vertices of the image surface are quite evenly space, the vast majority of them have their preimages on the right base grid triangle. This leads to high distortion values. We remedy this in the following way: We take the two triangles and 'fold them up' along their common edge so as to lie in the same plane. That is to say, we construct an isometry that maps the two triangles onto a plane quadrangle. In most cases, this quadrangle is convex. If it is not, we move the vertex at the inward corner just enough to ensure convexity. We then use the merging procedure described in Sec. 4.3.2 to turn the two triangles into a single DomainPolygon. After having applied the shape-preserving parametrization on this polygon, we cut it into two triangles, again using the algorithms from Sec. 4.3.2. The steps of merging and recutting are visualized in Fig. 4.12.

The result can be illustrated in two different ways: looking at the plane graph on the base grid triangles, we can see that it has moved. With the dividing edge gone, the graph nodes, which had been predominantly on the right base grid triangle, could move freely and are now evenly distributed. The parametrization has *relaxed across the edge*. On the other hand, it is also instructive to regard the image of the base grid edge that has been removed and reinserted. It has been significantly straightened out, which is why the procedure is also called *edge relaxation*.

Applying this edge relaxation iteratively to all edges of the base grid lets the parametrization converge to a stable limit with substantially improved quality, as can be seen in Fig. 4.14.

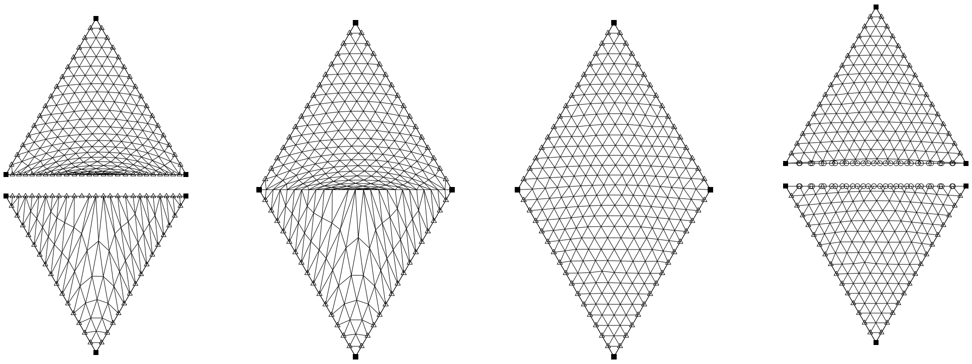

Figure 4.12: *After the edge relaxation procedure, the plane graph representing the parametrization function ϕ is stretched out evenly across both triangles. The triangulations on the two triangles on the far left actually do match.*

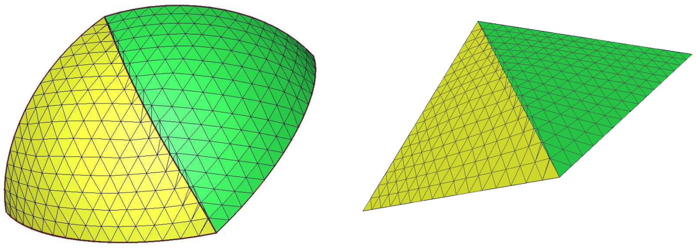

Figure 4.13: *After the edge relaxation procedure, the image of the dividing base grid edge is straighter, and the parametrization is more uniform.*

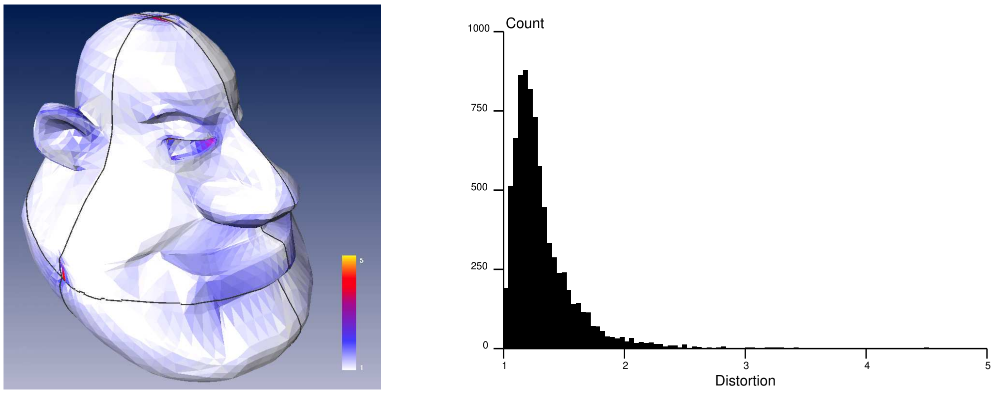

Figure 4.14: *The same model as in Sec. 4.5.1 after three edge relaxation iterations. Note the different scale of the horizontal axis of the histogram.*

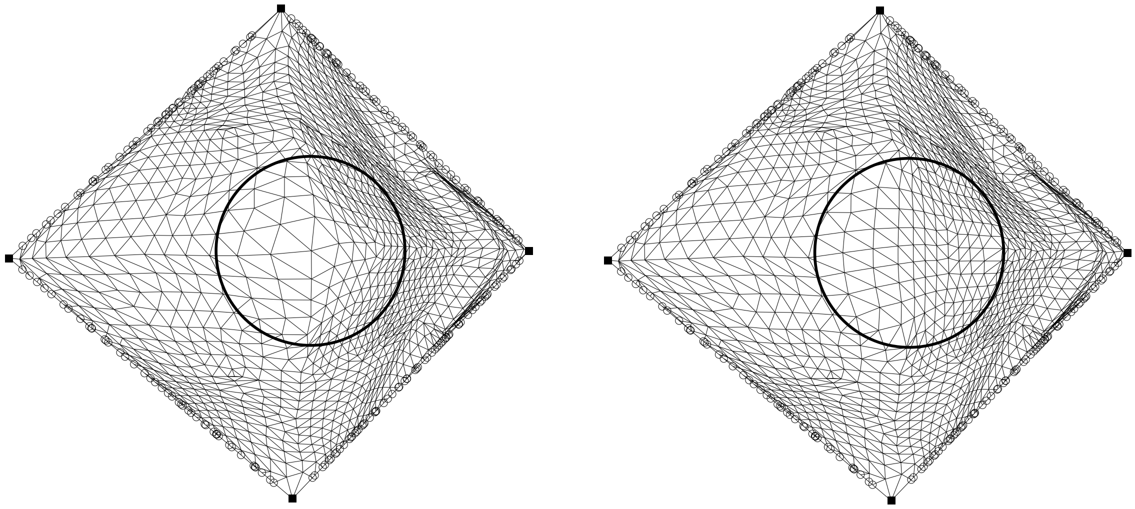

Figure 4.15: *The parametrization is smoothed only in a circle around the origin.*

4.5.2 Relaxation across Vertices

The relaxation of edges described in the preceding section leads to a vastly improved quality of the parametrization. However, regions of high distortion remain around the vertices of the base grid. It seems natural to try to relax the parametrization on more than two triangles at once. Following an idea of Guskov *et al.* [36], we have implemented an algorithm that allows relaxation across the star of a vertex. For this, we simply take all triangles that border a given vertex flatten them with the polar map and merge them into a single polygon, exactly as for the surface simplification in Sec. 4.3.2. We then apply Floater's parametrization to the whole polygon, thus relaxing all involved triangles at once. Finally, we choose the graph node that is now closest to the origin of the parameter plane as being the new central base grid vertex and recut the polygon accordingly. Since this node is generally different from the original center node, the base grid vertex around which we have smoothed has changed as well.

However, this doesn't quite work. As is actually being hinted at in [36], the base grid vertices move far too much, leading to a base grid that may be heavily distorted. The solution is to restrict the liberty of action of the center of the polygon a bit. Since we have just noted that the worst distortion occurs near the base grid vertices, hence near the center of our polygon, it should already help to only relax the parametrization near that center. More technically, after merging the triangles into one polygon, we proceed as follows: We define a circle of user-controllable radius and place it in the coordinate origin. We then mark each node that is outside of this circle as *boundary*, and compute Floater's matrix for this constellation (Fig. 4.15). Again, the node then being closest to the center is being chosen as the new base grid vertex. That way, the amount of movement the graph makes is limited by the circle radius, since movement can only take

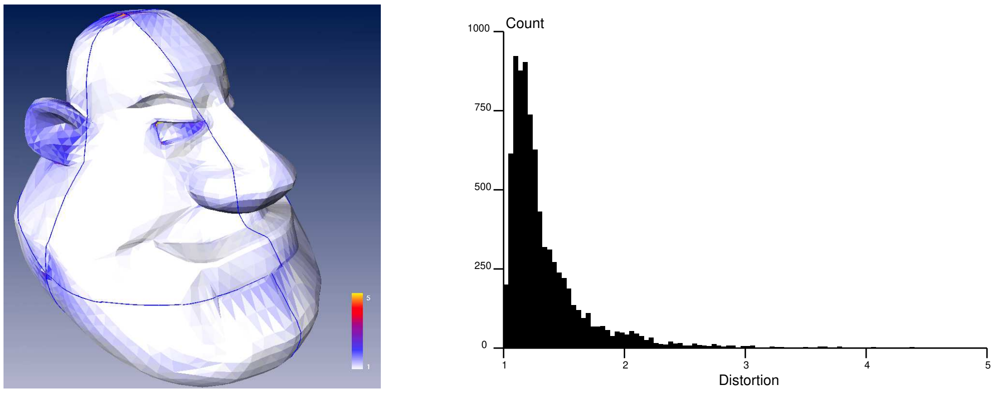

Figure 4.16: *Vertex Relaxation doesn't seem to lead to a relevant improvement. Compare this figure to the results of the simple edge relaxation (Fig. 4.14).*

place inside that circle.

While this refined method leaves the base grid in good shape, the overall results are not truely satisfying (Fig. 4.16). We have tried the approach on several models alone and in conjunction with the edge relaxation, and we have found that when it is applied by itself, the parametrization quality actually decreases. When used together with edge relaxation, the resulting parametrization is of similar quality as a parametrization smoothed by using edge relaxation alone. For comparison with our results of the last section, we show here again the cartoon figure head. After having relaxed its edges, we have applied the vertex relaxing scheme with a circle radius of 0.3 times the length of the shortest edge incident to each vertex. This value yielded the best results. After that step, edge relaxation was necessary again, and the final result was comparable to that obtained without vertex relaxation in Sec. 4.5.1.

5 Mesh Refinement

When doing finite element computations, it is possible (and general practice) to estimate the local discretization errors. This *error estimation* is possible without knowledge of the exact solution, and is usually done by comparing the results of two approximation schemes of different asymptotic quality. When errors occur that exceed a prescribed upper bound, the grid has to be refined in the region that contains the error, and the computation has to be redone at least locally. In this chapter we'll describe the refinement process in some detail and show how parametrized surfaces can improve the quality of the numerical results.

5.1 Refinement of Tetrahedra

Refinement of tetrahedral grids is usually an iterative procedure. It is therefore appropriate to regard a whole sequence $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_k, \ldots$ of triangulations which have to fulfill the following conditions:

- **Nestedness :** Each element $T \in \mathcal{T}_k, k > 0$ is contained in exactly one element $T' \in \mathcal{T}_{k-1}$ and any corner of T is either a corner or an edge point of T'. This will be partially relaxed near boundaries later on when working with parametrized surfaces.

- **Consistency :** The intersection of any two tetrahedra in $\mathcal{T}_k$ is either empty, a common face, a common edge, or a common corner.

- **Stability :** The sequence $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_k, \ldots$ is stable in the sense that the element aspect ratios of all tetrahedra are bounded from above by a constant.

In order to ensure those three criteria, the standard procedure, which is called *regular refinement*, is to split a father tetrahedron into eight sons as follows: First, bisect each of the four edges and decompose the tetrahedron into four smaller tetrahedra and an octahedron as shown in Fig. 5.1. The four tetrahedra are congruent to their father tetrahedron. Then, dissect the octahedron into four tetrahedron as in Fig. 5.2. Unfortunately, this last dissection is not unique, because there are three ways to choose the diagonal of the octahedron (dotted in the figure). Each choice of the diagonal provides a regular refinement of the given tetrahedron. There are examples showing that successive regular refinement may be unstable in the sense given above if the diagonal interior edge is not chosen properly [79]. However, a strategy proposed by Bey [11] gives rules for

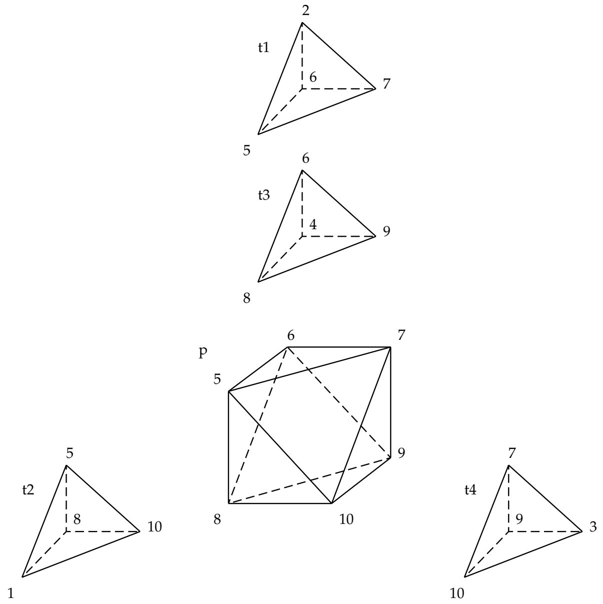

Figure 5.1: *A tetrahedron is split into four smaller tetrahedra and an octahedron. Corresponding numbers point out corresponding points. This and the following three figures are taken from the article of Bornemann, Erdmann and Kornhuber [14] and are used with the kind permission of the authors.*

picking the diagonal such that sequences of regular refinements are guaranteed to be stable.

Subjecting a subset of the tetrahedra to this refinement procedure introduces *hanging nodes* on the interface between refined and nonrefined cells, which are a violation of the consistency condition. In order to avoid this, one applies special dissection schemes, called the *green closure*, which have been introduced by Bank *et al.* in [7] for the planar case. Generalizations to three dimensions can be found in Bornemann *et al.* [14] (see Fig. 5.3). The newly inserted *irregular tetrahedra* have an aspect ratio worse than those produced by the regular refinement pattern, however they are removed before and reapplied after each new refinement step. Thus, the overall element aspect ratio remains bounded.

5.2 Boundary and Interface Surfaces

The local refinement of tetrahedra induces a local refinement of the triangles that form the boundary and interface surfaces. It is identical to the *red/green refinement* introduced by Bank *et al.* in [7]. There, the main operation is the *regular*, or *red* refinement, which splits a triangle into four congruent ones by bisecting each triangle edge. After a set of triangles has been refined, triangles that border two bisected edges are themselves refined regularly until all triangles are either refined or touch at most one bisected edge.

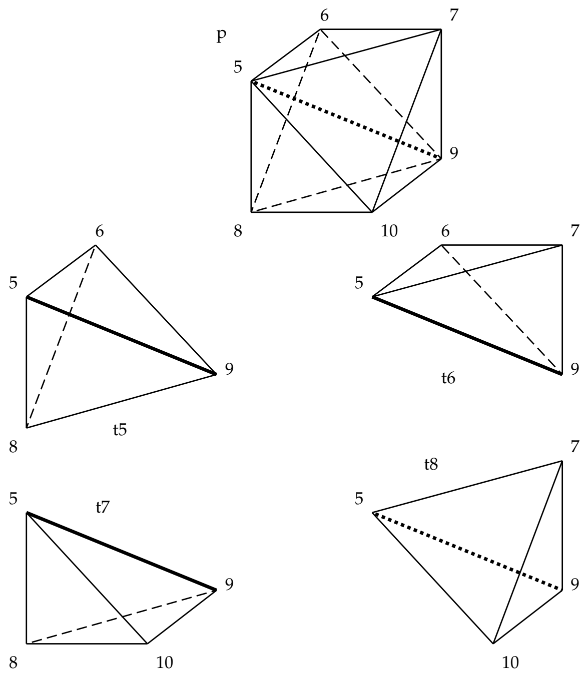

Figure 5.2: *There are three ways to split an octahedron into four tetrahedra. They differ in the way the diagonal is chosen.*

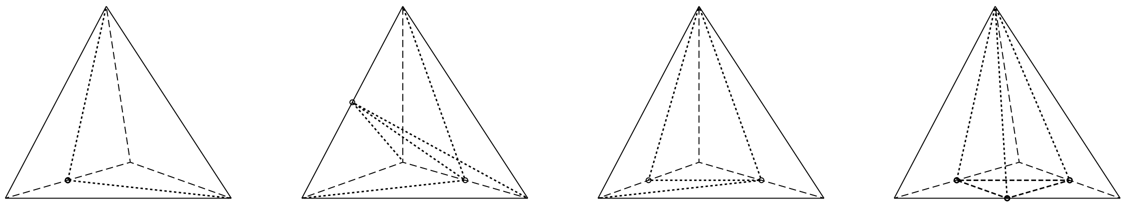

Figure 5.3: *The* green closure *in three dimensions.*

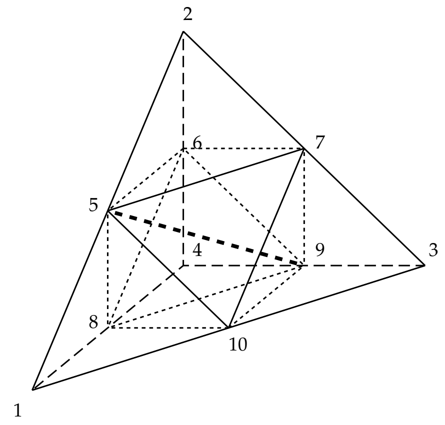

Figure 5.4: *Regular refinement of tetrahedra that touch boundary or material interface surfaces entail a regular refinement of the surface triangles.*

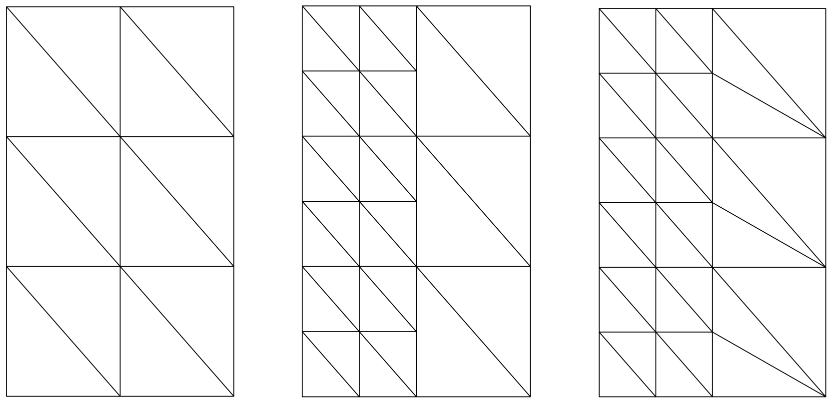

Figure 5.5: *The planar red/green refinement scheme. Left: a uniform planar grid. Center: the left column of triangles has been refined regularly. Right: The irregular (green) closure.*

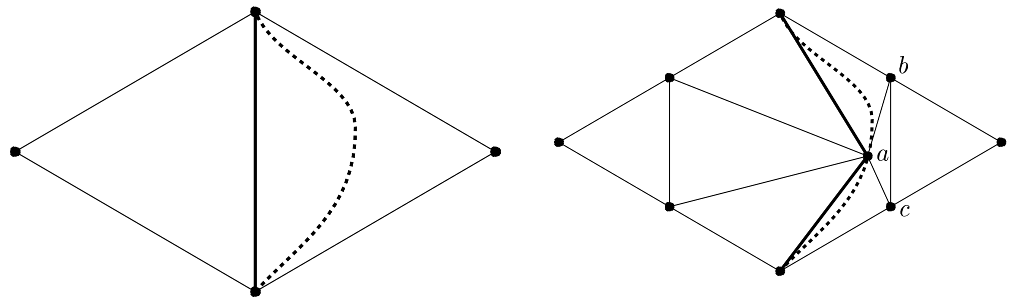

Figure 5.6: *When refining grids using parametrized interface surfaces, nearby interior nodes have to be repositioned in order to avoid low quality cells.*

In a last step, those triangles that do border a single bisected edge are split using an *irregular* or *green* refinement. This green refinement again creates triangles with a higher aspect ratio than their father triangles. Therefore, if more refinement is requested later on, all triangles that have been created by the irregular scheme are replaced by their father triangles before again applying the red refinement as needed.

Grid refinement is definitely advantageous when the computational domain is triangulated too coarsely to allow a reasonably sampling of the solution function. However, near the boundaries, the standard scheme is not satisfying. That is because successive refinement doesn't introduce new *geometrical* information into the domain. Many domains that are used for FEM computation are known in far more detail than what is actually used. Remember that, for example, grids used to plan hyperthermia treatment are generated by using boundary surfaces that have to be simplified by a considerable factor before they become manageable for mesh generation. However, standard mesh refinement doesn't take this extra information into account at all. No matter how often a coarse domain grid triangle is refined, geometrically speaking it will always remain a coarse triangle.

Parametrized surfaces are a solution to this problem. Every time a high-resolution boundary surface is available, using the algorithms described in Chapters 3 and 4, it can be turned into a coarse surface that still 'remembers' its original shape. This surface can be used as the input of a mesh generator. However, when doing grid refinement at the boundary or interface surface, edges are not simply bisected. Instead, one uses the fact that the original surface is available as the image of the parametrization function ϕ defined on the base grid. Rather than placing a new vertex on the midpoint of an edge, one queries the function ϕ at that point. The function value is a point on the original surface. Thus, when iteratively refining the surface, it will converge to its high-resolution ancestor.

Even though the advantages of using parametrized surfaces for mesh refinement are quite evident, there are a few problems that need to be handled in order to be able to get the full benefit of the technique. For example, the grid generator (i.e., the programs that creates and controls all new vertices) has to be aware that interface

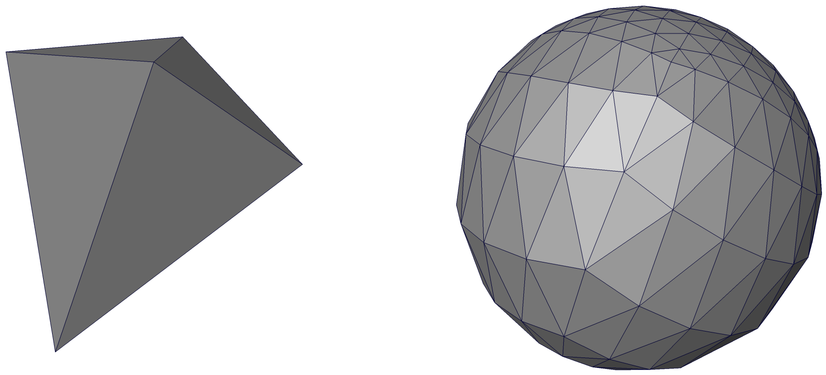

Figure 5.7: *Regular edge bisection may lead to refined meshes with widely varying triangle sizes across domain triangle boundaries.*

surfaces can move when they are refined and has to appropriately reposition interior vertices in order to avoid low quality grid elements. This is illustrated in two dimensions in Fig. 5.6. There, two triangles are separated by a parametrized interface surface. When using the parametrization to regularly refine both triangles, the new vertex a moves dangerously close to vertices b and c, yielding a triangle with a high aspect ratio. Even worse, a might actually pass beyond b and c, leading to a self-intersection of the computational grid. Note that this intersection is different from the ones considered in Sec. 3.6. There, the interface surface intersected *itself*. Here, the surface by itself is completely intersection free. However, the grid that has been modified by the regular refinement contains now self-intersections and elements with negative volume. In order to avoid these degradations, *at least* the vertices b and c have to be repositioned.

A second problem is that standard error estimating routines do not take into account that the geometry of an element can change upon refinement. Such estimators might compute a low error value for an intermediate solution that is actually quite imprecise when considered on a more accurate domain. It is therefore necessary to use special error estimators that do consider changing computational domains. Such estimators have in fact been developed (cf. [24]). Similar requirements arise in the numerical treatment of *obstacle problems*, where the domain represents a moving body that encounters obstacles and gets deformed by them. For details on error estimators that deal with this setting see [18, 76].

5.3 Loop-Smoothing

One sees by looking at an example surface that successive regular refinement of coarse grids using parametrized boundaries leads to surfaces that are visually pleasing within each base domain triangle. However, at the borders between those triangles, jumps in the triangle size occur (Fig. 5.7). These are caused by the effect that the size of the original base grid triangle has on the size of all its descendants. In order to obtain

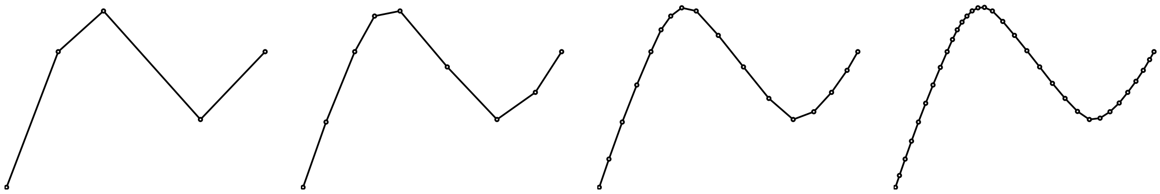

Figure 5.8: *Suitable repeated bisection makes the polyline approach a smooth surface. Each new vertex depends on the two vertices to its left and the two vertices to its right via the formula (5.1).*

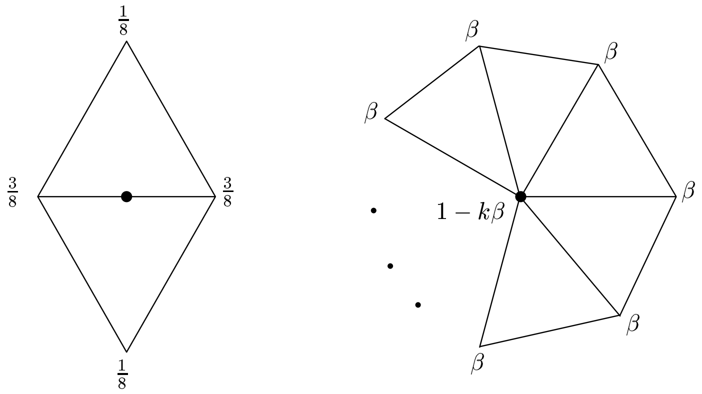

Figure 5.9: *The weights used for Loop subdivision. The choice of β is not unique. Loop himself proposed $\beta = \frac{1}{k}(\frac{5}{8} - (\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{k})^2)$.*

refined surfaces with globally continuous triangle sizes, Lee *et al.* [52] proposed a special refinement scheme, called *Loop-smoothing*, which is based on *subdivision surfaces*.

Subdivision is a method that defines smooth curves or surfaces as the limit of a sequence of refinements of polygonal curves or surfaces. An example is shown in Fig. 5.8. There, a continuous, piecewise linear curve is shown on the left. Then, in each of the following images, the curve has been refined by bisecting each segment and placing each new vertex $\mathbf{p}'_i$ at a position entirely defined by the positions of the two vertices $\mathbf{p}_{i-1}$ and $\mathbf{p}_i$ to its left, and the two vertices $\mathbf{p}_{i+1}$ and $\mathbf{p}_{i+2}$ to its right by the formula

$$\mathbf{p}'_i = -\frac{1}{16}\mathbf{p}_{i-1} + \frac{9}{16}\mathbf{p}_i + \frac{9}{16}\mathbf{p}_{i+1} - \frac{1}{16}\mathbf{p}_{i+2}. \tag{5.1}$$

After several refinement steps, the curve starts to become rather smooth. In fact, the limit curve is in $\mathcal{C}^1$. The choice of coefficients in (5.1) is not unique. Other sets of values exist. The limit curves of some of them are *spline curves* [81].

The 1978 papers of Doo and Sabin [23] and of Catmull and Clark [17] were the first to extend the concept of subdivision to surfaces. While they worked with surfaces based on quadrilaterals, Loop [56] introduced a scheme that works on triangulated surfaces and is particularly simple. His methods works by splitting edges and computing the position of the new vertices using the weights shown in Fig. 5.9, left. Additionally, after each refinement step, all the old vertices (*even* vertices in subdivision lingo) are smoothed as a weighted average of all its direct neighbors (Fig. 5.9, right), which makes this subdivision scheme an *approximating* one. The result is a sequence of surfaces whose limit is $\mathcal{C}^2$-continuous everywhere except at the descendants of the original vertices, where it is only $\mathcal{C}^1$-continuous. In recent years, a wide variety of different subdivision schemes have been developed. An extensive introduction to the subject can be found in [81].

The Loop-smoothed refinement proposed by Lee *et al.* [52] now works as follows. Instead of choosing the parameter position of a newly inserted vertex at the midpoint of a given edge, the two triangles bordering that edge are considered. The new coordinates are then determined by applying the Loop weights from Fig. 5.9, left, *in the parameter plane* to that stencil. This is straightforward if all four vertices needed to compute the new position are located on the same domain triangle. However, if they are spread out over several domain triangles, those have to be flattened out and combined into a single large parameter domain. If only two adjacent domain triangles are involved, these are just folded up along their common edge. If more than two are concerned, the domain triangles will frequently all touch a common vertex and can therefore be flattened out using the polar map described in Sec. 3.2.1. Unfortunately, not all possible constellations are covered this way, and the remaining ones have to be dealt with on a case by case basis.

After all edges have been split, the *even* vertices are smoothed, using the stencil from Fig. 5.9, right, this time. Again, this smoothing is only done in the parameter domain.

The effect of the Loop-smoothing procedure is that the influence of size differences between adjacent domain triangles is evened out a fair bit. Intuitively, if an edge between two triangles of different sizes is bisected, the new vertex will effectively be 'drawn' into the larger one, thus leading to descendant triangles of a more uniform size. Visually, Loop-smoothing yields far better meshes as is shown in [52].

A problem when using this method together with volumetric grids is that now not only don't new vertices lie on the old boundary surface anymore, but at each refinement level, *each* vertex on the boundary moves. This leads to problems of elements of bad quality and even mesh inconsistencies similar to those described in Sec. 5.2. A simple solution would be to do without this smoothing of even vertices. One could also consider using an *interpolating* subdivision scheme, such as the *butterfly method* [26], which is slightly more complex, but where even vertices remain untouched.

6 Results

6.1 Hyperthermia

We have applied our parametrization-constructing algorithm to a patient model surface generated automatically from a labeled volume data set using the *generalized marching cubes* algorithm (*GMC*). The input surface can be seen in Fig. 6.1 with the skin layer removed. As is typical for such surfaces, it has a very complex geometric structure, since it contains the boundary surfaces between all main organs in the human lower body, including muscles, fat, bones, major blood vessels, intestines and, of course, the tumor. Also, since the surface comes straight out of the GMC-programm, it is very highly resolved. The one that can be seen here consists of 217284 vertices and 444550 triangles. Simplifying such a surface while keeping the topology invariant is a daunting task for any algorithm.

We have tried to simplify this surface to about 10% of its original complexity. Choosing equal weights for the Hausdorff-distance and the aspect ratio quality criteria, we obtained simplified models carrying a valid parametrization of rather high quality (cf. Fig. 6.2). Since we demanded that no self-intersections should be introduced (and we will continue to do so throughout this chapter), the result actually is completely intersection-free. One can see however, that in flat parts like the top and bottom of the model, very large triangles occur. This is not surprising, since in those areas with little geometric variation, large triangles *do* resolve the geometry well. However, the subsequent step of building a volumetric model from these surfaces is made very complicated if its input data consists of triangles of widely varying sizes. Therefore, we have implemented another quality criterion which allows the removal of a point only if that doesn't lead to new edges that exceed a given length. Running the algorithm again and this time limiting the edge lengths to 2.5cm yields a much better output (Figs. 6.3 and 6.4).

Making use of the fact that a parametrization of the original surface was available, it was now possible to refine the surface again. Since mesh refinement is usually done by the FEM software, no adaptive mesh refinement algorithm has been implemented in AMIRA. However, for demonstration purposes, a uniform refinement is possible. After several iterations of edge smoothing, the base grid could be refined to yield a satisfying new mesh (Fig. 6.5 and 6.6). Even further improvement should be obtainable by using the Loop-smoothed refinement algorithm described in Sec. 5.3.

The above results can be only preliminary. The merit of our method can't really be judged until actual computations have been done on actual grids produced using our simplification and refinement schemes. Work is in progress to connect the parametrized

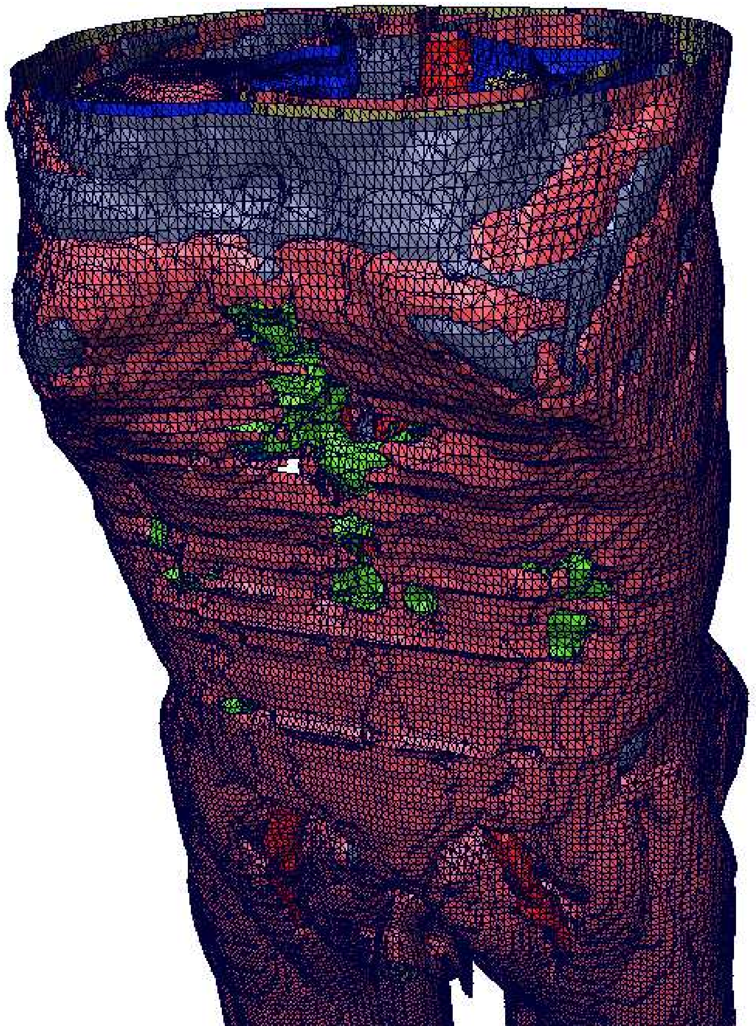

Figure 6.1: *The original interface surfaces as created by the GMC algorithm. The skin layer is not shown.*

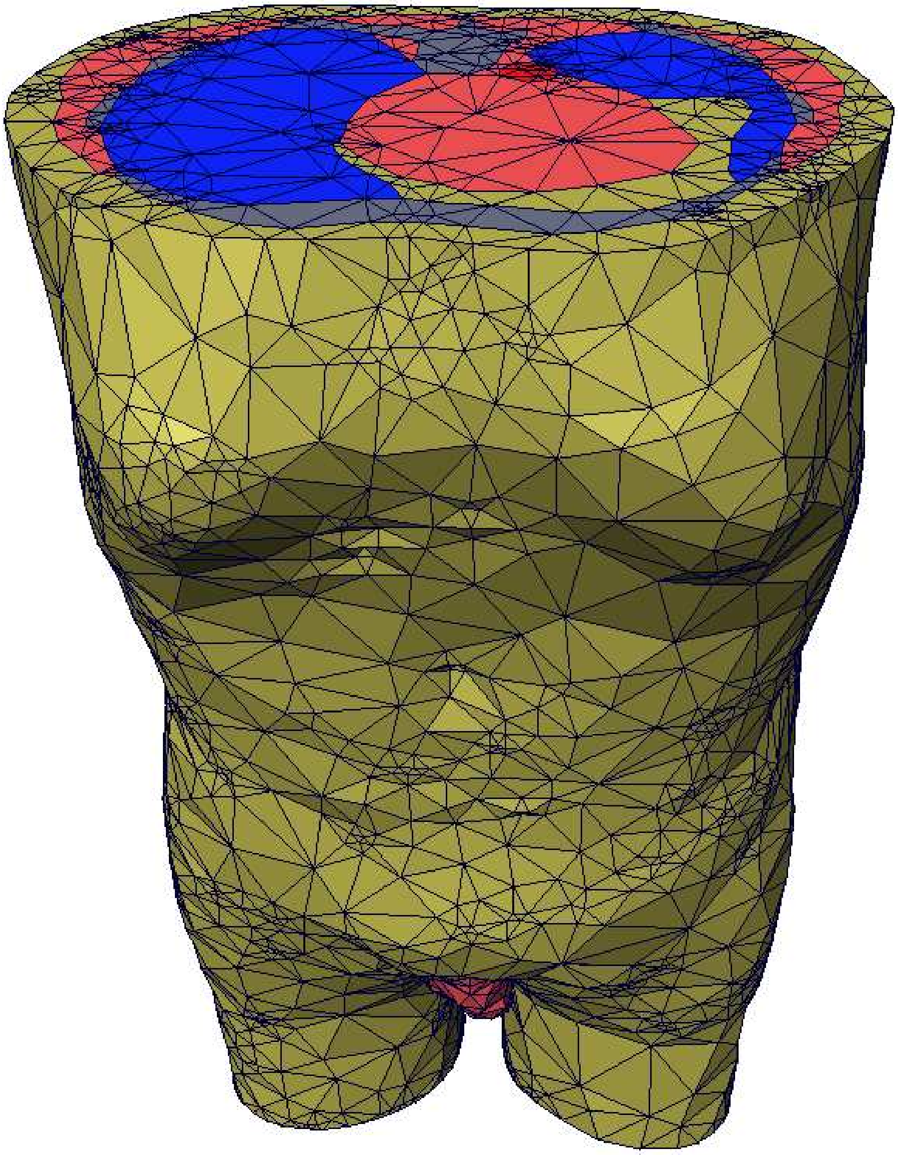

Figure 6.2: *The interface surfaces simplified without limitations on edge length.*

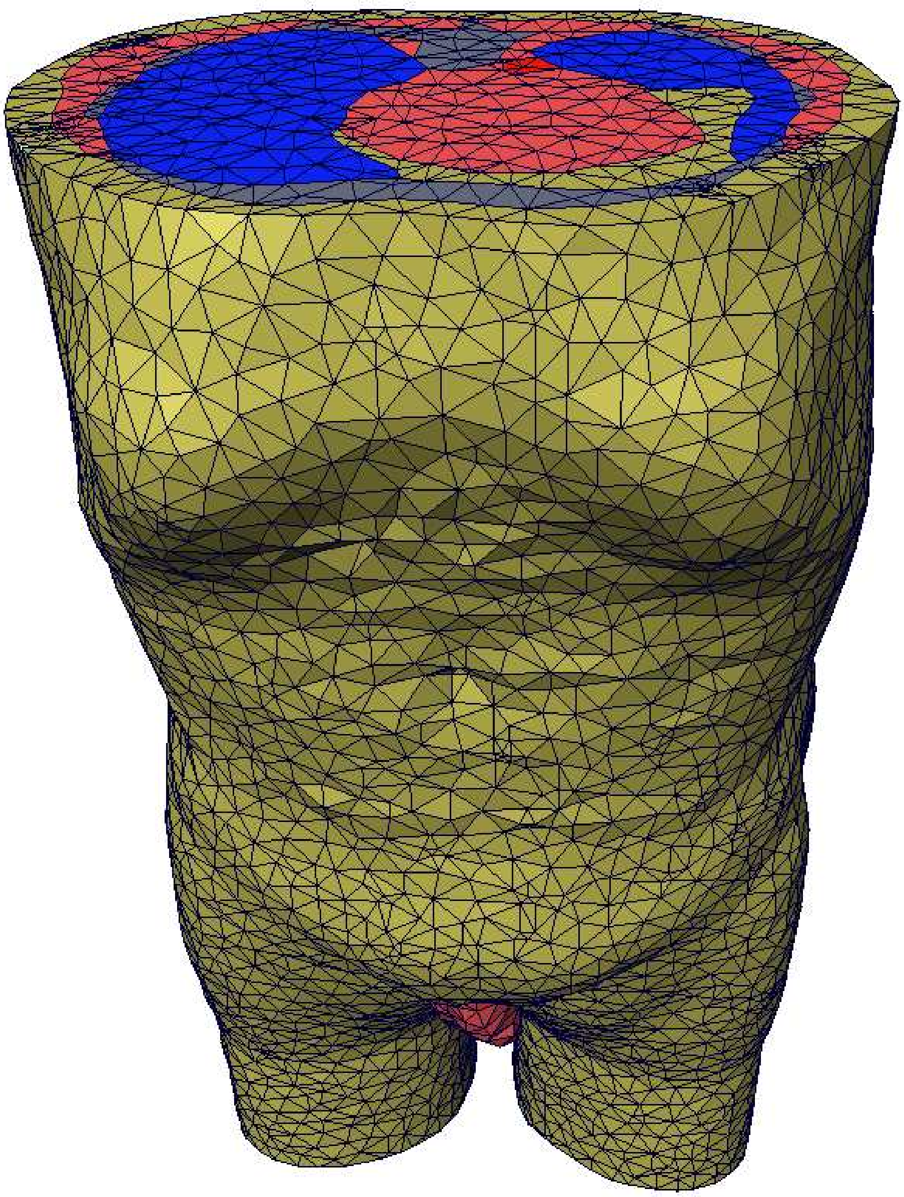

Figure 6.3: *The surfaces simplified with an edge limit of* $2.5cm$.

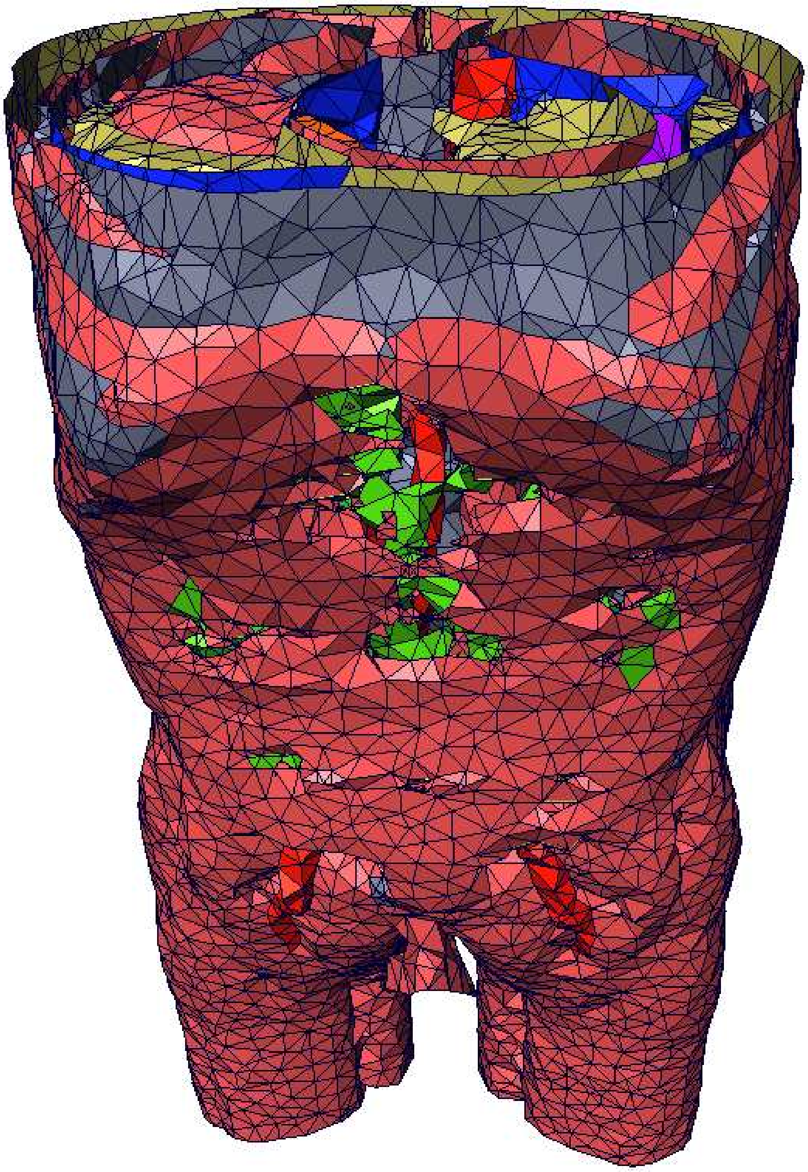

Figure 6.4: *The same surfaces as in Fig. 6.3, but with the skin layer removed.*

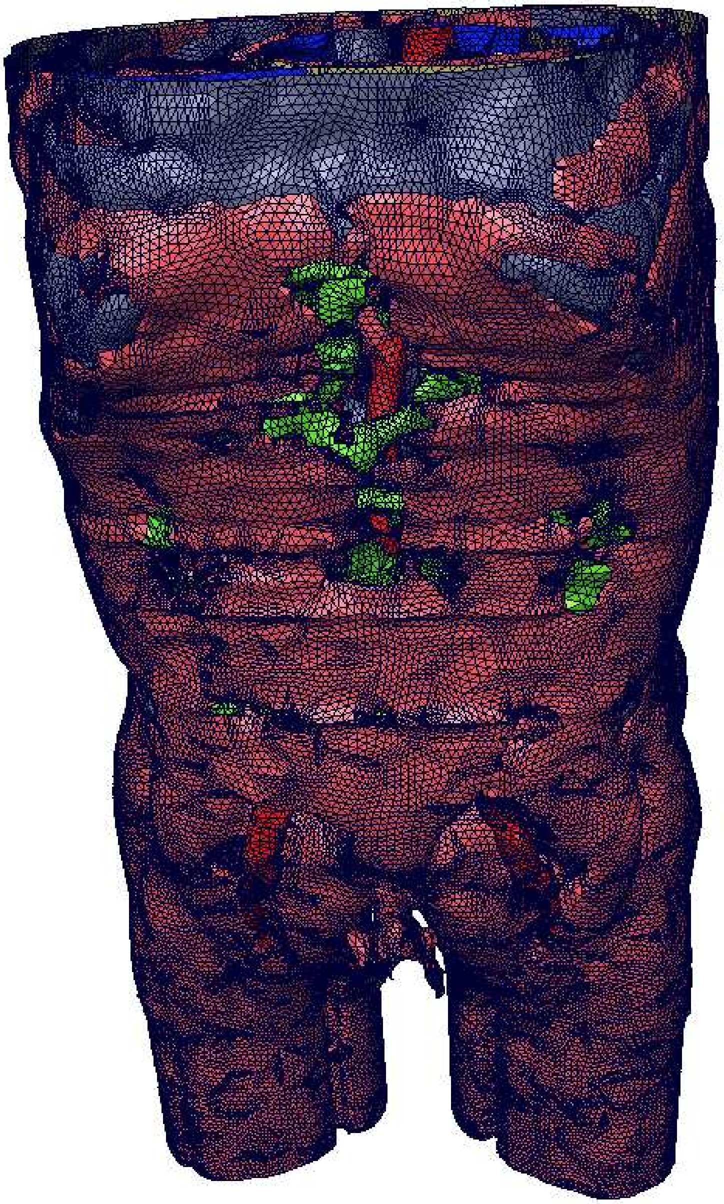

Figure 6.5: *The coarse surfaces after two regular refinement steps.*

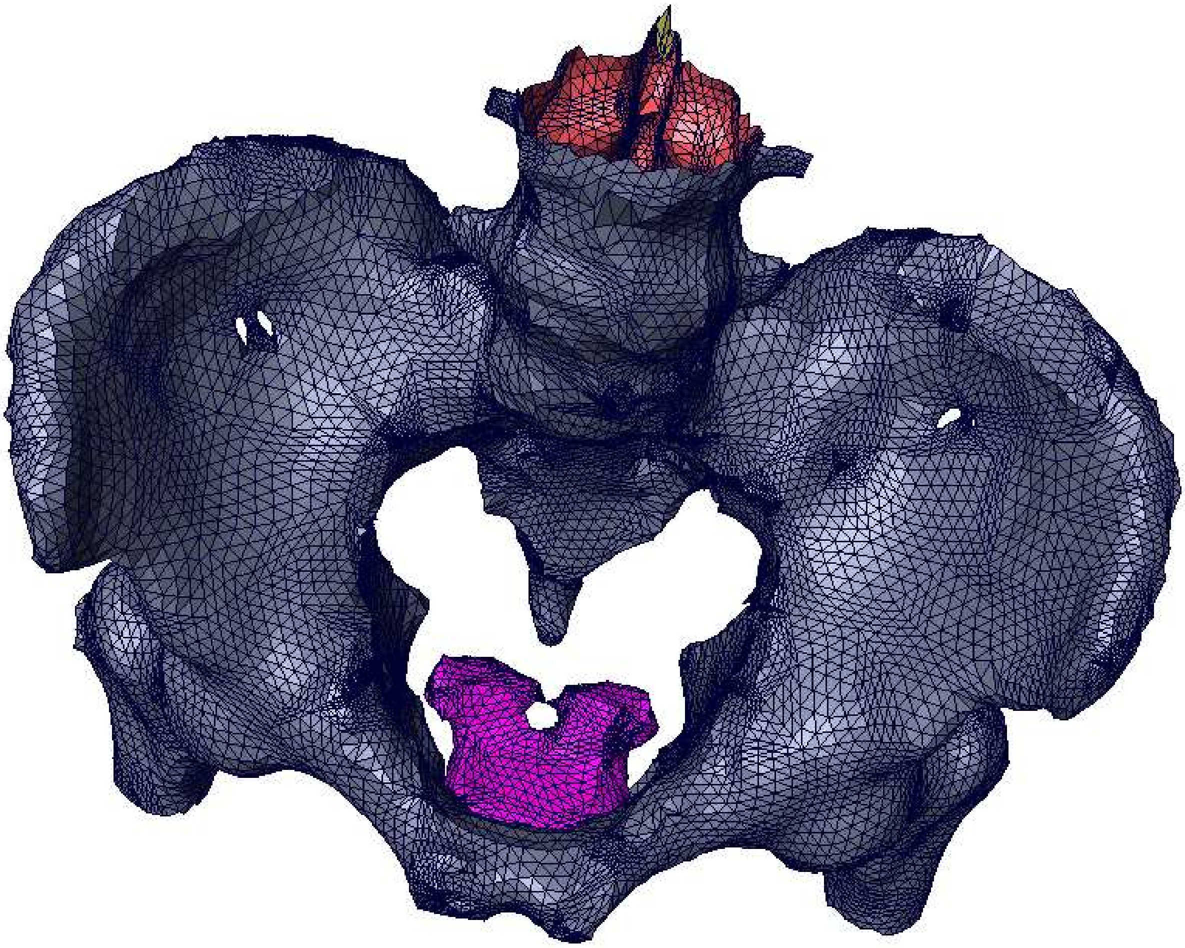

Figure 6.6: *Partial view of Fig. 6.5, showing the pelvic bone and the tumor.*

surfaces to Kaskade [9], the finite element software delevoped at the ZIB that is used for the treatment planning computations.

6.2 Biomechanical Contact Problems

This section briefly describes a second application of the parametrized surfaces. In close collaboration with Dr. Rolf Krause from the Department of Mathematics at the Freie Universität Berlin, we used them for handling a problem in biomechanics. Using the skull model shown in Fig. 6.7, we wanted to compute the strains and stresses in the bone that result from biting on a rigid block. For that purpose, we again used our simplification algorithm to generate coarse base grids of the mandible and the skull. Fig. 6.8 shows the result of the simplification process.

The two base grids were used as the input for the Amira mesh generator. Based on the advancing-front approach by Löhner and Parikh [55], high-quality tetrahedral meshes were created. Together with the carefully smoothed respective parametrizations, these two meshes were imported into the finite-element toolbox UG [8]. We also supplied the rigid obstacle, which is a little box that fits between the teeth of our skull model. For the boundary conditions, we chose constant vertical pressure on the upper third of the skull and the lower third of the mandible, forcing the head to bite. We used homogeneous Neumann boundary conditions, i.e., surface strains equal to zero, on the remaining boundary.

The problem at hand is an instance of *Signorini's problem*. Signorini's problem models the contact of an elastic body with a rigid foundation. It is nonlinear and nondifferentiable. We let UG use an adaptive nonlinear monotone multigrid method [48] for solving the PDE problem. However, when refining the tetrahedral meshes using the boundary parametrization for accurate geometry information, we obtained some mesh inconsistencies (see Fig. 6.9). These occur when the refined boundary deviates so much from its linear interpolation that the newly inserted vertices penetrate more than one layer of tetrahedra into the volume grid (cf. also Sec. 5.2). We have tried to smooth out those inconsistencies by expressing the new boundary surface as a set of deformation vectors on the canonically refined surface and using those vectors as the Dirichlet boundary values of a linear elasticity problem. This led to some improvement, however, we could not avoid forsaking geometric accuracy for consistency in some areas and simply using canonical refinement on some edges.

Aside from these technical problems, we achieved very good results. Since geometric detail can be regained whenever necessary, computation can be started on a grid that is much coarser, resulting in lower execution times without loss of accuracy. As the final result, we show the adaptively refined grid in Fig. 6.11 and the magnitudes of the computed strains as a color-encoding on the surface in Fig. 6.10.

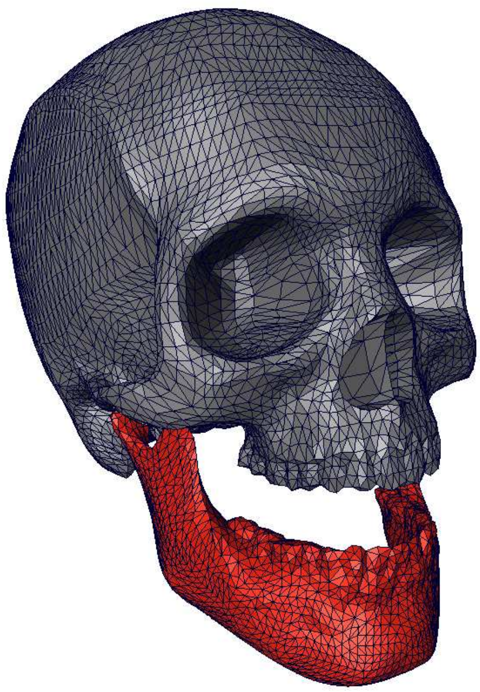

Figure 6.7: *The original skull surfaces.*

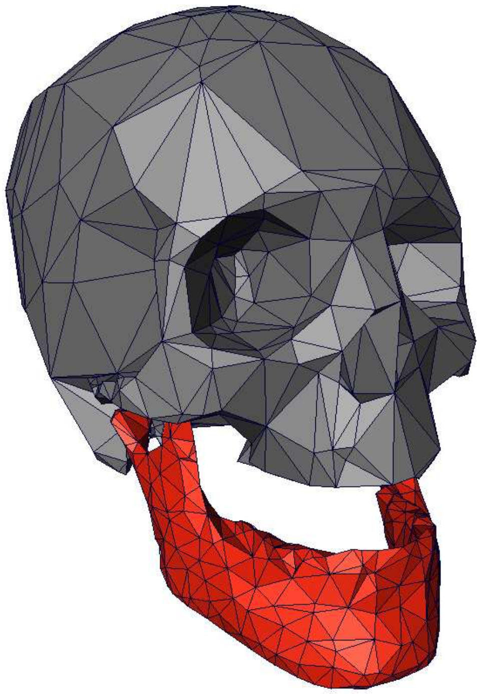

Figure 6.8: *The simplified version of the skull.*

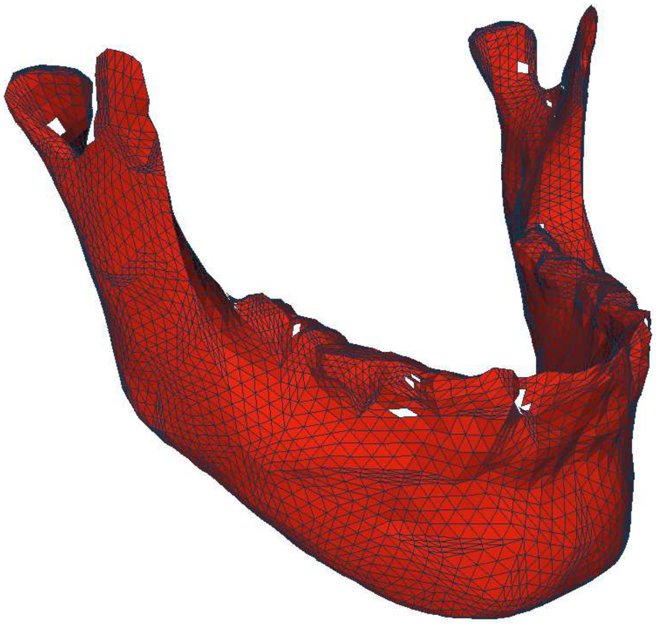

Figure 6.9: *Straightforward usage of boundary parametrizations leads to mesh inconsistencies, which show up as little holes.*

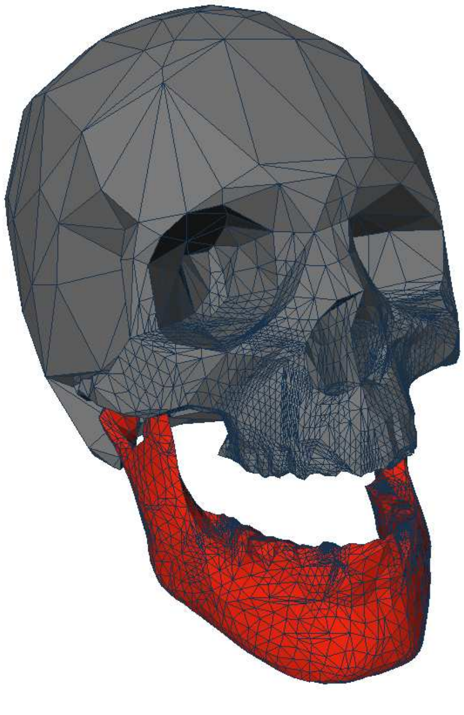

Figure 6.10: *The adaptively refined grid as produced by the finite element toolbox* UG.

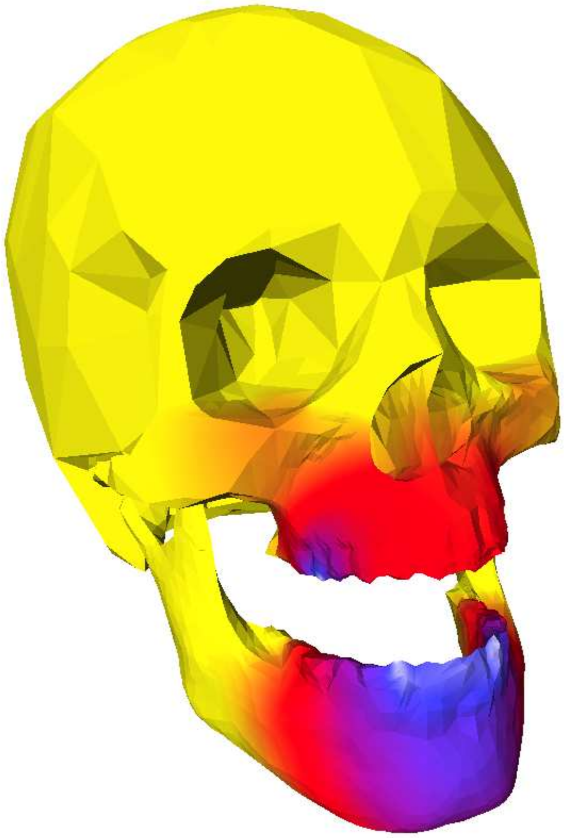

Figure 6.11: *The deformation of the skull computed on the grid in Fig. 6.10.*

7 Outlook and Conclusion

7.1 Conclusion

In this thesis we have presented a system that allows to store arbitrary triangulated surfaces as functions over simplified versions of themselves. They are intended to be used as boundary parametrizations by finite-element solvers that use adaptive mesh refinement. Thus, the efficiency and accuracy of such solvers can be increased.

The parametrized surfaces are constructed using a surface simplification algorithm. We have given an overview over the different possibilities to simplify polygonal surfaces. We have then proceeded to show how a simplification algorithm based on the point-removal primitive can be extended to carry and propagate a parametrization of the original surface. As an additional feature, the algorithm is not restricted to manifold surfaces, which is necessary for use with interface surfaces that have been extracted from labeled volume data.

In a further chapter we have discussed the issue of parametrization quality. We have first presented three ways to map disk-like triangulated surface patches to convex polygons in the plane. We have then introduced a measure for the local distortion of a parametrization. The shape-preserving parametrization as proposed by Floater was found to be superior to the others. The remainder of that chapter described several ways to extend the desirable qualities of Floater's parametrization to surfaces that are not homeomorphic to the unit disk.

As part of this thesis, an actual implementation of a surface parametrization system has been accomplished. This has been done as an extension to the scientific visualization package AMIRA which is being developed at the *Zuse Institute Berlin (ZIB)*. The central part of the implementation is the new data type. Further important modules are the simplifyer and an editor that implements all smoothing algorithms from Chapter 4. Additionally, several utility modules have been implemented such as a comfortable display module, an editor that allows manual changes to the base grid, and the shape morpher that will be described below. Last but not least, the data type together with routines for access and file-I/O has been made available as a standalone library.

The system has been tested in real-life settings. We have used it with hyperthermia patient models using the AMIRA-extension HYPERPLAN. We have also used it for biomechanical problems with the FEM-package UG.

7.2 Further Research Directions

7.2.1 Topology Changes

While designing our surface simplifyer, we have made a large effort to make sure that the topological type of the input surface remains unchanged during reduction. The reason for this is quite obvious: Given two manifolds $\mathcal{M}$ and $\mathcal{N}$, there can only exist a continuous bijection from $\mathcal{M}$ to $\mathcal{N}$ if the two are of the same topological type, which means they have to have the same genus, the same number of boundary manifolds, etc. Of course, the situation doesn't get better when non-manifold surfaces are considered. However, representing a high-resolution surface as a function over a coarse base grid is just such a bijection. Therefore, if we allow topology changes to occur, the straightforward approach used in this thesis to retain the original surface doesn't work anymore.

In some situations, however, it is quite desirable to permit topology changes. When dealing with complicated geometries, of which hyperthermia models are a prime example, topology invariance is very restrictive. While simplifying, one would like little holes and bubbles to disappear, small structures to grow together to form larger structures, and parts that are only connected through a single edge or vertex to detach. There are a few surface simplification algorithms that do just that without difficulty, e.g., the approach by Garland and Heckbert [33]. El-Sana and Varshney [29] describe a simplification algorithm that reduces the surface genus in a *controlled* fashion. However, both approaches neglect the need to be able to reextract the original geometry later, which is where the real problem is.

The only ones to our knowledge who have considered genus-reducing simplification in a remeshing context are Guskov *et al.* [35]. They introduce a new data structure they call the *hybrid mesh*, which allows arbitrary regular refinement with topology changes. However, they require that all occuring genus changes be marked by hand, which means a lot of work when working with complicated models. An ideal solution should be fully automated and would probably include elements from all of the articles cited above.

7.2.2 Grid Quality

A lot remains to be done in the field of mesh generation and adaptive refinement. The degradation of element quality and the mesh inconsistencies that occur due to moving boundaries (Sec. 5.2) aren't theoretical problems but a serious hassle. So far, paying close attention to the quality of the base grid and meticulously smoothing the parametrization helps quite a lot, but mesh self-intersections invariable occur as soon as models get reasonably complex.

Neglecting low-quality elements so far, we have tried several heuristics to avoid self-intersecting meshes. Some of them do work, however, they lessen the improvement brought in by the parametrized boundaries. Also, they are by no means 'elegant'. For a more theoretically sound approach, we refer the reader to [65]. The task of creating and maintaining high-quality FEM grids remains one of most challenging problems of today.

7.3 Further Uses of Parametrized Surfaces

Parametrized surfaces constitute a tool that is actually very versatile. Its applications extend way beyond the field of numerical computations. In this section, we'll briefly describe several problems that can be approached using parametrized surfaces. The first one, *morphing*, has been implemented in AMIRA using the data structures described in this thesis. The list is by no means exhaustive. In addition to the following three topics one could mention *level-of-detail rendering* [27], *geometry compression* [42], and *texture map generation* [59].

7.3.1 Morphing

Morphing has received a great deal of attention in recent years. Due to its increasing use as a special effect in movies, it is one of the rather few problems in computer graphics that is known to the general public.

Morphing, or, as it is sometimes called, *warping*, denotes the problem of continuously deforming a surface model into another one of not necessarily the same topology. That is, we're looking for a one-parameter family of surfaces $\mathcal{S} = \{S(\rho) \mid \rho \in [0,1]\}$ such that $S(0)$ is the original surface, $S(1)$ is the target surface and that the map $\rho \to S(\rho) \in \mathcal{S}$ is continuous. Since, given two surfaces, the family $\mathcal{S}$ is by no means unique, the user is usually requested to mark sets of feature points on the surfaces that will be required to correspond via the morphing. These points will generally have some geometric significance. For example, when morphing two heads, one might choose the tip of the nose, the chin, the corners of the mouth and the like. The algorithm should then construct some surface family under those constraints.

Such a morph sequence generator can be implemented using the data structures for parametrized triangulated surfaces described in this thesis. A basic prototype looks like the following: given two triangulated surfaces S and S' of identical topological type, the user marks an equal number of feature points on each of them. On each surface, the points are then connected by paths that triangulate each surface such that the two triangulations are combinatorically equivalent. This can be done manually or using an automated tool for example implementing the algorithm described by Praun *et al.* in [64]. The triangulated surfaces are then transformed into the parametrized form using the explicitely given triangulations as base grids. This transformation is of course *not* done using surface simplification but by explicitly creating a base grid triangle for each triangular patch on the surface. We map points on patch boundaries onto base grid edges via a one-dimensional chord-length parametrization and we use the Floater-scheme for all other vertices. The quality of the parametrization can be improved further by applying one of the relaxation schemes described in Chapter 4. This sequence of steps is shown for the cartoon figure head in Fig. 7.1.

Since the two triangulations were required to be equivalent, we now have the two surfaces S and S' parametrized over two isomorphic base grids $\mathcal{B}$ and $\mathcal{B}'$ using the respective parametrization functions ϕ and ϕ'. Thus, there exists a bijection θ from the geometric realization of $\mathcal{B}$ to the geometric realization of $\mathcal{B}'$. This is basically all

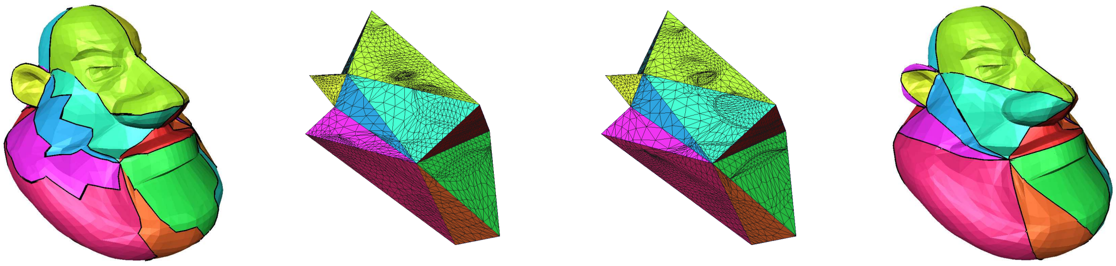

Figure 7.1: *Constructing a parametrization of 'Al', the cartoon figure, that is suitable for morphing. i) the surface with marked points and a rough triangulation; ii) the corresponding base grid; iii) the base grid after the application of edge-relaxation; iv) the original surface showing the relaxed triangulation.*

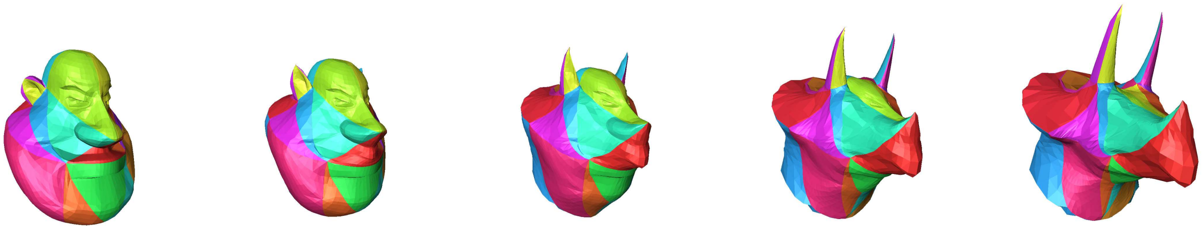

Figure 7.2: *A morph sequence from the cartoon figure into the triceratops head showing five equidistant steps.*

we need, because it gives us a bijection Φ from $S = S(0)$ to $S' = S(1)$ as follows: To each point $\mathbf{p} \in S$ we consider its projection onto the base grid $\phi^{-1}(\mathbf{p})$. From there, we find the corresponding point on $\mathcal{B}'$ using the map θ, and then a point $\mathbf{p}'$ on $\mathcal{S}'$ by the parametrization function ϕ' of $\mathcal{B}$. In formulas:

$$\mathbf{p}' = \Phi(\mathbf{p}) = \phi' \circ \theta \circ \phi^{-1}(\mathbf{p}).$$

Any intermediate model can now be generated easily by simply interpolating between corresponding points. Fig. 7.2 demonstrates this for the case of linear interpolation.

The simple morphing scheme described above has still many shortcomings. Linear interpolation, for example, doesn't always generate image sequences that are visually pleasing. Also, nothing has been said about the triangular meshes used to represent the surface. A mesh that suits one surface well isn't necessarily fit to represent the other one, too (see [80] for more details on those two points). Also, the restriction of being only able to transform two surfaces of identical topological type is too severe for many purposes. Therefore, completely different approaches have been suggested, as for example the function interpolation method described in [71].

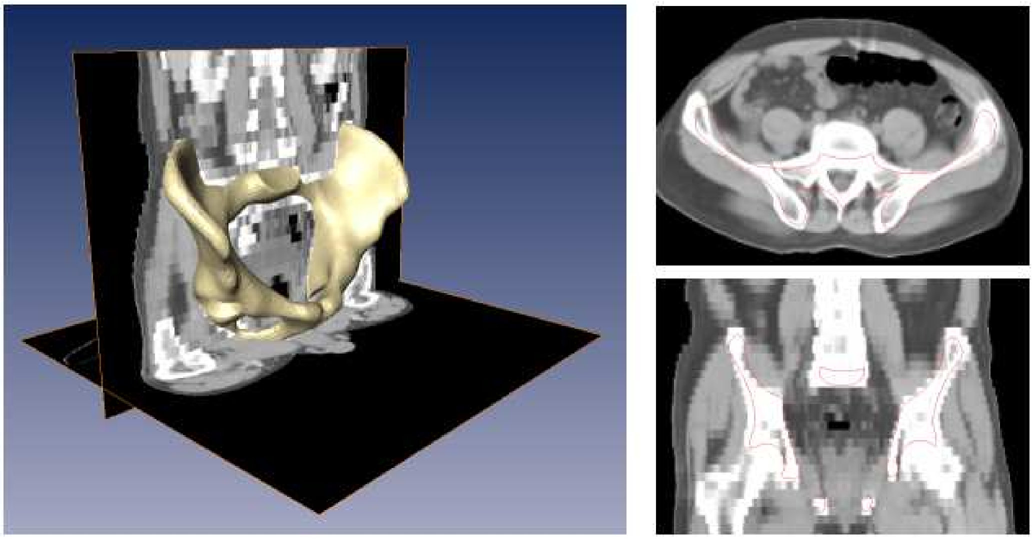

Figure 7.3: *The average of eight different pelvic bones placed in a stack of image data.*

7.3.2 Model-Based Image Segmentation

A second, completely different application for parametrized surfaces arises in *image segmentation*. As has been mentioned in Sec. 2.4, image segmentation means labeling voxels with materials. It is of very high importance for all medical model building, and still includes a lot of manual work.

A particular feature of medical image segmentation is that one is frequently dealing with similar, reccuring instances of a few 'archetypal' models. For example, when treating hyperthermia patients with tumors in the abdomen, there is always the pelvic bone to be segmented, which has a shape that doesn't vary much from one patient to another. This calls for a technique called *model-based segmentation*.

Model-based segmentation uses a set of example shapes that are supposed to contain all possible variations of the shape to be treated. These shapes are extracted manually and are available in form of triangulated surfaces. Using the approach described in the section on morphing, these shapes are all put in correspondence with each other. This correspondence allows to perform a *principal component analysis (PCA)* of the set of shapes (see e.g. [12]). The result is an average model plus a set of vectors that spans the space of all possible variations. Intuitively, the idea is that a few of those *principal components* should suffice to represent all relevant shapes.

The process of actually segmenting a new data set starts by placing the average model at an approximate position in the volume spanned by the image data. For each vertex on the model surface, one then looks at the grey values of the data in the direction normal to the surface at that point. From this grey value profile, one can compute a normal displacement vector that would move the vertex to a position on the actual boundary.

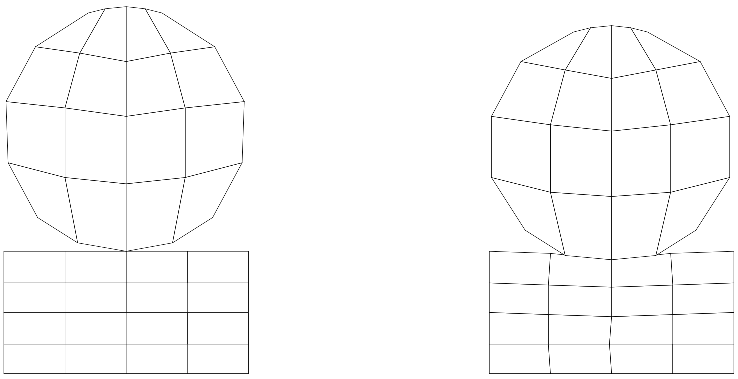

Figure 7.4: *An example contact problem. A sphere is pressed onto a flat surface, which results in a deformation. Both illustrations in this section are courtesy of Rolf Krause.*

However, simply applying the respective displacements to all model vertices would result in a surface that is very distorted. Therefore, one imposes the constraint that the resulting surface has to be element of the space spanned by the principal component vectors. This ensures that the result is still ‘pelvis-shaped’.

A system for model-based segmentation of the pelvic bone is currently being developed at ZIB based on the visualization system Amira. An average pelvis together with a surrounding trial data set can be seen in Fig. 7.3.

7.3.3 Correspondence-Finding in Contact Problems

In contact problems, the goal is to compute the deformations and internal stresses that bodies undergo when they are subjected to external forces and encounter obstacles (Fig. 7.4). In the simplest case, these obstacles might come in form of a rigid, completely smooth plate, such that only the deformation of one body has to be computed. In the more complex *multi-body problems*, several objects interact that all show elastic behaviour.

When two bodies come into contact, two conditions have to hold at the contact surface. First, there’s the obvious *impenetrability condition*. Second, in order to obtain a stable solution, the normal components of the surface stresses have to be equal. This interaction between positions and stresses of originally separate computation domains is a highly nontrivial problem [48].

The two bodies are of course only given in form of disrete meshes. Computing the impact of the stresses of one mesh onto the other involves interpolation that depends on the exact relative position of the meshes. However, the mesh position constitutes already a part of the desired solution, which can’t be computed without the force interactions. Therefore, what is needed is an approximation of the final shapes of the meshes at the surface of contact.

In simple cases, the initial grids can be set up such as to assure a perfect matching upon contact. In those cases, equality of position and normal stress on the whole contact surface reduces to equality at all contact vertices. However, for all practical problems this approach is unusable. Direct projections of the potential contact surface into a plane cannot guarantee to be a bijection and can lead to high distortions.

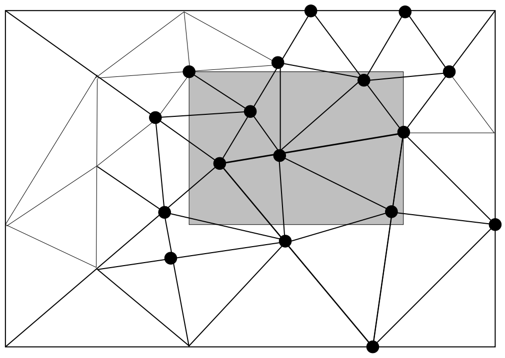

Figure 7.5: *At the contact surface, the grid of the grey cube (grid not shown) is overlapped by the other grid.*

We therefore propose to use the parametrized surfaces described in this thesis for this problems. Given the potential interaction regions on both bodies, these can be mapped to suitable plane polygons. Using the techniques described in Chapter 4, the mappings will be bijective and relatively undistorted. This approximate correspondence between the two possible regions of contact would then allow to compute the integrals necessary to obtain mutual stresses.

Bibliography

[1] Gabriel Acosta and Ricardo G. Durán. The maximum angle condition for mixed and nonconforming elements: Application to the Stokes equations. *SIAM J. Numer. Anal.*, 37(1):18–36, February 2000.

[2] P. Alexandrov. *Combinatorial Topology*, volume 1. Greylock, Rochester, New York, 1956.

[3] Gabrielle Allen, Tom Goodale, Gerd Lanfermann, Edward Seidel, Werner Benger, Hans-Christian Hege, André Merzky, Joan Massó, and John Shalf. Solving Einstein's equation on supercomputers. *IEEE Computer*, pages 52–59, December 1999.

[4] Hans Wilhelm Alt. *Lineare Funktionalanalysis*. Springer Verlag, 1985.

[5] Amira visualization and modeling system. `http://www.AmiraVis.com`.

[6] I. Babuška and A. K. Aziz. On the angle condition in the finite element method. *SIAM J. Numer. Anal.*, 13:214–227, 1976.

[7] Randolph E. Bank, Andrew H. Sherman, and Alan Weiser. Refinement algorithms and data structures for regular local mesh refinement. In R. Stepleman et al, editor, *Scientific Computing, IMACS Trans. on Scientific Computation*, volume 1. North-Holland, 1983.

[8] P. Bastian, K. Birken, K. Johannsen, S. Lang, N. Neuß, H. Rentz–Reichert, and C. Wieners. UG – a flexible software toolbox for solving partial differential equations. *Computing and Visualization in Science*, 1:27–40, 1997.

[9] Rudolf Beck, Bodo Erdmann, and Rainer Roitzsch. KASKADE 3.0 – An object oriented adaptive finite element code. Technical Report TR 95-04, Konrad-Zuse-Zentrum für Informationstechnik, 1995.

[10] Martin Berzins. Mesh quality: A function of geometry, error estimates or both? In *7th International Meshing Roundtable*, Dearborn, Michigan, 1998.

[11] Jürgen Bey. Analyse und Simulation eines Konjugierte–Gradienten–Verfahrens mit einem Multilevel–Präkonditionierer zur Lösung dreidimensionaler, elliptischer Randwertprobleme für massiv parallele Rechner. Master's thesis, Institut für Geometrie und Praktische Mathematik, RWTH Aachen, 1991.

[12] Andrew Blake and Michael Isard. *Active Contours*. Springer Verlag, 1998.

[13] Béla Bollobás. *Graph Theory; An Introductory Course.* Springer-Verlag, 1979.

[14] Folkmar Bornemann, Bodo Erdmann, and Ralf Kornhuber. Adaptive multilevel–methods in three space dimensions. *Int. J. Numer. Methods in Engng.*, 36:3187–3203, 1993.

[15] Dietrich Braess. *Finite Elements.* Cambridge University Press, 2001.

[16] Peter J.C. Brown and Christopher T. Faigle. A robust efficient algorithm for point location in triangulations. Technical report, Cambridge University, February 1997.

[17] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Comput. Aided Design*, 10:350–365, 1978.

[18] Zhiming Chen and Ricardo H. Nochetto. Residual type a posteriori error estimates for elliptic obstacle problems. *Numerische Mathematik*, 84(4):527–548, 2000.

[19] Jonathan Cohen, Dinesh Manocha, and Marc Olano. Simplifying polygonal models using successive mappings. In *Proceedings IEEE Visualization*, pages 395–402, October 1997.

[20] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[21] Tony DeRose, Michael Kass, and Tien Truong. Subdivision surfaces in character animation. In *Proceedings of SIGGRAPH*, pages 85–94, 1998.

[22] Peter Deuflhard and Andreas Hohmann. *Numerische Mathematik.* de Gruyter, Berlin, 1991.

[23] D. Doo and M. Sabin. Behaviour of recursive division surfaces near extraordinary points. *Comput. Aided Design*, 10:356–360, 1978.

[24] W. Dörfler and Martin Rumpf. An adaptive strategy for elliptic problems including a posteriori controlled boundary approximation. *Math. Comput.*, 67(224):1361–1382, 1998.

[25] Tom Duchamp, A. Certain, Antony DeRose, and Werner Stuetzle. Hierarchical computation of PL harmonic embeddings. Technical report, University of Washington, July 1997.

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

[27] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. *Computer Graphics*, 29(Annual Conference Series):173–182, 1995.

[28] Herbert Edelsbrunner. *Geometry and Topology for Mesh Generation.* Cambridge University Press, 2001.

[29] Jihad El-Sana and Amitabh Varshney. Controlled simplification of genus for polygonal models. In Roni Yagel and Hans Hagen, editors, *IEEE Visualization '97*, pages 403–412, 1997.

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

[31] Theodore Frankel. *The Geometry of Physics.* Cambridge University Press, 1997.

[32] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[33] Michael Garland and Paul Heckbert. Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH*, pages 209–218, 1997.

[34] Paul-Louis George and Houman Borouchaki. *Delaunay triangulation and meshing.* Editions HERMES, 1998.

[35] Igor Guskov, Andrei Khodakovsky, Peter Schröder, and Wim Sweldens. Hybrid meshes. Submitted for publication, 2001.

[36] Igor Guskov, Kiril Vidimče, Wim Sweldens, and Peter Schröder. Normal meshes. In *Proceedings of SIGGRAPH*, 2000.

[37] Wolfgang Hackbusch. *Theorie und Numerik elliptischer Differentialgleichungen.* B.G. Teubner, Stuttgart, 1986.

[38] Hans-Christian Hege, Martin Seebass, Detlev Stalling, and Malte Zöckler. A generalized marching cubes algorithm based on non-binary classifications. Technical Report SC 97-05, Konrad-Zuse-Zentrum für Informationstechnik, Berlin, 1997.

[39] John David Jackson. *Classical Electrodynamics.* John Wiley & Sons, third edition, 1998.

[40] Klaus Jänich. *Vektoranalysis.* Springer Verlag, 1993.

[41] Fritz John. *Partial Differential Equations.* Springer Verlag, 1982.

[42] Andrei Khodakovsky, Peter Schröder, and Wim Sweldens. Progressive geometry compression. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 271–278. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[43] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983.

[44] Reinhard Klein, Gunther Liebich, and Wolfgang Straßer. Mesh reduction with error control. In Roni Yagel and Gregory M. Nielson., editors, *IEEE Visualization '96*, pages 311–318, 1996.

[45] Wilhelm Klingenberg. *A Course in Differential Geometry.* Springer Verlag, 1977.

[46] Leif Kobbelt. Discrete fairing and variational subdivision for freeform surface design. *The Visual Computer*, 16:142–158, 2000.

[47] Leif Kobbelt, Swen Campagna, and Hans-Peter Seidel. A general framework for mesh decimation. In *Graphics Interface Proceedings*, pages 43–50, 1998.

[48] Rolf Krause. *Monotone Multigrid Methods for Signorini's Problem with Friction.* PhD thesis, Freie Universität Berlin, Department of Mathematics, July 2001.

[49] U. Labsik, K. Hormann, and G. Greiner. Using most isometric parametrizations for remeshing polygonal surfaces. In *Proceedings of Geometric Modelling and Processing 2000*, pages 220–228, 2000.

[50] Jens Lang, Bodo Erdmann, and Martin Seebass. Impact of nonlinear heat transfer on temperature control in regional hyperthermia. *IEEE Transactions on Biomedical Engineering*, pages 1129–1138, 1999.

[51] Thomas Lange. Regionenbasierte Segmentierungsverfahren für dreidimensionale Bilddaten. Master's thesis, TU Berlin, 1999.

[52] Aaron W. F. Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. MAPS: Multiresolution adaptive parametrization of surfaces. In *Proceedings of SIGGRAPH*, pages 95–104, 1998.

[53] Stanley B. Lippman. *C++, Einführung und Leitfaden.* Addison-Wesley, 1991.

[54] Vladimir D. Liseikin. *Grid Generation Methods.* Springer Verlag, 1999.

[55] Rainald Löhner and Paresh Parikh. Generation of three-dimensional unstructured grids by the advancing-front method. *International Journal For Numerical Methods in Fluids*, 8:1135–1149, 1988.

[56] Charles Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, 1987.

[57] W.E. Lorensen and H.E. Cline. Marching cubes: a high resolution 3D surface construction algorithm. *Computer Graphics*, 21:163–169, 1987.

[58] David P. Luebke. Robust view-dependent simplification for very large-scale CAD visualization. Technical Report CS-99-33, University of Virginia, 1999.

[59] Makoto Maruya. Generating a texture map from object-surface texture data. *Computer Graphics Forum*, 14(3):397–406, 1995.

[60] J. Nadobny, D. Sullivan, P. Wust, M. Seebass, P. Deuflhard, and R. Felix. A high-resolution interpretation of arbitrary interfaces for the FDTD method. *IEEE Transactions on Microwave Theory and Techniques*, 46(11):1759–1766, 1998.

[61] Colm O'Dunlaing, Colum Watt, and David Wilkins. Homeomorphism of 2-complexes is equivalent to graph isomorphism. Technical Report TCDMATH 98-04, Math. Dept., Trinity College, Ireland, 1998.

[62] K.D. Paulsen, S. Geimer, J. Tang, and W.E. Boyse. Optimization of pelvic heating rate distributions with electromagnetic phased arrays. *International Journal of Hyperthermia*, 15:157–186, 1999.

[63] Harry H. Pennes. Analysis of tissue and arterial blood temperatures in the resting human forearm. *Journal of Applied Physics*, 1:93–122, 1948.

[64] Emil Praun, Wim Sweldens, and Peter Schröder. Consistent mesh parametrizations. In *Proceedings of SIGGRAPH*, 2001.

[65] Martin Rumpf. A variational approach to optimal meshes. *Numer. Math.*, 72(4):523–540, 1996.

[66] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. *Computer Graphics*, 26(2):65–70, 1992.

[67] Martin Seebass, Rudolf Beck, Johanna Gellermann, Jacek Nadobny, and Peter Wust. Electromagnetic phased arrays for regional hyperthermia. *International Journal of Hyperthermia*, 17, 2001.

[68] Alla Sheffer and Eric de Sturler. Surface parameterization for meshing by triangulation flattening. In *9th International Meshing Roundtable*, pages 161–172, 2000.

[69] C.W. Song, A. Lokshina, J.G. Rhee, M. Patten, and S.H. Levitt. Implication of blood flow in hyperthermic treatment of tumors. *IEEE Transactions on Biomedical Engineering*, 31:9–16, 1984.

[70] Detlev Stalling, Martin Seebass, Malte Zöckler, and Hans-Christian Hege. Hyperthermia treatment planning with HyperPlan – User's Manual. Technical Report ZR 00-27, Konrad-Zuse-Zentrum für Informationstechnik, October 2000.

[71] Greg Turk and James F. O'Brien. Shape transformation using variational implicit functions. *Computer Graphics*, 33(Annual Conference Series):335–342, 1999.

[72] W.T. Tutte. Convex representations of graphs. *Proc. London Math. Soc.*, 10:304–320, 1960.

[73] W.T. Tutte. How to draw a graph. *Proc. London Math. Soc.*, 13:743–768, 1963.

[74] Jeroen van de Kamer. *High-Resolution Regional Hyperthermia Treatment Planning.* PhD thesis, University of Utrecht, 2001.

[75] H. A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *Siam Journal on Scientific and Statistical Computing*, 13:631–644, 1992.

[76] Andreas Veeser. Efficient and reliable A posteriori error estimators for elliptic obstacle problems. *SIAM J. Numer. Anal.*, 39(1):146–167, February 2002.

[77] Natascha Westerhoff. Geometrische Methoden zur 3D-Segmentierung. Master's thesis, TU Berlin, 1999.

[78] Peter Wust, Martin Seebass, Jacek Nadobny, Peter Deuflhard, G. Mönich, and R. Felix. Simulation studies promote technological development of radiofrequency hyperthermia. *International Journal of Hyperthermia*, 12:477–494, 1996.

[79] S. Zhang. *Multilevel Iterative Techniques.* PhD thesis, Pennsylvania State University, 1988.

[80] Malte Zöckler, Detlev Stalling, and Hans-Christian Hege. Fast and intuitive generation of geometric shape transitions. *The Visual Computer*, 16(5):241–253, 2000.

[81] Denis Zorin and Peter Schröder. Subdivision for modeling and animation. SIGGRAPH 2000 Course Notes.