TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

Diplomarbeit

Curvature Based Surface Mesh Simplification

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Masterstudium Technische Mathematik

eingereicht von

Christoph Lenz, BSc
Matrikelnummer 0828641

ausgeführt am Institut für Mikroelektronik
eingereicht an der Fakultät für Mathematik und Geoinformation der Technischen Universität Wien

Betreuung

Betreuer:	O.Univ.Prof. Mag.rer.nat. Dr.techn. Helmut Pottmann
Mitwirkung:	Univ.Ass. Dipl.-Ing. Dr.techn. Josef Weinbub, BSc
	Univ.Ass. Dipl.-Ing. Lukas Gnam, BSc

Wien, 14.05.2019

(Unterschrift Verfasser) (Unterschrift Betreuer)

Contents

Chapter 1

Introduction

1.1 Motivation and Objectives

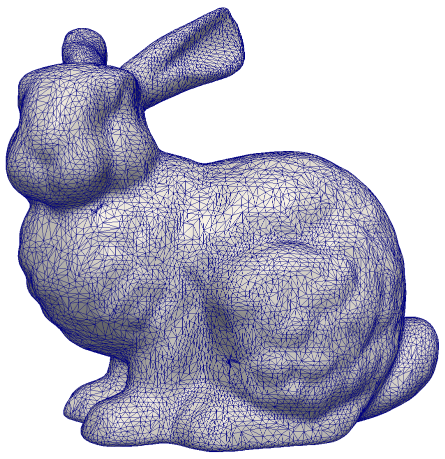
In the modern world nearly all people use electronic devices based on microprocessors to simplify their lives, e.g., smart phones, computers, or virtual assistants. Due to the increasing complexity of these devices, the investigation and evaluation of new as well as adapted fabrication techniques and processes is of major importance in the semiconductor industry. Examples are the necessary adaptations to improve fabrication processes to sustain the continued miniaturization efforts of microelectronic devices, e.g., transistor-type devices. Such new processes pose new challenges, as well as increasing the complexity of both, device design and fabrication processes. These fabrication steps include processes like etching, deposition, and diffusion [40]. Due to their high complexity, experimental evaluations have become very time and resource consuming [20]. Thus, computer simulation became a key tool to accurately predict specific fabrication steps, thus, enabling the partial replacement of conventional experiments with much faster and orders-of-magnitude cheaper computer simulations. In semiconductor research and industries, the umbrella term *Technology Computer-Aided Design* (TCAD) is used to refer to such computer simulations. TCAD covers not only fabrication processes but also the simulation of the electrical characteristics of individual devices and the electrical behaviour of entire circuits. [40].

As previously hinted, etching is an important fabrication process and can be modelled and thus predicted (as for other fabrication processes) using process TCAD [1] simulation tools. In this case, etching simulations rely on the representation of an evolving surface; evolving because the actual etching process selectively removes material from the surface and thus the simulation must be capable of handling time-evolving deformations. Such time-evolving simulations (typically referred to as topography simulations) depend on an accurate representation of the surface. This representation can for example be conducted using the level-set method [31]. However, the level-set is an implicit representation; to generate an explicit surface an extraction algorithm has to be applied, e.g., a *marching cubes algorithm* [19]. The explicit discretization of the geometry resulting from the extraction algorithm is commonly known as a *mesh*.

There are multiple kinds of meshes, e.g., *volume meshes* or *surface meshes*. Meshes use the union of *simple* base geometries to represent more complex geometries. *Simplex meshes* use, for example, triangles in the case of surface meshes, or tetrahedrons in the case of volume meshes for their base geometries. The three points of the triangle, respectively four of a tetrahedron, are called *vertices*, and the lines that connect the vertices are called *edges*.

[1] In this context, *process* refers to fabrication processes necessary to build semiconductor devices, e.g., a transistor-type device.

In this work these base geometries are referred to as the *elements* of a mesh. In literature other commonly used names for elements are *facets* or *cells*. In Figure 1.1 an example of a simplex surface mesh [2] is shown. There are also non-simplex meshes like *quad meshes* [1]. However, this work focuses on simplex surface meshes.


(a) A standard reference model of the so called Stanford Bunny consisting of 30,338 elements.

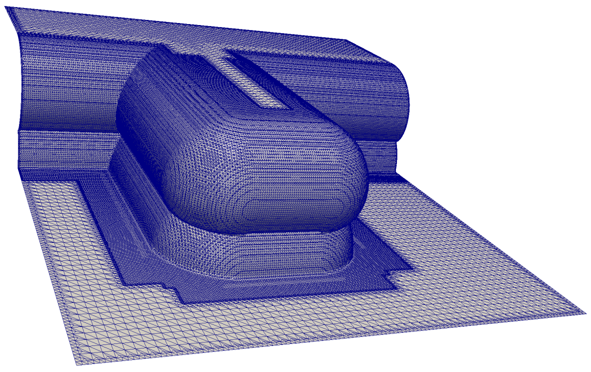

(b) A typical surface mesh occurring in semiconductor device simulation with 140,698 elements.

Figure 1.1: Two example surface meshes.

A key aspect in topography simulations for process TCAD is the calculation of quantity distributions on the surface itself. This is, for example, achieved by a technique called surface flux calculations which is implemented by using *ray tracing* [23]. Although level-sets handle complex surface evolutions robustly [32], ray tracing on level-sets can be accelerated by extracting an explicit surface of the level-set and subsequently performing the ray tracing on this explicit surface [21]. After the extraction from the level-set the surface mesh can potentially have a too high resolution, yielding unnecessarily computationally expensive calculations in the subsequent simulation steps, see for example, the flat planes in Figure 1.1(b).

A typical use case of surface meshes is the creation of volume meshes, were the surface mesh acts as a starting point for the subsequent volume mesh generation process, e.g., the *advancing front method* [18]. The element quality (tetrahedrons) of the resulting volume mesh depends on the element quality (triangles) of the surface mesh. The resulting volume mesh is then used for solving partial differential equations using, for instance, the *Finite Element Method* (FEM) [2]. The computational complexity of a FEM increases with the number of elements of the given volume mesh. Additionally, the accuracy of the FEM depends on the quality and the resolution of the volume mesh elements [10]. All in all, volume meshes, and thus surface meshes, have a significant impact on the numerical solutions. Optimizing the originating surface meshes via mesh simplification algorithms by balancing the reduction of the number of elements with geometrical feature preservation is thus of paramount importance and the focus of this work.

The general method to simplify a surface mesh used in this work is the so called *edge collapse*. This method removes one edge from the surface mesh and replaces it with a vertex.

[2]Surface meshes find wide-spread application outside of TCAD. They are used, for example, in *three-dimensional* (3D)-rendering, e.g., video games or animated films [30].

There are multiple strategies to decide which edge is collapsed and which vertex is chosen to replace it, e.g., the *Lindstrom-Turk* simplification approach [17] or a simplification approach based on a *discrete curvature norm* [16].

The Lindstrom-Turk simplification uses the intersection of three linear independent constraints to calculate the new optimal vertex position after an edge collapse.

The simplification based on the discrete curvature norm calculates the value for the discrete curvature norm for the whole mesh and afterwards removes one vertex at a time and recalculates the value for the discrete curvature norm. The edge that changes the value of the curvature norm the least, if removed, is chosen and collapsed.

However, these methods have their drawbacks, they have high computation times and do not consider the element quality in the case of the discrete curvature norm [16]. The approach of Lindstrom and Turk reduces the resolution of geometric features too much with respect to the other elements in the surface mesh [17]. Therefore, this approach creates a simplified surface mesh that deviates unacceptably far from the geometry of the original surface mesh [3]. Furthermore, the calculation time of the simplification process is important, because when the calculation time of the simplification process takes longer than the additional computation time of, e.g., the ray tracing on the original geometry, there is no meaningful improvement to be gained.

The Lindstrom-Turk simplification algorithm [17] is also designed to simplify surface meshes were most elements contain information about the geometry of the surface mesh, e.g., most elements of the mesh do not lie in the same plane as their adjacent elements (see Figure 1.1(a)). However, when certain areas of the surface mesh have simpler geometry than other areas, the overall number of elements in these areas can be reduced by a larger amount, without deteriorating the geometry.

1.2 Research Goals

The goal of this work is to reduce the number of elements whilst balancing with feature preservation of a surface mesh created during the design process of a semiconductor device. Surface meshes extracted within such a process simulation step potentially consist of too many elements, were not all of them are necessary for preserving the geometry of the surface (see Figure 1.1(b)). The simplification approach presented in this work, called *region simplification* utilizes the so called *curvature* of a surface mesh. The curvature measures the rate at which the normal vector changes when it is moved over the surface. Subsequently, the elements are binned into different *regions* depending on their curvature. Hence, the created regions have different geometrical properties, e.g., flat region or high curvature region. For the proper simplification of the different regions, the Lindstrom-Turk simplification approach with different parameters, depending on the region's geometrical properties, is used.

To guarantee a fast and reliable method of calculating the curvature of a mesh, two commonly used methods of curvature calculation are compared to determine their viability for the simplification process developed in this thesis. These methods are *jet-fitting* [6], which tries to approximate the local geometry of the given surface with the help of a quadratic minimization problem, and *spatial averaging* [22], which uses areas and the values of certain operators to calculate the curvature.

[3]Deviations from the original mesh result in principal errors of the overall simulation as the intended geometry is not properly represented.

The surface simplification is only one step of many during the simulation of a fabrication process of a semiconductor device. Therefore, the computation time of the surface simplification is of importance regarding the overall simulation performance. To reduce the overall number of calculations that have to be computed during the simplification process several advanced data structures, like *hash tables*, are used to avoid redundant calculations.

1.3 Outline

In Chapter 2, mesh terminologies are defined and the general method used in this thesis to simplify a mesh is introduced.

In Chapter 3, the mathematical foundation, used in this thesis are established. These contain surfaces, the curvature of a surface in a given point, and two methods of calculating the curvature of a discrete surface or a mesh. Finally, both methods are evaluated.

In Chapter 4, the software tools that were used to develop the simplification process are introduced.

In Chapter 5, the process to simplify a high resolution mesh developed for this work is introduced.

In Chapter 6, the results from the simplification processes are compared against the approach of Lindstrom and Turk [17].

In Chapter 7, the work is concluded by giving a summary and an outlook to future work.

Nomenclature

$X, Y, Z, \dots$	Sets
$\mathbb{N}$	The set of natural numbers
$\mathbb{R}$	The set of real numbers
$x, y, z, \dots$	Scalars
$\mathbb{R}^n$	An n-dimensional vector space over the real numbers
$\bar{n}, \mathbf{x}_u, \dots$	Vectors in $\mathbb{R}^n$
$\lVert \cdot \rVert$	The euclidean norm
$\cdot$	The standard scalar product in $\mathbb{R}^n$
τ_n	The typology induced by the Euclidean norm
v	A vertex
p	A point
S	A surface in $\mathbb{R}^3$
$T_p(S)$	The tangent plane in the point p on the surface S
$\mathcal{M}$	A mesh
e	An edge in a mesh $\mathcal{M}$
e_v, e_w	The two vertices of an edge

Chapter 2

Meshes and Mesh Simplification

The first part of this chapter starts with the formal definition of a mesh. Afterwards the triangle quality of a mesh is discussed.

In the second part the basic version of the simplification process used in this thesis is introduced, followed by the description of the simplification algorithm of Lindstrom and Turk. In the last part a metric is introduced that is used to measure the error introduced by the simplification process.

2.1 Meshes

The first part of this section is dedicated to the formal definition of a mesh, and the introduction of necessary terminology which is used in the remainder of this thesis. This section ends with a short discussion on the quality of meshes and their elements.

Simple geometries like, planes or spheres, are often used to prove the correctness of newly adapted simulation processes, since it is easy to verify the results, because an analytic solution can be calculated. However, typical geometries created by process TCAD have a more complex structure, were no simple analytic solution can be calculated. This introduces the need to find discrete representations of such geometries.

2.1.1 Formal Definition

In simplest terms a mesh can be described as a subset of $\mathbb{R}^n$ that consists of simple connected elements of this subset. The terms simple, connected, and elements will later be specified, at first the intuition of these terms should suffice. The remainder of this section is dedicated to the formal definition of this intuition [28].

2.1.1 Definition
If a set $\emptyset \neq X \subset \mathbb{R}^n$ satisfies

1. X is piecewise connected
   $X = \bigcup_{i=1}^{l} X_i$, with $X_i \cap X_j \neq \emptyset$, $\forall i,j \in \{1,\dots,l\}$, and $l < \infty$,
2. every point $\bar{x} \in X$ has a neighbourhood based on the topology $\tau_n|_X$ which is homeomorph (see remark) to either $\{\bar{y} \in \mathbb{R}^n : \|\bar{y}-\bar{x}\| < r\}$ or $\{\bar{y} \in \mathbb{R}^n : \|\bar{y}-\bar{x}\| < r \wedge \bar{y} \geq \bar{0}\}$,

then X is called a **k-manifold**.

Let X be a k-manifold then the dimension of X is defined as $\mathrm{DIM}(X) = k$.

The set of all k-manifolds of $\mathbb{R}^n$ is denoted by $\mathfrak{M}_k^n$ and $\mathfrak{M}^n = \bigcup_{k=0}^{n} \mathfrak{M}_k^n$.

Remark: Two sets X, Y are said to be **homeomorph** if there exists a function $f : X \to Y$ that is one-to-one, continuous, and f^{-1} is continuous.

Example: An example of a manifold is an atlas. Each page of the atlas is a subset of $\mathbb{R}^2$ which can be mapped homeomorphically onto a sphere in $\mathbb{R}^3$ and all pages in the atlas envelope the sphere.

Before we take the next step and describe certain subsets of a k-manifold that will become a mesh we need to define the relative interior and the relative boundary of a k-manifold.

2.1.2 Definition
Let X be a k-manifold, then the **relative interior** $\mathrm{int}_k^\star(X)$ of X is defined as the set of $\bar{x} \in X$ which have a neighbourhood based on the topology $\tau_n|_X$ which is homeomorph to $\{\bar{y} \in \mathbb{R}^n : \|\bar{y} - \bar{x}\| < r\}$.

The **relative boundary** of X is defined as

$$\mathrm{bnd}_k^\star(X) := X \backslash \mathrm{int}_k^\star(X).$$

For the investigations in this work special subsets of a k-manifold are considered.

2.1.3 Definition
Let $\mathcal{E} \subset \mathfrak{M}^n$ be a non-empty subset, then $\mathcal{E}$ is called an **element space** and a set $E \in \mathcal{E}$ is called an **element**.

In the next step the elements of an element space are categorized, which helps in discussing the properties of all elements in a certain category.

2.1.4 Definition
Let $\mathcal{E}$ be an element space, then the maximum dimension of all its elements $\mathrm{DIM}_{\mathrm{cell}}(\mathcal{E}) := \max_{E \in \mathcal{E}}\{\mathrm{DIM}(E)\}$ is called the **cell dimension**.

An element is called a **facet** if $\mathrm{DIM}_{\mathrm{facet}}(E) = \mathrm{DIM}_{\mathrm{cell}}(\mathcal{E}) - 1$.

The set of all elements of dimension k in $\mathcal{E}$ is denoted by $\mathrm{elem}_k(\mathcal{E})$.

The elements in $elem_0(\mathcal{E})$ are called **vertices**.

The goal in this section is to describe geometries as the union of facets. To guarantee that facets do not intersect we expand the definition of an element space.

2.1.5 Definition
An element space $\mathcal{E}$ is called **face complete**, if for every element $E \in \mathcal{E}$ the union of all facets of E is equal to the relative boundary of E and the intersection of the relative interior of two different facets of E is empty.

Before the next property an element space can have is considered, the following fact has to be discussed: A facet of dimension k can be considered as a cell that has its own facets of dimension $k-1$. This process can be continued till the element becomes a vertex. The union of all those facets of facets is called a **face**.

2.1.6 Definition
Let $\mathcal{E}$ be an element space, then the **co-faces** of an element $E \in \mathcal{E}$ are all elements for which E is a face

$$\mathrm{cofaces}_{\mathcal{E}} := \{F \in \mathcal{E} | E \in \mathrm{faces}_{\mathcal{E}}(F)\},$$

with

$$\mathrm{faces}_{\mathcal{E}}(E) := \{E\} \cup \bigcup_{F \in \mathrm{facets}_{\mathcal{E}}(E)} \mathrm{faces}_{\mathcal{E}}(F)$$

With the current definition it is possible that the intersection of two elements creates subsets that are not part of the element space, e.g., two triangles that lie in the same plane and intersect each other in such a way that the intersection of these two triangles is a rectangle that is not an element of the element space. This leads to the following extension of the definition of an element space:

2.1.7 Definition
Let $\mathcal{E}$ be an element space. If for any two sets $E_1, E_2 \in \mathcal{E}$, their intersection $E_1 \cap E_2$ is either empty or a face of both, then the element space is called an **element complex**.

By uniting all the definitions given above a mesh can be defined as follows.

2.1.8 Definition
Let $\mathcal{M} \subset \mathcal{E}$ be a subset of an element complex $\mathcal{E}$, then $\mathcal{M}$ is called a **mesh** if

1. $\mathcal{M}$ is finite,
2. $\mathcal{M}$ is face complete in $\mathcal{E}$,
3. $\mathcal{M}$ has no dangling elements, meaning that every element $E \in \mathcal{M}$ is either a cell or E has at least one co-face cell.

Since not all subsets of $\mathbb{R}^n$ are suitable elements of a mesh, a set of subsets of $\mathbb{R}^n$ has to be described that satisfies the above definition.

In this thesis a special class of meshes is considered: triangle meshes. To formally define the elements from which a triangle mesh is built a set of special subsets of $\mathbb{R}^n$ has to be defined.

2.1.9 Definition
A set $X \subset \mathbb{R}^n$ is called **convex** if for every two points $\bar{x}_1, \bar{x}_2 \in X$ there exists a point that satisfies $\lambda\bar{x}_1 + (1-\lambda)\bar{x}_2$, with $0 < \lambda < 1$.

The **convex hull** of a set X is defined as

$$\text{conv} := \bigcap_{\bar{x} \subset K, K \text{ Convex}} K.$$

To construct a set of minimal convex elements in $\mathbb{R}^n$ defined by some points we define an affine combination.

2.1.10 Definition
Let $X = \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k\} \subset \mathbb{R}^n$ be a set of points in $\mathbb{R}^n$, then we call the linear combination $\sum_{i=1}^{k} \omega_i \bar{x}_i$ with $\omega_i \in \mathbb{R}$ and $\sum_{i=1}^{k} \omega_i = 1$ an **affine combination** of the set of points X.

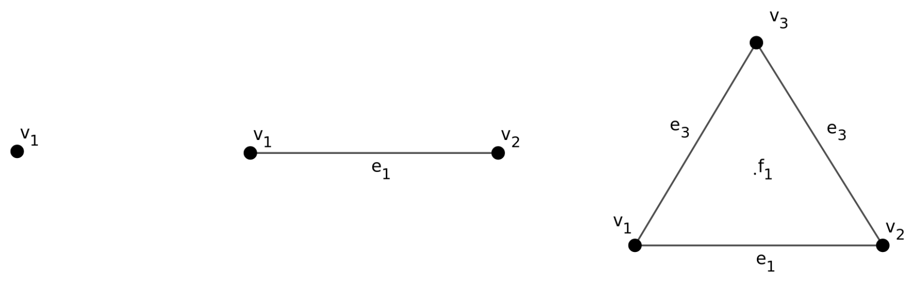
If there exist no weights ω_i such that X is an affine combination the set is called **affinely independent**.

2.1.11 Definition
Let $X = \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_{k+1}\}$ be a set of affinely independent points then

$$\text{simplex}(X) = \text{conv}(X)$$

is called a **k-simplex**.

In Figure 2.1 the first three types of a simplex are depicted. These are the building blocks of a triangle mesh. For the remainder of this work, a mesh is considered to be a subset of $\mathfrak{M}^3$ with its elements being 0- to 2-simplices.



(a) 0-simplex (vertex) (b) 1-simplex (edge) (c) 2-simplex (triangle)

Figure 2.1: First three types of a simplex

Remark: An example for a non-simplex mesh is a quad meshes [1].

2.1.2 Triangle Quality

This section gives a brief introduction into triangle quality metrics. The shape of mesh elements is a very important metric for the convergence of boundary value problems, one can say that equilateral triangles have a 'good' shape for the stability of boundary value problems [33]. Many different functions have been proposed to measure the quality of elements in a mesh [24]. The measures used in this thesis are introduced.

Let $T = ABC$ be a non-degenerate triangle, the vertices of T are A, B, C. The angle at A is α, at B is β, and at C is γ. The edges are $a = BC$, $b = AC$, and $c = AB$. The inscribed circle is denoted by r and the circumscribed circle as R. The minimum and maximum Angle of T are denoted by

$$\begin{aligned}\theta_0 &= \min(\alpha, \beta, \gamma),\\ \theta_\infty &= \max(\alpha, \beta, \gamma).\end{aligned}$$

For this work the following two metrics are used:

- The **minimum angle** θ_0 measures if T is a Needle, needle is a triangle with one angle close to 0.
- The **radius ratio** $\frac{R}{2r}$ measures how close T is to being a right triangle.

2.2 Surface Simplification

In this thesis the edge decimation or edge collapse algorithm is used for the simplification process [7]. This method requires a mesh that represents a manifold and generally preserves the topology of the mesh. Furthermore, this method does not need re-triangulation after an edge is removed.

2.2.1 Edge Collapse

At first a basic version of the edge collapse algorithm is discussed. Let in this section $E_{\mathcal{M}}$ be the set of all edges in the mesh $\mathcal{M}$, and e an edge in $\mathcal{M}$.

The simplification process uses three functions:

- a weight function $w(e) : E_{\mathcal{M}} \to \mathbb{R}$ which calculates the weight of an edge,
- a placement function $p(e) : E_{\mathcal{M}} \to \mathbb{R}^3$ which calculates the position of a new vertex,
- and a termination function $t : \mathcal{M} \to \{0, 1\}$ which decides if the simplification process has to stop.

Example: An example for a simple weight function is the length of the edge e, a simple placement function chooses the midpoint of the edge e, and a simple termination function terminates the simplification process when a certain number of edges from the mesh have been removed.

Now that all functions needed for the simplification process are defined, the process itself can be described:

First the weight of each edge in the mesh is calculated and the values are stored in a priority queue. Then the edge with the lowest weight in the queue is chosen. The chosen edge and all edges adjacent to one of the two vertices that make up the chosen edge are deleted. Afterwards, the placement function calculates a position were the new vertex is placed. Then all surrounding vertices are connected to the new vertex (see Figure 2.2). The weight of each of the new edges is calculated and they are added to the priority queue. Finally, the termination function is called which decides if the process starts over again with the next edge at the front of the priority queue or if the simplification process stops.

After a successful edge collapse two faces, four edges, and one vertex are removed from the mesh (see Figure 2.2).

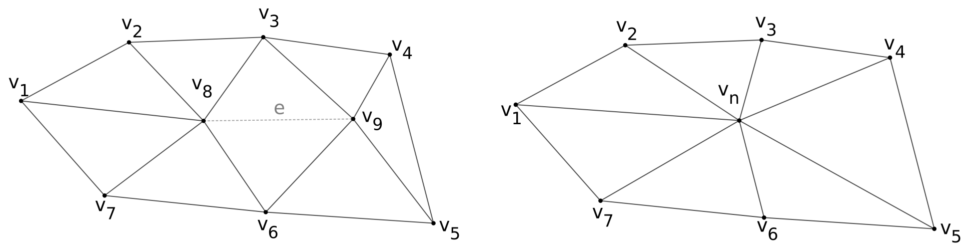


Figure 2.2: Example of an edge collapse of e into the vertex v_n.

2.2.2 Lindstrom-Turk Simplification

The Lindstrom-Turk simplification process [17] tries to preserve the original geometry of a mesh by minimizing the difference in volume between the original and the simplified mesh. Therefore, a system of three linearly independent constraints is created for each edge in the mesh. This system can be interpreted as the intersection of three linearly independent planes in a three-dimensional space.

Solving this system yields the optimal vertex position with respect to the given constraints.

A **tetrahedron** is a 4-simplex. The Lindstrom-Turk simplification process uses the volume of tetrahedrons to formulate two of the four constraints. To avoid ambiguities it is mentioned here that tetrahedrons are used in volume meshing as elements of a mesh. However, for the purposes of this work tetrahedrons are not part of the mesh and are only used in the formulation of the constraints.

Each of the constraints is a quadratic optimization problem, which is solved in the least squared sense. During the Lindstrom-Turk simplification four different constraints are considered

- *Boundary Preservation*
  If a boundary edge is removed the simplification process tries to place the new vertex v in such a way that it minimizes the change in area between the triangle that is deleted and the new triangles that replace it.
- *Volume Preservation*
  When a non-boundary edge e is removed from the mesh the new vertex v creates a tetrahedron with each triangle adjacent to e_{v_0} and e_{v_1}. This constraint tries to minimize the sum of the volumes of all the tetrahedrons that are created by the edge collapse.
- *Volume Optimization*
  Since in general the solution of the above constraint is not unique, this constraint tries to minimize the volume of each individual tetrahedron.
- *Triangle Shape Optimization*
  In some cases the constraints above do not yield a single solution, for example, if the vertices are coplanar, meaning all vertices are in the same plane. In this case the shape of the triangles is optimized to favour equilateral triangles.

The constraints are considered in the above order and are either accepted or discarded if they are incompatible with the given edge. When three constraints are found the system of constraints is solved and the new position of the vertex is calculated. The weight of an edge is calculated by the weighted sum of all the optimization terms, the weights can be chosen manually.

2.2.3 Distance Between Meshes

Simplifying a mesh reduces its complexity and thus changes its geometry. A commonly used metric to determine the error introduced by simplifying a mesh is the *Hausdorff distance* between two meshes [35].

2.2.12 Definition
Let $X, Y \subset \mathbb{R}^3$ be two sets. Then the **one-sided Hausdorff distance** is defined as follows:

$$d_{H'}(X,Y) := \max_{\bar{p}\in X}[\min_{\bar{q}\in Y}[||\bar{p}-\bar{q}||]].$$

The one-sided Hausdorff distance is not symmetric, meaning that $d_{H'}(X, Y)$ is not necessarily equal to $d_{H'}(Y, X)$. When comparing the distance between two objects it is useful to make it independent of the order in which the objects are compared. To make the one-sided Hausdorff distance symmetric both distances are calculated and the maximum is chosen:

2.2.13 Definition
Let $X, Y \subset \mathbb{R}^3$ be two sets. Then the **Hausdorff distance** is defined as:

$$d_H(X, Y) := \max\{d_{H'}(X, Y), d_{H'}(Y, X)\}.$$

Chapter 3

Mathematical Foundations

The simplification algorithm developed in this work uses the curvature of a vertex to detect the complexity of the local geometry of a mesh. To define the curvature of a vertex, the mathematical concept of a surface has to be introduced.

The first part of this chapter discusses surfaces in general, and proofs that parts of a surface can be represented by a differentiable functions. This function can then be used to calculate several properties of the surface, one of these properties is the curvature in a point on the surface $\mathbf{x} \in S$ with $\mathbf{x} \in \mathbb{R}^3$, see Definition 3.1.2 .

The second half of this chapter introduces two methods to approximate the curvature of the vertices in a mesh, as, in general parts, of a mesh cannot be expressed as a differentiable function.

3.1 Differential Geometry

This section introduces some basic concepts from differential geometry. At first surfaces in $\mathbb{R}^3$ are discussed and how they can be described in a global and local context. Further, it is shown that parts of a surface can locally be expressed as a linear map [5].

3.1.1 Surfaces

Before we define a surface, we have to introduce parametrized curves, helping us in defining some properties of surfaces.

3.1.1 Definition
A **regular curve parametrized by arc length** is a differentiable map $\alpha : I \to \mathbb{R}^3$, were $I = (a, b) \subset \mathbb{R}$ and $|\alpha(s)'| = 1 \ \forall s \in I$.

The condition $|\alpha(s)'| = 1 \ \forall s \in I$ guarantees the existence of a tangent line at $\alpha(s)$.

Let $\alpha(s)$, $s \in (a, b)$ be a regular curve parametrized by arc length. If $|\alpha''(s)| = 0$ then α defines a straight line in s, so if $|\alpha''(s)| \neq 0$ there exists a unit normal vector $n(s)$ in the direction of $\alpha''(s)$ which is well defined by $n(s) = \frac{\alpha''(s)}{|\alpha''(s)|}$. Furthermore, $n(s)$ is normal to $\alpha'(s)$ because if we differentiate $\alpha'(s) \cdot \alpha'(s) = 1$ we get $\alpha''(s) \cdot \alpha'(s) = 0$, so we are justified in calling it the **normal vector** to α at the point s.

For the sake of convenience we call regular curves parametrized by arc length *parametrized curves* or *curves* throughout this thesis.

3.1.2 Definition
A subset $S \subset \mathbb{R}^3$ is a regular surface if, for each $p \in S$, there exists a neighbourhood V in $\mathbb{R}^3$ and a map $\mathbf{x} : U \to V \cap S$ of an open set $U \in \mathbb{R}^2$ onto $V \cap S \subset \mathbb{R}^3$ such that

1. $\mathbf{x}$ is differentiable,
2. $\mathbf{x}$ is a homomorphism,
3. (Regularity condition) For each $q \in U$, the differential $d\mathbf{x}_q : \mathbb{R}^2 \to \mathbb{R}^3$ is injective.

Example: The unit sphere defined by

$$\{(x, y, z \in \mathbb{R}^3; x^2 + y^2 + z^2 = 1\}$$

is a regular surface [5].

The map $\mathbf{x}_p(u, v) = (x(u, v), y(u, v), z(u, v))$ is called a **parametrization of the regular surface** in the neighbourhood of p.

The regularity condition will be very important, so let us bring it into a more familiar form. For this purpose let us explicitly express the Jacobi-Matrix of $d\mathbf{x}(u, v)$ in the standard basis $\bar{e}_1 = (1, 0)$, $\bar{e}_2 = (0, 1)$, in the coordinates (u_0, v_0), and $\bar{f}_1 = (1, 0, 0)$, $\bar{f}_2 = (0, 1, 0)$, $\bar{f}_3 = (0, 0, 1)$, in the coordinates (x, y, z).

Let us consider two curves u_{curve} and v_{curve} that intersect in the point $p = (u, v)$, with tangent vectors $\bar{e}_1$ and $\bar{e}_2$. The derivatives along these vectors are

$$d\mathbf{x}_p(\bar{e}_1) = \left(\frac{\partial x}{\partial u}, \frac{\partial y}{\partial u}, \frac{\partial z}{\partial u}\right),$$
$$d\mathbf{x}_p(\bar{e}_2) = \left(\frac{\partial x}{\partial v}, \frac{\partial y}{\partial v}, \frac{\partial z}{\partial v}\right).$$

The linear map of $d\mathbf{x}_p$ is given by

$$d\mathbf{x}_p = \begin{pmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \\ \frac{\partial z}{\partial u} & \frac{\partial z}{\partial v} \end{pmatrix}. \tag{3.1}$$

The regularity condition guarantees that the vectors $d\mathbf{x}_p(\bar{e}_1)$ and $d\mathbf{x}_p(\bar{e}_2)$ are linear independent, or in other words at least one of the Jacobi determinants

$$\frac{\partial(x, y)}{\partial(u, v)} = \begin{vmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \end{vmatrix}, \frac{\partial(y, z)}{\partial(u, v)}, \frac{\partial(x, z)}{\partial(u, v)},$$

is not zero.

We now discuss a special class of regular surfaces, namely surfaces that can be described as the graph of a function $z = f(x, y)$. We use this finding to motivate further investigations into local properties of surfaces.

3.1.3 Theorem
Let U be an open subset of $\mathbb{R}^2$ and $f : U \to \mathbb{R}$ a differentiable function.
Then the graph of f, which is defined as

$$(u, v, f(u, v)) \subset \mathbb{R}^3 \text{ for } (u, v) \in U,$$

is a regular surface.

Proof :
We only have to show that a map $\mathbf{x} : U \to \mathbb{R}^3$ given as

$$\mathbf{x}(u, v) = (u, v, f(u, v))$$

satisfies the three conditions of definition 3.1.2.
It is clear that condition 1 holds. For the second condition we see that each point (x, y, z) of the graph is an image under $\mathbf{x}$ of the unique point $(u, v) = (x, y) \in U$. Therefore, $\mathbf{x}$ is one to one, and since $\mathbf{x}^{-1}$ is the restriction to the graph of f, the projection of $\mathbb{R}^3$ onto the xy-plane, $\mathbf{x}^{-1}$ is continuous. To show that condition 3 holds we calculate $d\mathbf{x}$ as seen in equation (3.1) yielding

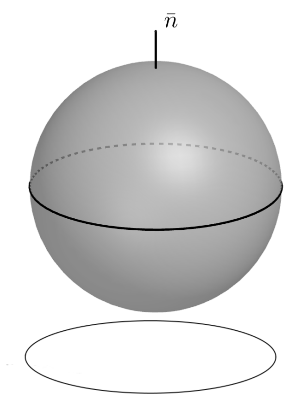
$$d\mathbf{x} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ \frac{\partial f}{\partial u} & \frac{\partial f}{\partial v} \end{pmatrix}. \tag{3.2}$$

We have shown that $\frac{\partial(x,y)}{\partial(u,v)} = 1$. ■

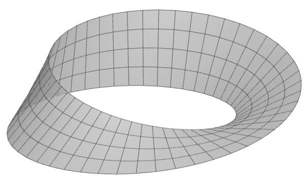
In the above theorem we see that some regular surfaces can be described globally as the graph of a differentiable function. This does not hold for all regular surfaces. Take, for example, the unit sphere. It does not matter if we choose the xy-, xz-, or yz-plane as the coordinate plane, because there cannot exist a map that has two different function values for the same parameter. However, if we choose the north pole of the unit sphere and look at an area around that point, for example, the northern hemisphere (Figure 3.1(a)), we see that it is possible to select a point and an appropriate neighbourhood around that point to describe at least part of the sphere as the graph of a differentiable function. In the next theorem we show that there always exists an area around a point on any given surface that can be described as the graph of a differentiable function.

3.1.4 Theorem
Let $S \subset \mathbb{R}^3$ be a regular surface and $p \in S$. Then there exists a neighbourhood V of p in S such that V is the graph of a differentiable function which has one of the following three forms:

$$z = f(x, y),\ y = g(x, z),\ x = h(x, z).$$



(a) Parametrization of the northern hemisphere of the sphere, which can be represented as the graph of a differentiable function with values in its projection plane.



(b) Möbius strip, a parametrized surface which is non orientable.

Figure 3.1: Examples of parametrized surfaces.

Proof :
Let $\mathbf{x} : U \subset \mathbb{R}^2 \to S$ be a parametrization of S in p and write $\mathbf{x}(u,v) = (x(u,v), y(u,v), z(u,v))$, $(u,v) \in U$. By the third condition of definition 3.1.2, one of the Jacobi determinants

$$\frac{\partial(x,y)}{\partial(u,v)}, \frac{\partial(y,z)}{\partial(u,v)}, \frac{\partial(x,z)}{\partial(u,v)}$$

is not zero at $\mathbf{x}^{-1}(p) = q$.

Suppose first that $\frac{\partial(x,y)}{\partial(u,v)}(q) \neq 0$ and consider the map $\pi \circ \mathbf{x} : U \to \mathbb{R}^2$, were π is the projection $\pi(x,y,z) = (x,y)$. Then $\pi \circ \mathbf{x}(u,v) = (x(u,v), y(u,v))$ and since $\frac{\partial(x,y)}{\partial(u,v)}(q) \neq 0$ we can apply the inverse function theorem to guarantee the existence of neighbourhoods V_1 of q, V_2 of $\pi \circ \mathbf{x}(q)$ such that $\pi \circ \mathbf{x}$ maps V_1 diffeomorphically onto V_2. It follows that π restricted to $\mathbf{x}(V_1) = V$ is one to one and that there is a differentiable inverse $(\pi \circ \mathbf{x})^{-1} : V_2 \to V_1$. Observe that, since $\mathbf{x}$ is a homeomorphism, V is a neighbourhood of p in S. Now, if we compose the map $(\pi \circ \mathbf{x})^{-1} : (x,y) \to (u(x,y), v(x,y))$ with the function $(u,v) \to z(u,v)$, we find that V is the graph of the differentiable function $z = z(u(x,y), v(x,y)) = f(x,y)$, and this settles the first case. The remaining cases can be treated in the same way. ■

This theorem allows us to express local properties of a given discrete surface patch without the need to consider the whole surface.

We now give a definition of the tangent plane of a surface. A formal proof can be found in [5]. When we talk about a tangent vector in the next definition, we mean the tangent vector of a parametrized curve $\alpha : (-\varepsilon, \varepsilon) \to S$ with $\alpha(0) = p$ in 0.

3.1.5 Definition
Let S be a surface and p a point on S then the set of all tangent vectors in p create a 2-dimensional subspace which we call the **tangent plane** of S in the point p (short $T_p(S)$).

Let $\mathbf{x}$ be a fixed parametrization of S and $T_p(S)$ the tangent plane in p. If we calculate the partial derivatives of $\mathbf{x}$ in the directions of u and v, we get two vectors $\bar{x}_u, \bar{x}_v \in T_p(S)$ which build a basis of $T_p(S)$. Further, we know from linear algebra that the cross product $\bar{x}_u \times \bar{x}_v$ yields a vector that is orthogonal to $\bar{x}_u$ and $\bar{x}_v$. By normalizing this vector we get the following definition:

3.1.6 Definition
Let $\mathbf{x}(u, v) : U \subset \mathbb{R}^2 \to S$ be a parametrization of a surface. Then the **normal vector** of the surface in the point p is defined as

$$\bar{n} = \frac{\mathbf{x}_u \times \mathbf{x}_v}{|\mathbf{x}_u \times \mathbf{x}_v|}(p).$$

3.1.7 Definition
A regular surface S is called **orientable** if it is possible to cover it with a family of coordinate neighbourhoods in such a way that if a point $p \in S$ belongs to two neighbourhoods of this family, then the change of coordinates has positive Jacobian at p. The choice of such a family is called an orientation of S, and S, in this case, is called oriented. If such a choice is not possible, the surface is called non orientable.

Remark: Not all regular surfaces are globally orientable, see for example, the Möbius strip (Figure 3.1(b)) [5].

To describe an intuitive approach to orientability let p be a point on a surface S and $\bar{n}$ the normal vector in p. If we now choose a parametrized curve $\beta : (a, b) \to S$ on S, $s \in (a, b)$, and move $\bar{n}$ along this curve, each time $\beta(s) = p$ the choice of the normal vector has to be the same. This is, for example, impossible on the Möbius strip. The positive Jacobian guarantees us that the coordinate change from one tangent plane into the next does not change the orientation of $T_p(S)$.

With the help of the next theorem and theorem 3.1.4 we see that each surface can be oriented locally. This theorem guarantees locally well defined surface properties.

3.1.8 Theorem
Let S be a regular surface defined by the graph of a differentiable function $z = f(x, y)$ with $(x, y) \in U \subset \mathbb{R}^2$ open. Then S is orientable.

Proof :
We can parametrize S as $\mathbf{x}_p = (x, y, f(x, y))$, if we calculate the matrix $d\mathbf{x}_p$ (see Equation (3.2)). It is clear that we have Jacobian 1 in each p which makes S orientable.

■

3.1.2 The Weingarten Map

In this section the Gauss map is defined which is used to describe the Weingarten map. With the latter specific geometric properties of a given surface can be expressed.

3.1.9 Definition
Let S be a regular surface with an orientation N. The map $N : S \to \mathbb{R}^3$ takes its values in the unit sphere

$$S^2 = \{(x, y, z) \in \mathbb{R}^3; x^2 + y^2 + z^2 = 1\}.$$

The map $N : S \to S^2$ is called the **Gauss map** of S.

To describe the functionality of the Gauss map in a different way let us consider a surface and each normal vector in each point. The Gauss map maps the shaft of these normal vectors into the center of the unit sphere (Figure 3.2). The direction of these normal vectors describe points on the surface of the unit sphere and the union of these points describe a curve.

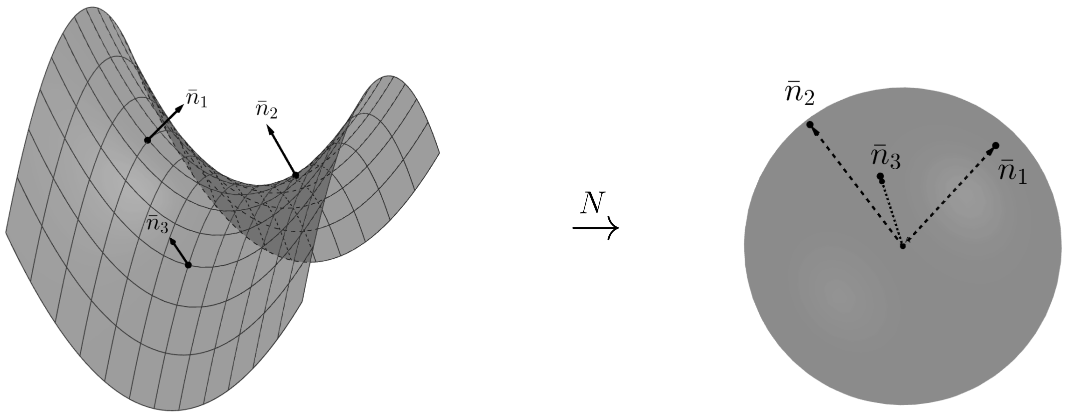


Figure 3.2: Example of three normal vectors on a hyperbolic paraboloid (saddle) when the gauss map is used on the hyperbolic paraboloid.

With knowledge of the Gauss map we can define the Weingarten map.

3.1.10 Definition
The negative derivative of the Gauss map

$$\omega := -dN_p$$

is called the **Weingarten map**.

At first glance this definition looks arbitrary, but in the next theorem we show that the Weingarten map is linear and self-adjoint which allows us to use many results from linear algebra.

3.1.11 Theorem
The Weingarten map ω is a self-adjoint linear map.

Proof :
It is obvious that the Weingarten map is a linear map, so let $\{\mathbf{x}_u, \mathbf{x}_v\}$ be a given basis associated with $T_p(S)$. Let $\alpha(t) = \mathbf{x}(u(t), v(t))$ be a parametrized curve in S, with $\alpha(0) = p$, if we use $-\omega$ on this curve we get

$$-\omega(\alpha'(t)) = -\omega(\mathbf{x}_u \cdot u'(t) + \mathbf{x}_v \cdot v'(t)).$$

If we look at the above equation at the point p and at $t = 0$ we get

$$-\omega_p(\alpha(0)) = \bar{n}_u \cdot u'(0) + \bar{n}_v \cdot v'(0),$$

which implies that $-\omega(\mathbf{x}_u) = \bar{n}_u$ and $-\omega(\mathbf{x}_v) = \bar{n}_v$.

We want to show that

$$\mathbf{x}_u \omega(\mathbf{x}_v) = \omega(\mathbf{x}_u)\mathbf{x}_v \Leftrightarrow -\mathbf{x}_u \cdot \bar{n}_v = -\mathbf{x}_v \cdot \bar{n}_u. \tag{3.3}$$

$\forall u$ $\mathbf{x}_u \in T_p(S)$ it follows that $\mathbf{x}_u \cdot \bar{n} = 0$, so

$$\frac{d}{dv}(\mathbf{x}_u \cdot \bar{n}) = \mathbf{x}_{uv} \cdot \bar{n} + \mathbf{x}_u \cdot \bar{n}_v = 0 \Leftrightarrow \mathbf{x}_{uv} \cdot \bar{n} = -\mathbf{x}_u \cdot \bar{n}_v. \tag{3.4}$$

$\forall v$ $\mathbf{x}_v \in T_p(S)$ it follows that $\mathbf{x}_v \cdot \bar{n} = 0$, so

$$\frac{d}{du}(\mathbf{x}_v \cdot \bar{n}) = \mathbf{x}_{vu} \cdot \bar{n} + \mathbf{x}_v \cdot \bar{n}_u = 0 \Leftrightarrow \mathbf{x}_{vu} \cdot \bar{n} = -\mathbf{x}_v \cdot \bar{n}_u. \tag{3.5}$$

If we use Schwarz's theorem either on equation (3.4) or (3.5) we get

$$-\mathbf{x}_u \cdot \bar{n}_v = \mathbf{x}_{uv} \cdot \bar{n} = -\mathbf{x}_v \cdot \bar{n}_u,$$

which is equivalent to equation (3.3). ■

From linear algebra we know that self-adjoint linear maps have real eigenvalues and orthogonal eigenvectors. It also allows us to associate the map dN_p with a quadratic from II_p, which is called the **second fundamental form** [13].

Since II_p is a quadratic form it is obvious that it can be expressed as a matrix. We discuss now how we can express the matrix of the Weingarten map in a given basis. In the first part of the proof of theorem 3.1.11 we defined the parametrized curve $\alpha(t)$. Let us consider this curve in the point 0. Then the tangent vector to this curve is given by $\alpha'(0) = \mathbf{x}_u u' + \mathbf{x}_v v'$ and

$$dN(\alpha'(0)) = N'(u(t), v(t)) = \bar{n}_u \cdot u' + \bar{n}_v \cdot v'. \tag{3.6}$$

Since both vectors $\bar{n}_u$ and $\bar{n}_v$ lie in $T_p(S)$ they can be written as

$$\begin{aligned} \bar{n}_u &= a_{11}\mathbf{x}_u + a_{21}\mathbf{x}_v, \\ \bar{n}_v &= a_{12}\mathbf{x}_u + a_{22}\mathbf{x}_v. \end{aligned} \tag{3.7}$$

If we now substitute these findings into equation (3.6), we see that dN can be be described in the basis $\{\mathbf{x}_u, \mathbf{x}_v\}$ as the following matrix

$$dN \begin{pmatrix} u' \\ v' \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} u' \\ v' \end{pmatrix}.$$

We now calculate the matrix of dN in the basis $\{\mathbf{x}_u, \mathbf{x}_v\}$.

$$\begin{aligned} -II_p(\alpha') &= dN(\alpha')\cdot\alpha' = (\bar{n}_u u' + \bar{n}_v v')\cdot(\mathbf{x}_u u' + \mathbf{x}_v v') \\ &= (\bar{n}_u\cdot\mathbf{x}_u)(u')^2 + (\bar{n}_u\cdot\mathbf{x}_v)u'v' + (\bar{n}_v\cdot\mathbf{x}_u)u'v' + (\bar{n}_v\cdot\mathbf{x}_v)(v')^2 \\ &= e(u')^2 + 2fu'v' + g(v')^2 \end{aligned}$$

To obtain the values a_{ij} in terms of the coefficients e, f, and g, we are substituting equation (3.7) into our previous findings and get

$$\begin{aligned} -f = \bar{n}_u\cdot\mathbf{x}_v &= a_{11}\mathbf{x}_v^2 + a_{21}\mathbf{x}_u\cdot\mathbf{x}_v \\ &= a_{11}F + a_{21}G, \\ -f = \bar{n}_v\cdot\mathbf{x}_u &= a_{12}E + a_{22}F, \\ -e = \bar{n}_u\cdot\mathbf{x}_u &= a_{11}E + a_{21}F, \\ -g = \bar{n}_v\cdot\mathbf{x}_v &= a_{12}F + a_{22}G, \end{aligned}$$

which leads us to,

$$-\underbrace{\begin{pmatrix} e & f \\ f & g \end{pmatrix}}_{=II_p} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}\underbrace{\begin{pmatrix} E & F \\ F & G \end{pmatrix}}_{:=I_p}.$$

Remark: The matrix I_p is called the first fundamental form, which can be used, for example, to calculate the area of a given surface patch [5]. The coefficients are calculated as follows

$$\begin{aligned} E &= \mathbf{x}_u\cdot\mathbf{x}_u, \\ F &= \mathbf{x}_u\cdot\mathbf{x}_v, \\ G &= \mathbf{x}_v\cdot\mathbf{x}_v. \end{aligned}$$

After a short transformation we have found a way to express the Weingarten map as the matrix $(a_{i,j})$, $i,j = 1,2$ in the basis $\{\mathbf{x}_u, \mathbf{x}_v\}$

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = -\begin{pmatrix} e & f \\ f & g \end{pmatrix}\begin{pmatrix} E & F \\ F & G \end{pmatrix}^{-1}.$$

Now we discuss how we can calculate the Weingarten map of a surface that is described by a given differentiable function. We have seen in theorem 3.1.4 that we can express parts of a surface as a differentiable function. So let $z = h(u,v)$ be this differentiable function, with $(u,v)\in U\subset\mathbb{R}^2$ and U open. Then we can parametrize our surface as

$$\mathbf{x}(u,v) = (u, v, h(u,v)) \quad (u,v)\in U.$$

After a simple calculation we get

$$\mathbf{x}_u = (1, 0, h_u), \quad \mathbf{x}_v = (0, 1, h_v),$$
$$\mathbf{x}_{uu} = (0, 0, h_{uu}), \quad \mathbf{x}_{vv} = (0, 0, h_{vv}), \quad \mathbf{x}_{uv} = (0, 0, h_{uv}).$$

Thus, we can calculate the coefficients of the first fundamental form

$$\begin{aligned} E &= 1 + h_u^2, \\ F &= h_u h_v, \\ G &= 1 + h_v^2. \end{aligned} \tag{3.8}$$

To calculate the coefficients of the second fundamental form we have to calculate the unit normal field of the surface by

$$N(x, y) = \frac{(-h_x, -h_y, 1)}{\sqrt{1 + h_x^2 + h_y^2}}.$$

The coefficients are given by

$$\begin{aligned} e &= \frac{h_{xx}}{\sqrt{1 + h_x^2 + h_y^2}}, \\ f &= \frac{h_{xy}}{\sqrt{1 + h_x^2 + h_y^2}}, \\ g &= \frac{h_{yy}}{\sqrt{1 + h_x^2 + h_y^2}}. \end{aligned} \tag{3.9}$$

3.1.3 The Curvature of a Surface

This section starts with an intuitive approach for defining the curvature of a surface. Then this intuition is formalized with the help of the Weingarten map.

Let us consider a surface S and a point on this surface p_0. Let $\bar{n}_0$ be the normal vector in p_0 and $T_{p_0}(S)$ the tangent plane in p_0. Since $T_{p_0}(S)$ is a 2-dimensional vector space we can choose a basis with its origin in p_0. Let $\{\bar{b}_1, \bar{b}_2\}$ be this basis and let u, v be two parametrized curves that intersect in p_0 (Figure 3.3). If we now move $\bar{n}_0$ in the direction of $\bar{b}_1$ or $\bar{b}_2$ along one of the curves u or v we see that. The direction of the normal vector can change. Let S now be a plane we see that the direction of the normal vector does not change no matter were we move it. Moreover, if S is a sphere and we move the normal vector along one of the previously defined curves we see that the normal vector changes direction.

It is obvious that the direction of the normal vector can change in two directions. The rates at which the normal vector changes in these directions are called the principal curvatures κ_1 and κ_2 in the direction of $\bar{b}_1$ and $\bar{b}_2$.

To formalize our findings from above we recall that the Weingarten map has real eigenvalues k_1, k_2 and orthogonal eigenvectors $\bar{e}_1, \bar{e}_2$. If we choose $\bar{e}_1, \bar{e}_2$ as the orthonormal basis of the tangent plane $T_{p_0}(S)$ in the point p_0 we get $\omega(\bar{e}_1) = k_1 \bar{e}_1$ and $\omega(\bar{e}_2) = k_2 \bar{e}_2$.

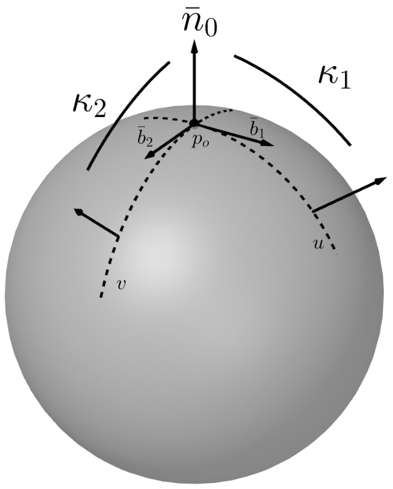


Figure 3.3: Principal curvatures κ_1 and κ_2 in a point p_0 on a sphere.

Since the second fundamental form is a quadratic form its eigenvalues k_1 and k_2 are the maximum and the minimum of the second fundamental form. This yields the following definition.

3.1.12 Definition
Let S be a regular surface and ω its Weingarten map in the point p_0. Then the eigenvalues $\kappa_1, \kappa_2 \in \mathbb{R}$ of ω are called the **principle curvatures** of S in p_0.

We recall from linear algebra that eigenvalues, eigenvectors, trace, and determinant are invariant to basis transformations. So we know that these properties of the Weingarten map are independent of the basis we consider the map in.
We use this to further consider the study of the determinant and trace of the Weingarten map. If we calculate the determinant of the Weingarten map we get

$$\det(\omega) = k_1 k_2. \tag{3.10}$$

Since the orientation of the surface does not change the determinant, we can define the Gaussian curvature.

3.1.13 Definition
Let κ_1 and κ_2 be the principal curvatures of a surface $S \subset \mathbb{R}^3$ in the point $x \in \mathbb{R}^3$. Then

$$K = \kappa_1 \cdot \kappa_2$$

is called the **Gaussian curvature** of S in the point x

Remark: Surfaces with Gaussian curvature 0 in all points are called developable surfaces.

Let us now consider the trace of the Weingarten map.

$$\operatorname{trace}(\omega) = k_1 + k_2 \tag{3.11}$$

We know that the orientation of the surface does not impact the determinant, but it changes the sign of the trace.

3.1.14 Definition
Let κ_1 and κ_2 be the principal curvatures of a surface $S \subset \mathbb{R}^3$ in the point $x \in \mathbb{R}^3$. Then

$$H = \frac{\kappa_1 + \kappa_2}{2}$$

is called the **mean curvature** of S in the point x

Remark: Surfaces with mean curvature 0 in all points are called minimal surfaces.

For the last part of this section we define $\bar{v} \in T_{p_0}(S)$ with $\bar{v} \neq 0$ and $i = 1, 2$.
We now discuss how to calculate the principal curvatures of a point from its mean and Gaussian curvature. Since $-k_1$ and $-k_2$ are the eigenvalues of the negative Weingarten map $-\omega$, they satisfy the equation

$$-\omega(\bar{v}) = -k_i \bar{v} = -k_i I \bar{v},$$

were I is the identity map. Therefore, the determinant is zero.

$$\det(-\omega(\bar{v}) + k_i I) = 0$$

Thus, k_1 and k_2 satisfy the following quadratic equation

$$k_i^2 + k_i \underbrace{(a_{11} + a_{22})}_{=-2H} + \underbrace{a_{11}a_{22} - a_{21}a_{12}}_{=K} = 0,$$

which can be rewritten into the following quadratic equation

$$k_i^2 - 2Hk_i + K = 0$$

Solving this quadratic equation we get

$$k_{1,2} = H \pm \sqrt{H^2 - K}. \tag{3.12}$$

This is another way to express the principal curvatures.

3.2 Curvatures of Discrete Surfaces

After introducing surfaces and their curvatures we turn our attention to meshes. In the previous section the curvature of a surface was defined with respect to a differentiable representation, the parametrization, of this surface. Meshes are, in general, not a differentiable function, e.g., along an edge that connects two triangles, which do not lie in the same plane, the mesh is not differentiable. So a method has to be found that approximates a subset of the mesh, e.g., a vertex and all elements adjacent to it, with a differentiable representation of this part of the mesh. In this section two methods of calculating the curvature of a mesh are discussed, the **jet-fitting** method (see Section 3.2.1) and the **spatial averaging** method (see Section 3.2.2). This section only discusses the mathematical background of these methods. The results of the evaluation studies concerning these two methods are presented in Section 5.1.

3.2.1 Differential Quantities through Osculating Jets

The first method that is introduced interpolates a given surface patch and reconstructs its Weingarten map in order to extract the principal curvatures of a given point.

Since a mesh does not only store point data, but also information about the connectivity between points (edges) it stores additional information about the surface. This method does not need information about the connectivity between points, because it is solving a quadratic minimization problem that only uses point data, so it can be used on more general data sets.

3.2.1.1 n-Jets and Jet-Fitting

In this section a way of approximating the Taylor expansion of a surface, described by a point cloud is discussed [6].

We have shown in theorem 3.1.4 that any smooth part of a regular surface can be written as a differentiable function. We call such a function a **height function**. We can express such a height function in its Taylor expansion.

3.2.15 Definition
Let f be a height function, $x, y \in \mathbb{R}$ and $b, j, n, k \in \mathbb{N}$. Then

$$f(x,y) = J_{B,n} + \mathcal{O}(\|(x,y)\|^{n+1}), \tag{3.13}$$

with

$$J_{B,n} = \sum_{k=1}^{n} H_{B,k}(x,y), H_{B,k}(x,y) = \sum_{j=0}^{k} B_{k-j,j} x^{k-j} y^{j}, \tag{3.14}$$

is called an n-degree jet or **n-jet**.

The differential properties of an n-jet coincide with the the first n terms of the Taylor expansion of the surface. We say that the jet has a n order contact with the surface. The n-jet of a surface contains $N_n = 1 + 2 + 3 + \cdots + (n+1)$ terms since there are $i + 1$ monomials of degree i. As seen in equation (3.9) we need at least a 2-jet to obtain information about the curvature of a given surface.

The next challenge is finding a way to reconstruct an n-jet from a set of points. Let us assume we have a set of $N + 1$ points with $p_i = (x_i, y_i, z_i)$ and $i = 1, \dots, N + 1$. We assume without loss of generality that we want to calculate the n-jet of a given surface in the point $p' := p_{N+1}$, were the origin of the coordinate system is located at p'. In this thesis the point p' is the vertex for which the curvature should be calculated, and the points p_i the vertices adjacent to p'.

Remark: From a theoretical standpoint we only have to move the origin to the point p'. However, when implementing this algorithm on a computer system we have to consider the additional costs of calculating a transformation matrix $\mathbf{T}$ from world coordinates (x_w, y_w, z_w) into the fitting coordinate system, were the origin is p', (x_f, y_f, z_f).

We want to approximate an n-jet $J_{A,n}$ such that $\forall i \in 1, \dots, n$

$$f(x_i, y_i) = J_{A,n} + \mathcal{O}(\|(x_i, y_i)^{n+1}\|). \tag{3.15}$$

We approximate $J_{A,n}$ in the least squared sense, which gives us a minimization problem

$$\sum_{i=1}^{N} (J_{A,n}(x_i, y_i) - f(x_i, y_i))^2. \tag{3.16}$$

The formulation of this problem in terms of linear algebra is

$$\begin{aligned} A &= (A_{0,0}, A_{1,0}, A_{0,1}, \dots, A_{0,d})^t, \\ Z &= (z_1, z_2, \dots, z_N)^t, \\ M &= (1, x_i, y_i, x_i^2, \dots, x_i y_i^{n-1}, y_i^n)_{i=1,\dots,N}, \end{aligned}$$

were A is our solution vector containing the coefficients of the Taylor polynomials

$$J_{A,n} = A_{0,0} + A_{1,0}x + A_{0,1}y + \frac{1}{2}(A_{2,0}x^2 + 2A_{1,1}xy + A_{0,2}y^2) + \dots. \tag{3.17}$$

Z is the vector that holds the function values of $f(x_i, y_i) = z_i$ given by the set of points and M holds all Taylor polynomials up to degree n. The minimization problem (3.16) can be written with the rectangular $N \times N_n$ matrix M as $\min\|MA - Z\|_2$, which can be solved by using a singular value decomposition.

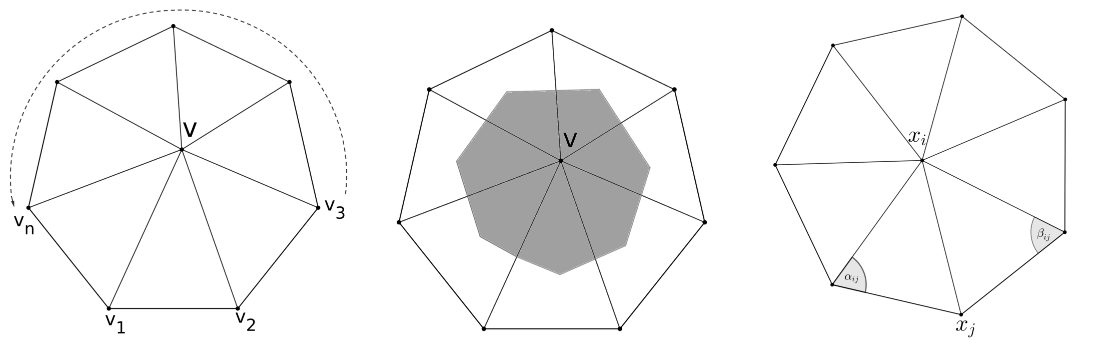
In Section 3.1.2 we have discussed how to calculate the Weingarten map of a two times differentiable function and in this section we have found a Taylor-approximation to a surface patch of at least order 2. By combining equations (3.17), (3.8), and (3.9) we get

$$\begin{aligned} E &= 1 + A_{1,0}^2 & e &= \frac{A_{0,2}}{\sqrt{1+A_{1,0}^2+A_{0,1}^2}}, \\ F &= A_{0,1}A_{1,0} & f &= \frac{A_{1,1}}{\sqrt{1+1,0^2+A_{0,1}^2}}, \\ G &= 1 + A_{0,1}^2 & g &= \frac{A_{0,2}}{\sqrt{1+1,0^2+A_{0,1}^2}}. \end{aligned}$$

After these calculations we have a matrix A which represents the Weingarten map of the surface patch defined by the given point cloud. To calculate the principal curvatures we only have to calculate its eigenvalues.

3.2.2 Discrete Differential-Geometry Operators

The second method uses the discrete version of two operators, the Laplace-Beltrami operator and the Gauss-Bennot theorem, to calculate the curvatures of a given vertex in a mesh [22]. This method does not need to transform points to different locations or solve a minimization problem. Additionally, all calculations needed are values of angles and the length of vectors, resulting in a much faster calculation time.



(a) 1-ring neighbourhood (b) Voronoi region (c) Voronoi region calculation

Figure 3.4: Definition of terms in the 1-ring neighbourhood of a vertex.

The drawbacks of this method are the need for additional information about the connectivity between points, which is given in a mesh, and the points that are de facto co-local, e.g., points that have nearly identical coordinates in double precision, will result in numerical errors. Both operators are integral operators so we have to find a suitable surface area around each vertex, on which we can apply these operators. From now on we call the set of all vertices incident to a vertex x_i the **1-ring neighbourhood of $\mathbf{x_i}$** ($N_1(x_i)$) (Figure 3.4(a)). Voronoi regions give a tight error bound for the discrete operators we will use [22].

The Voronoi region (Figure 3.4(c)), of a triangle is defined as

$$A_{\text{Voronoi}} = \frac{1}{8} \sum_{j \in N_1(x_i)} (\cot \alpha_{ij} + \cot \beta_{ij}) \|\mathbf{x}_i - \mathbf{x}_j\|^2. \tag{3.18}$$

Since we cannot guarantee that each triangle in an arbitrary mesh is non-obtuse we have to consider these special triangles in our area calculation. Both operators yield valid results even for 1-ring neighbourhoods, which only consist of obtuse triangles. Algorithm 1 shows how these special cases can be handled. The algorithm iterates over each triangle in the 1-ring neighbourhood and checks if it has an obtuse angle. If this is the case, the Voronoi region of the triangle is not well defined and it adds a fraction (see Algorithm 1) of the area of the triangle to the area of the 1-ring neighbourhood. Otherwise, it calculates the Voronoi region of the triangle.

3.2.2.1 Discrete Mean Curvature

This section describes how the the mean curvature normal operator of a point p_i is used, to calculate its mean curvature [9].

```
input  : 1-ring neighbourhood of the vertex $x$
output: $A_{\text{mixed}}$

$A_{\text{mixed}} = 0$;
foreach Triangle $T$ in the 1-ring neighbourhood of $x$ do
    if $T$ is non obtuse then
        // Voronoi formula see (3.18)
        $A_{\text{mixed}}$ += Voronoi region of $x$ in $T$;
    else
        if the angle of $T$ at $x$ is obtuse then
            $A_{\text{mixed}}$ += area($T$)/2;
        else
            $A_{\text{mixed}}$ += area($T$)/4;
        end
    end
end
```

Algorithm 1: Calculate the area A_{mixed}.

3.2.16 Definition
Let S be a regular surface and let p_i be a point on S, further let $\bar{n}_i$ be the normal vector in p_i. Then the **Mean Curvature Normal Operator** is defined as

$$\mathbf{H}(p_i) = 2H\bar{n}_i, \tag{3.19}$$

were H denotes the mean curvature of S in p_i.

The mean curvature normal operator in a point p_i can be considered in relation to an infinitesimal surface area A around p_i as

$$\mathbf{H}(p_i) = \lim_{\text{diam}(A)\to 0} \frac{\nabla A}{A}, \tag{3.20}$$

were diam(A) is the diameter of the surface patch A, and ∇A is the gradient with respect to the coordinates of p_i [9].

We want to find a way to calculate the mean curvature normal operator of the 1-ring neighbourhood of a point p_i. The gradient of the point p_i can be expressed as the integral of the mean curvature normal operator around the 1-ring neighbourhood [8]

$$\iint\limits_{A_{\text{mixed}}} H(p_i)dA = \nabla A_{\text{1-ring}}.$$

Therefore we need to find a way to calculate the value of $\iint\limits_{A_{\text{mixed}}} H(p_i)dA$. The mean curvature normal operator can be expressed as a Laplacian operator [9]. Thus, we get

$$\iint\limits_{A_{\text{mixed}}} H(p_i)dA = -\iint\limits_{A_{\text{mixed}}} \Delta_{u,v} du dv. \tag{3.21}$$

The operator above is called the Laplace-Beltrami operator.

3.2.17 Theorem The integral of the mean curvature normal operator in the point p_i on a triangle mesh can be expressed as

$$\iint\limits_{A_{\text{mixed}}} H(p_i) dA = \frac{1}{2} \sum_{j \in N_1(p_i)} (\cot \alpha_{ij} + \cot \beta_{ij})(\mathbf{x}_i - \mathbf{x}_j).$$

Proof :
Let $\mathbf{x}$ be a local parametrization of a surface S around a point p_i. We use equation (3.21) to calculate the value of the mean curvature normal operator. To do so, we need to calculate the integral of the Laplace operator with respect to our parametrization $\mathbf{x}$. By using Gauss's theorem we get

$$\iint\limits_{A_{\text{mixed}}} \Delta_{u,v} \mathbf{x} du dv = \int\limits_{\partial A_{\text{Mixed}}} \nabla_{u,v} \cdot \bar{n}_{u,v} dl,$$

were the subscripts point out that we look at the integral with respect to the parametrization of $\mathbf{x}$.

Since we assume that our surface is piecewise linear, its gradient is constant over each triangle of the mesh. Due to this fact the integral of the normal vector of the border $\partial A_{\text{Mixed}}$ results in the same expression as the integral of the normal vector within a triangle. Thus, inside a triangle $T = (x_i, x_j, x_k)$, we can write

$$\int\limits_{\partial A_{\text{Mixed}} \cap T} \nabla_{u,v} \cdot \bar{n}_{u,v} dl = \nabla_{u,v} \mathbf{x} \cdot [a - b]^{\perp}_{u,v} = \frac{1}{2} \nabla_{u,v} \mathbf{x} \cdot [x_j - x_k]^{\perp}_{u,v}, \tag{3.22}$$

were a is the midpoint between x_i and x_j, and b is the midpoint between x_i and x_k (Figure 3.5). Furthermore, $\perp$ denotes the counter-clockwise rotation of 90 degrees.

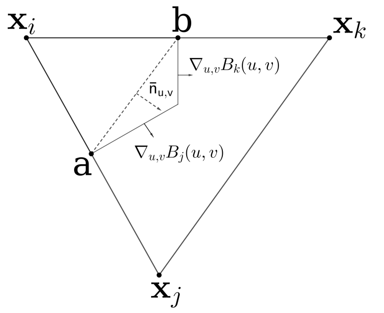


Figure 3.5: Sketch of a triangle T in a mesh. x_i, x_j, x_k are the vertices of T. a, b are the midpoints between the vertices x_i, x_j and x_i, x_k. $\bar{n}_{u,v}$ is the normal vector of T. $\nabla_{u,v} B_j(u,v), \nabla_{u,v} B_k(u,v)$ are the gradients of the two basis functions.

Since the function $\mathbf{x}$ is linear over any triangle T, we can use the linear basis functions B_l over the triangle to express $\mathbf{x}$ as

$$\mathbf{x} = \mathbf{x}_i B_i(u,v) + \mathbf{x}_j B_j(u,v) + \mathbf{x}_k B_k(u,v),$$

and its gradient as

$$\nabla_{u,v}\mathbf{x} = \mathbf{x}_i \nabla_{u,v} B_i(u,v) + \mathbf{x}_j \nabla_{u,v} B_j(u,v) + \mathbf{x}_k \nabla_{u,v} B_k(u,v). \tag{3.23}$$

Using the fact that the gradients of the three basis functions of any triangle T add up to zero

$$0 = \nabla_{u,v} B_i(u,v) + \nabla_{u,v} B_j(u,v) + \nabla_{u,v} B_k(u,v),$$

we can rearrange the terms and express $\nabla_{u,v} B_i(u,v)$ as

$$\nabla_{u,v} B_i(u,v) = -(\nabla_{u,v} B_j(u,v) + \nabla_{u,v} B_k(u,v)).$$

Substituting the above equation into equation (3.23) we get

$$\nabla_{u,v}\mathbf{x} = (\mathbf{x}_j - \mathbf{x}_i)(\nabla_{u,v} B_j(u,v)) + (\mathbf{x}_k - \mathbf{x}_i)(\nabla_{u,v} B_k(u,v)).$$

To find a more suitable expression for $\nabla_{u,v} B_j(u,v)$, we use the fact that the gradient of the j-th basis function can be expressed as $\frac{1}{2A_T}([\mathbf{x}_i - \mathbf{x}_k]^{\perp}_{u,v})^T$ (Figure 3.5), were the leading term is the normalization of the vector and A_T denotes the area of the triangle. This leads to

$$\nabla_{u,v}\mathbf{x} = \frac{1}{2A_T}\big((\mathbf{x}_j - \mathbf{x}_i)([\mathbf{x}_i - \mathbf{x}_k]^{\perp}_{u,v})^T + (\mathbf{x}_k - \mathbf{x}_i)([\mathbf{x}_j - \mathbf{x}_i]^{\perp}_{u,v})^T\big). \tag{3.24}$$

When we now combine equations (3.22) and (3.24) we get

$$\int\limits_{\partial A_{\text{Mixed}} \cap T} \nabla_{u,v} \cdot \bar{n}_{u,v} dl = \frac{1}{4A_T}\big(([\mathbf{x}_i - \mathbf{x}_k] \cdot [\mathbf{x}_j - \mathbf{x}_k])_{u,v}(\mathbf{x}_j - \mathbf{x}_i) + \\ ([\mathbf{x}_j - \mathbf{x}_i)] \cdot [\mathbf{x}_j - \mathbf{x}_k])_{u,v}(\mathbf{x}_k - \mathbf{x}_i)\big).$$

Moreover, the area A_T is proportional to the sine of any angle of the triangle. Therefore, we can use the cotangent of the two angles opposite to $\mathbf{x}_i$ to simplify the parameter space coefficients and write

$$\int\limits_{\partial A_{\text{Mixed}} \cap T} \nabla_{u,v} \cdot \bar{n}_{u,v} dl = \frac{1}{2}\big(\cot_{u,v}(\mathbf{x}_k)(\mathbf{x}_j - \mathbf{x}_i) + \cot_{u,v}(\mathbf{x}_j)(\mathbf{x}_k - \mathbf{x}_i)\big). \tag{3.25}$$

If we now look at equation (3.21) and use equation (3.25) on every triangle in the 1-ring neighbourhood of p_i we get

$$\iint\limits_{A_{\text{mixed}}} H(p_i)dA = \frac{1}{2} \sum_{j \in N_1(p_i)} (\cot \alpha_{ij} + \cot \beta_{ij})(\mathbf{x}_i - \mathbf{x}_j),$$

were $\alpha_{i,j}$ and β_{ij} are the two angles opposite to the edge in the two triangles sharing the edge (x_i, x_j) (Figure 3.4(c)).

■

To summarize the findings in this section, we first have to calculate the norm of the mean curvature normal vector over the 1-ring neighbourhood of the vertex x_i with the help of theorem 3.2.17. Furthermore, we have to divide the norm of the operator by the discrete mixed area around the vertex to get the correct spatial average. Finally, we have to divide this value by two (see definition 3.2.16).

Now we can calculate the **discrete Mean Curvature Operator** of a vertex x_i as

$$H_{x_i} = \frac{\| \sum_{j \in N_1(x_i)} (\cot \alpha_{ij} + \cot \beta_{ij})(\mathbf{x}_i - \mathbf{x}_j) \|}{4 A_{\text{mixed}}}. \tag{3.26}$$

3.2.2.2 Discrete Gaussian Curvature

To calculate the Gaussian curvature of a given vertex we use the Gauss-Bonnet theorem [5]. However, we first have to consider the relation between the Gaussian curvature at a point p_i and the infinitesimal surface patch around p_i. The Gaussian curvature can be expressed as the limit [9]

$$K = \lim_{\text{diam}(A) \to 0} \frac{A^G}{A},$$

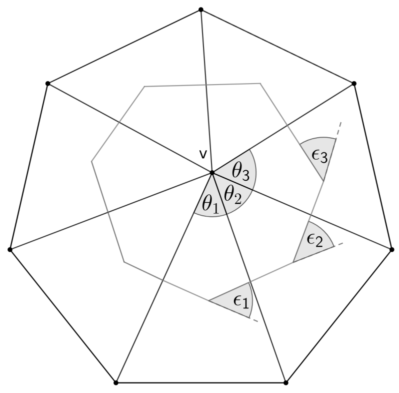
were A is again the infinitesimal surface patch around p_i, and A^G is the area of the image of the Gauss map associated with it. Since the Gauss map maps onto the unit sphere we calculate part of the surface area of S^2. Since, the geodesic curvature of a sphere is zero, we can use a simplified version of the Gauss-Bonnet theorem

$$\iint\limits_{A_{\text{mixed}}} K\, dA + \sum_{i=0}^{k} \varepsilon_i = 2\pi,$$

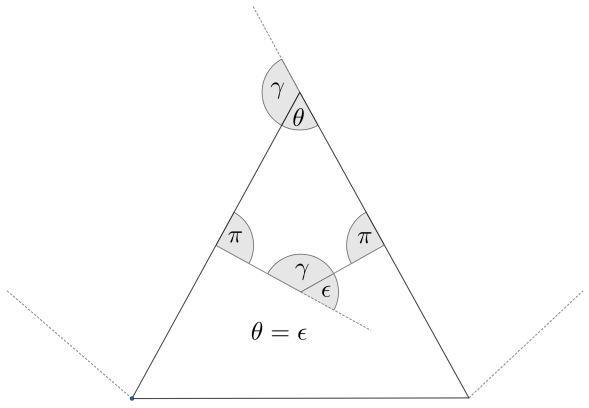
were ε denotes the external angles of the boundary (Figure 3.6(a)). For Voronoi regions it is easy to see that $\theta_i = \varepsilon_i$, because both edges of the Voronoi cell are perpendicular to the edges of the triangle and, therefore, $\theta_i + \alpha_i = \pi$ (Figure 3.6(b)). This equality also holds for non-Voronoi regions [25].

To conclude our findings we can calculate the **discrete Gaussian Curvature Operator** of a vertex x_i as

$$K_{x_i} = \frac{\left(2\pi - \sum_{j=1}^{f} \theta_j\right)}{A_{\text{mixed}}}. \tag{3.27}$$



(a) External angles



(b) The angle Θ is equal to the angle ε.

Figure 3.6: Angles for Gauss-Bonnet

3.2.2.3 Discrete Principal Curvatures

Since a mesh is a discrete representation of a surface it is possible, although very rarely, that $H_{p_i}^2 < K_{p_i}$, which would result in imaginary values for the principal curvatures. To avoid numerical errors we just set $H_{p_i}^2 - K_{p_i}$ to zero if it is negative. Then we can use equation (3.12) to calculate the principle curvatures.

Chapter 4

Software Tools

This chapter gives an overview over the software frameworks used in this thesis. These were used to develop the simplification algorithm and to calculate the metrics used to determine the quality of the simplification.

4.1 ViennaMesh

ViennaMesh is a C++ based meshing library that provides the user with a multitude of tools for generic high quality-mesh generation and mesh adaptation [29] [37]. This library allows the seamless exchange of meshing tools and mesh generation kernels.

The simplification algorithm presented in this work was created to add a flexible and fast way to detect the complexity of geometries in parts of a mesh and give ViennaMesh the ability to simplify parts of a mesh.

4.2 The Computational Geometry Algorithms Library

The Computational Geometry Algorithms Library project (CGAL) [36] is an open source software project that provides access to a number of geometric algorithms in the form of a C++ library. It provides the framework that is used to develop and test the simplification algorithm in this thesis.

CGAL provides a flexible implementation of the edge-collapse algorithm described in Section 2.2.1 and a method to calculate the curvature of a vertex, see Section 3.2.1. Additionally, CGAL stores a mesh in a *half-edge data structure*, which is a flexible high performance data structure.

The following packages from CGAL were used:

- *3D Polyhedral Surface* to convert the example meshes that were generated through simulation into a format that CGAL can interact with [15];
- *Estimation of Local Differential Properties of Point-Sampled Surfaces* to calculate the curvatures of meshes [26];
- *Triangulated Surface Mesh Simplification* to coarse the meshes [4].

4.2.1 Half-Edge Data Structure

In the previous section it was mentioned that CGAL uses a half-edge data structure to store meshes. A half-edge data structure is a special case of a directed graph:

4.2.1 Definition
An ordered pair $G = (V, A)$ is called a **directed graph** if

1. V is a set whose points are called **vertices**,
2. A is a set of ordered pairs of vertices which are called **directed edges**.

The naming similarity in the above definition with the naming conventions in Chapter 2 are intentional and are consistent. For the purposes of this thesis a vertex, either a 0-simplex of a mesh or a point in a graph, describes the same object.

An edge (1-simplex) is made up of two half-edges pointing in the opposite direction of each other. If an half-edge points from a vertex v_1 to another vertex v_2 its opposite half-edge points from v_2 to v_1. Each half-edge has one incident facet (2-simplex). Border edges have no incident facet, making it easy to detect the border of the mesh. A **half-edge data structure** is a directed graph that satisfies the two conditions above. An example of a half-edge is depicted in Figure 4.1.

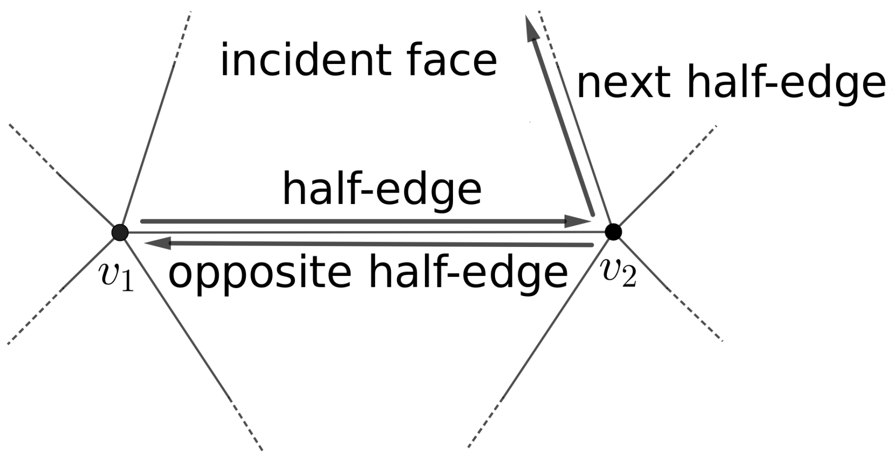


Figure 4.1: Example of a half-edge data structure describing the connection between two vertices v_1 and v_2.

4.3 Visualization Toolkit

The Visualization Toolkit (VTK) is an open source software system for 3D computer graphics, image processing, volume rendering, and scientific visualization [38].

VTK is used in two aspects in this work. First it is used to create the visualizations of the meshes. And second it provides an implementation to calculate the triangle quality metrics as described in Section 2.1.2 and an implementation to calculate the distance between two meshes as described in Section 2.2.3. These implementations are provided to the user in from of so called *filters*.

The following filters were used in this thesis:

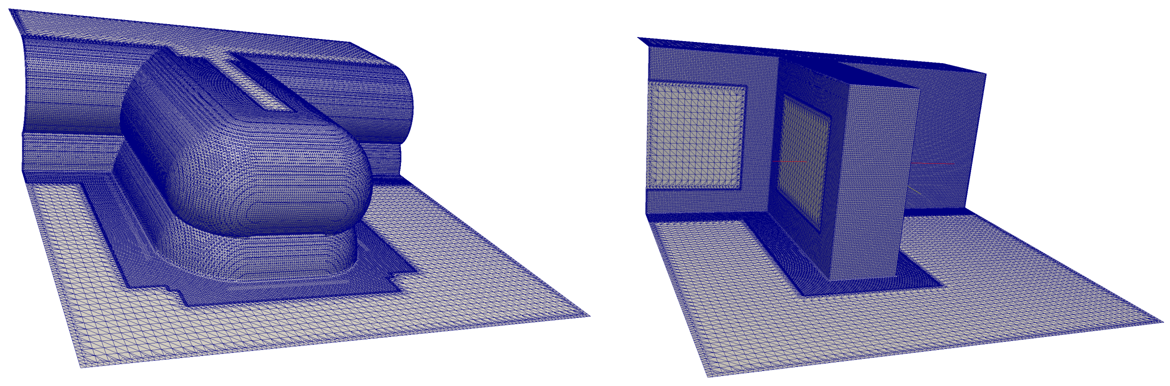
- vtkMeshQuality,
- vtkDistancePolyDataFilter.

Chapter 5

Simplification of Subdivided Surfaces

This chapter describes the developed method for simplifying subdivided surfaces. The basic idea is to detect geometric features using the curvature of the vertices in the mesh and divide the mesh, using this metric, into regions which can be simplified with different strategies and parameters.

The meshes shown in Figure 5.1(a) and Figure 5.1(b) are two examples originating from process TCAD workflows which are used to evaluate the developed techniques. The goal is to coarse these meshes with the following constraints: The geometric features of the mesh should be preserved, the elements have to be of high quality, the computation time should be in the same order of magnitude as the approach of Lindstrom and Turk [17], and the distance to the original mesh, as described in Section 2.2.3, should be small.



(a) Mesh *Smooth Bay* (b) Mesh *Square Bay*

Figure 5.1: The two reference meshes originating from process TCAD used within this thesis, these meshes were created by a marching cubes algorithm.

The two meshes have the number of elements and vertices shown in Table 5.1. When these metrics are compared to the geometries shown in Figure 5.1 it is intuitively clear that these geometries can be represented with a lesser amount of elements. Both meshes have locally flat and curved areas which can be represented with a differing amount of elements to preserve the original geometry. The first two sections of this chapter formalize this intuition.

	Smooth Bay	*Square Bay*
elements	140,698	132,016
vertices	70,831	66,468

Table 5.1: Number of vertices and elements for both test meshes.

All investigations were performed on an Intel®Core$^{\text{TM}}$i5-5200U CPU with 2.20GHz and 8 GB of RAM. The compilation of the code was conducted using gcc version 5.4.1 and optimization level -O3.

5.1 Curvature Comparison

The simplification method presented in this chapter uses the curvature of vertices in the mesh to detect the complexity of the local geometry. Hence, the first challenge is to calculate the curvature of each vertex in the mesh. In Section 3.2 two different methods of calculating the curvature of a mesh were introduced. The *jet-fitting* method is implemented in CGAL [26]. The *spatial averaging* method was implemented during this work. This section is dedicated to the comparison and evaluation of these two approaches.

Both test meshes have a small amount of edges, i.e., 12 for *Smooth Bay* and four for *Square Bay*, with a length being 20 orders of magnitude shorter than their neighbours. These edges were treated as numerical artefacts and, thus, have been removed in a pre-processing step.

The *jet-fitting* approach only uses vertex data and calculates an approximation of the Taylor expansion of the given surface patch. During this process a minimization problem is solved which has a computational complexity of $\mathcal{O}(N_n m)$ [14], were N_n is defined as in Section 3.2.1.1. For the investigations presented in this section n is set to 2, and $m \in \mathbb{N}$ is the number of points, or in the case of a mesh the number of vertices, used for the calculation.

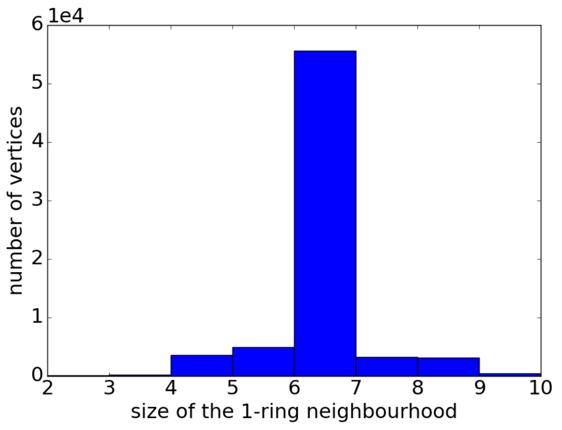
The *spatial averaging* approach calculates the value of two operators in a vertex and relates this value to a surface patch around this vertex. This approach uses the connectivity provided by the edges in a mesh. Thus, it only requires the calculation of angles, edge length, and areas, which has a computational complexity of $\mathcal{O}(m)$, were $m \in \mathbb{N}$ is the number of vertices in the 1-ring neighbourhood of the vertex.

Remark: The curvature of a surface is a real number and thus has no unit, but, the curvature changes with respect to the scale of the surface. Hence, to define a unit for the curvature the size of the object has to be taken into consideration. However, the chosen unit has no effect on the calculations in this thesis and would only result in additional computation costs. For this reason the curvatures are shown as real numbers and the scales in all Figures are chosen such that they highlight the important differences.

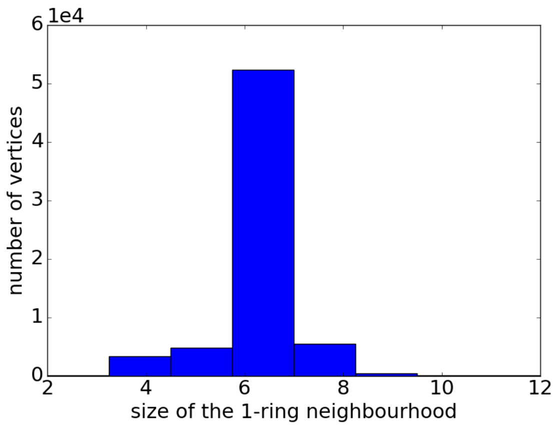
5.1.1 Evaluation Setup

To asses the curvature calculation two metrics were used in the following empirical studies. The first metric is the calculation time of the two algorithms and the second is the quality of the curvature calculation. The evaluated metrics are averaged using 100 executions.

Since spatial averaging always uses a fixed amount of vertices the 1-ring neighbourhoods of both test meshes were analyzed. This was done to find a starting point, were both approaches are provided with the same information, enabling a proper evaluation. The number of vertices in the 1-ring neighbourhoods of both test meshes is shown in Figure 5.2(a) and Figure 5.2(b).



(a) *Smooth Bay*



(b) *Square Bay*

Figure 5.2: Number of vertices in the 1-ring neighbourhood of both test meshes.

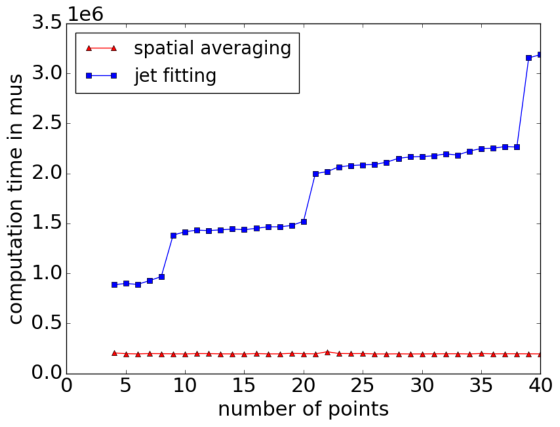
The average size of the 1-ring neighbourhoods in both meshes is approximately 6 (see Figure 5.2). This gives an average computation time of the curvature for each vertex of $\mathcal{O}(m^2)$ and $\mathcal{O}(m)$, respectively, were m is the size of the average 1-ring neighbourhood.

When provided with the same mesh and the same number of vertices both algorithms always produce the same results for the curvatures of the vertices in the mesh. However, when *jet-fitting* is used there is the option of providing additional point data which will increase the accuracy of the calculation at the cost of additional computation time.

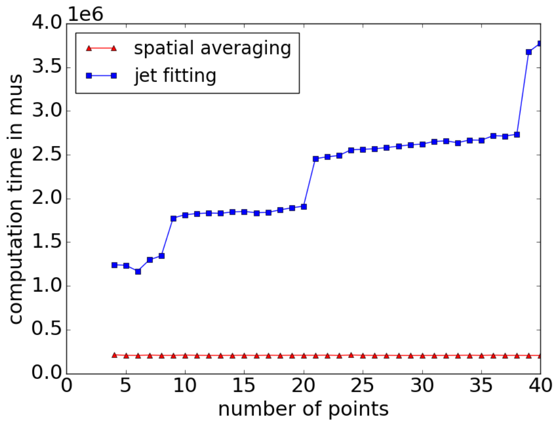
5.1.2 Comparison of Curvature Calculation Methods

First, the calculation time of both algorithms can be seen in Figure 5.3. It has to be noted, that the number of points that is provided to the approach implemented in CGAL is a lower bound, meaning that if there are more than the specified number of points in the 1-ring neighbourhood of a vertex all vertices are considered in the calculation of the curvature of this vertex. This explains the sudden jumps in the calculation times at 6, 18, 35, and 40 vertices seen in Figure 5.3.

It can be seen that the calculation times of both approaches follow the expected patterns due to their differing computational complexities. Since *jet-fitting* has a quadratic computational complexity and is able to utilize additional vertex data, the computation time is higher than with *spatial averaging*, which has a linear computational complexity. Additionally, it can be seen that *jet-fitting* takes more points into consideration when provided with 6, 18, 35, and 40 vertices. This has to be taken into account for a proper comparison of both methods.



(a) *Smooth Bay*



(b) *Square Bay*

Figure 5.3: Average curvature computation times.

Since the border of an open mesh is not continuous it has no curvature. However, both methods discussed yield values for such vertices. To guarantee that the border of the mesh has a consistent curvature, a constant Gaussian and mean curvature is assigned.

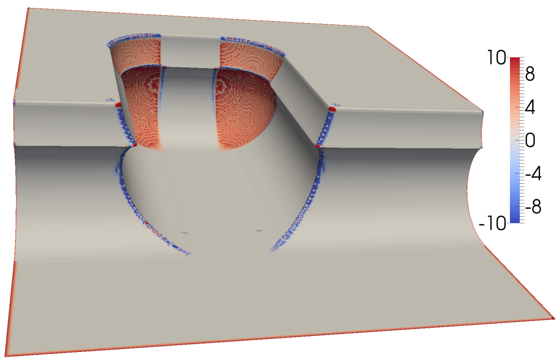
A mesh is a discrete representation of a surface, so there will be numerical errors in the calculation of the curvature. So the following definition is given:

5.1.1 Definition
The curvature calculation of a vertex v has an **error**, if the curvature value of v differs from the curvature value a continuous surface would have in this vertex.

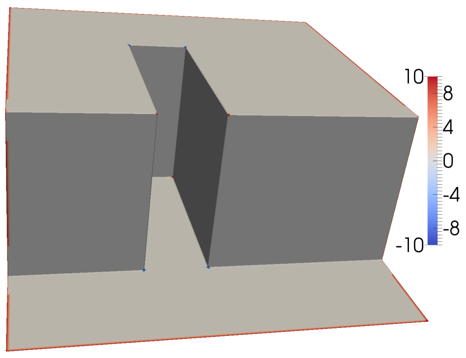
5.1.2.1 Gaussian Curvature

First, the Gaussian curvature of both approaches is compared. In Figure 5.4 the results from *spatial averaging* and in Figure 5.5 the results from *jet-fitting* as implemented in CGAL with an increasing number of points are shown.

When comparing the results of the Gaussian curvature calculation with 6 and 18 points two stark differences can be observed (see Figure 5.6 and Figure 5.7 as well as Figure 5.8 and Figure 5.9). The errors shown in Figure 5.6 and Figure 5.7 are a consequence of the position of the vertices in the 1-ring neighbourhood around the vertex v for which the curvature is calculated. These errors occur because, the function that is calculated during the *jet-fitting* process has to fit the provided point data and as a consequence of this, is acute in v. To counteract these errors additional points around the vertex have to be taken into consideration. However, this would give rise to the error shown in Figure 5.8 and Figure 5.9. The vertices along the ridge of the mesh are coplanar, meaning at least three vertices of the ridge lie on the same line, implying that one of the principal curvatures should be 0. As a consequence the Gaussian curvature should also be 0. However, the *jet-fitting* approach tries to find a C^2-function that fits the provided data. So, it introduces an error into the calculation and creates a curvature along the ridge.



(a) *Smooth Bay*



(b) *Square Bay*

Figure 5.4: Gaussian curvature calculated with *spatial averaging*. The colors indicate the rate of the curvature. *Spacial averaging* only takes the 1-ring neighbourhood of a vertex into consideration, thus, it is possible that ,locally, the curvatures of two adjacent vertices are curved in opposite directions.

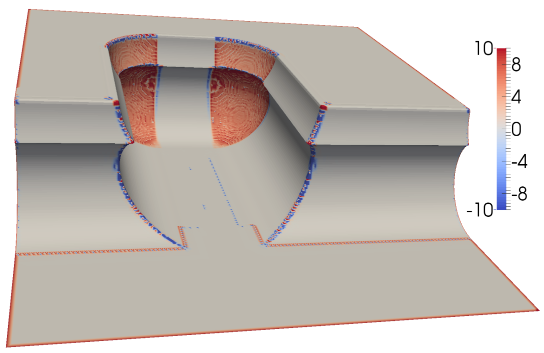
When considering the ridges of the meshes shown in Figures 5.5(c)-(h) a similar effect is visible. Since the vertices that are added to the calculation of the *jet-fitting* algorithm have to be chosen in a neutral way, vertices might lie in a locally flat part of the surface get a curvature value $\neq 0$. This happens because one vertex that was added to the calculation and is not part of the 1-ring neighbourhood of the vertex v does not lie in the same plane as all vertices in the 1-ring neighbourhood and v.

However, the Gaussian curvature in other parts of the meshes stays the same. To quantify this observation the difference in the calculated Gaussian curvature between jets with a different number of points is compared in Figure 5.10. The differences are plotted in the range of $[-100, 100]$, because there were approximately $1,000$ outliers with a very high difference in the calculated Gaussian curvature. These vertices are part of sharp ridges in the geometry of the mesh, so they are dismissed as numerical errors. Both histograms show that the vast majority of vertices have approximately the same Gaussian curvature and that on average only a few thousand points have a significant deviation in the difference of their Gaussian curvature. This shows that not much additional information about the local geometry of the mesh is gained when increasing the number of points provided to the *jet-fitting* algorithm.

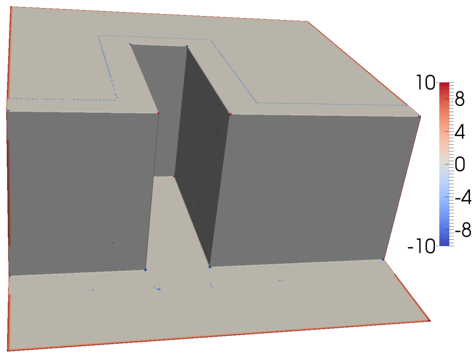
The results in Figure 5.10 suggest that 18 data points are a good trade-off between the two previously discussed errors. Furthermore, the errors shown in Figure 5.6 and Figure 5.7 are gone and the error discussed in the previous paragraph does not extend far into the flat planes of the mesh. This leads to the comparison between *spatial averaging* and *jet-fitting*. Figure 5.11 shows the differences between the values of the Gaussian curvature calculated with *spatial averaging* and *jet-fitting*. For this Figure the same specifications apply as for Figure 5.10.

The number of errors in the calculation of the Gaussian curvature increases when the number of points for *jet-fitting* is increased (see Figure 5.11(b)). This happens due to the error in the calculation of vertices that lie on ridges. As previously discussed the magnitude of this error is negligible when 18 points are used and thus can be ignored.

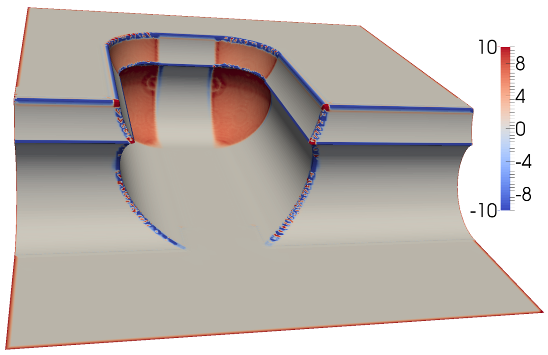
Of the two presented methods that calculate the Gaussian curvature of a vertex in a mesh, the preferred method is *spatial averaging*. This method has a far superior computation time and as has been shown, it computes superior results over *jet-fitting*.



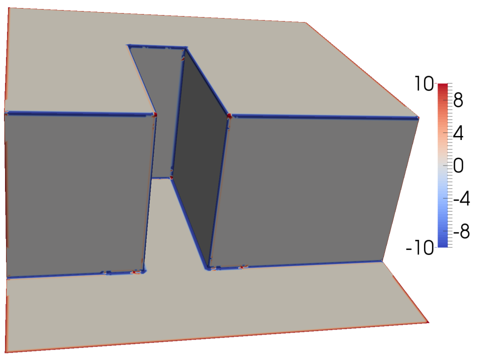
(a) *Smooth Bay* with a minimum of 6 vertices.



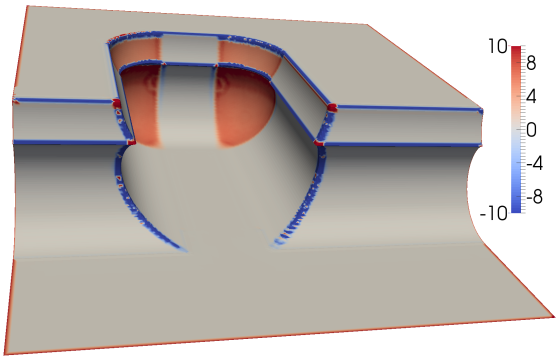
(b) *Square Bay* with a minimum of 6 vertices.



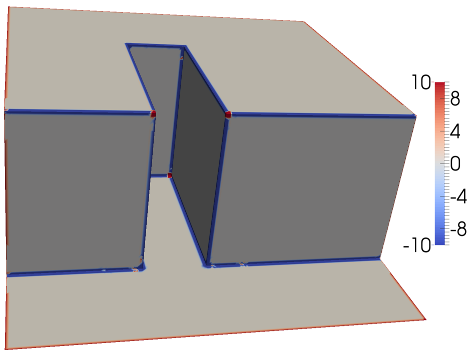
(c) *Smooth Bay* with a minimum of 18 vertices.



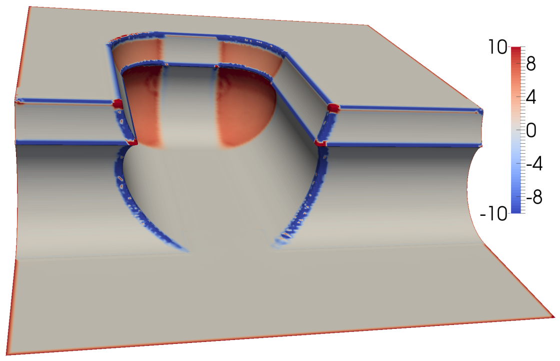
(d) *Square Bay* with a minimum of 18 vertices.



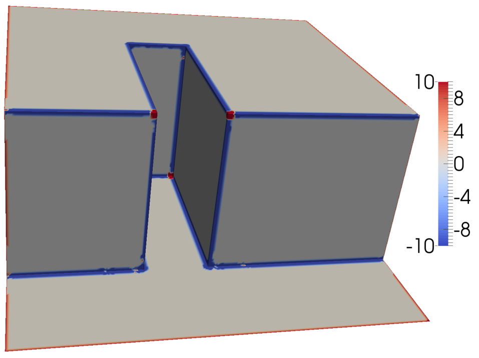
(e) *Smooth Bay* with a minimum of 35 vertices.



(f) *Square Bay* with a minimum of 35 vertices.

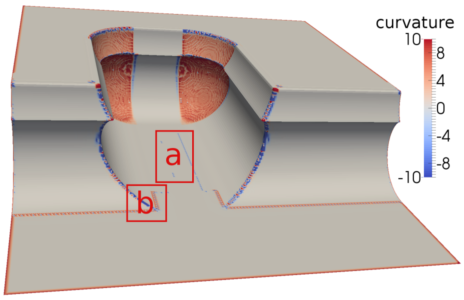


(g) *Smooth Bay* with a minimum of 40 vertices.

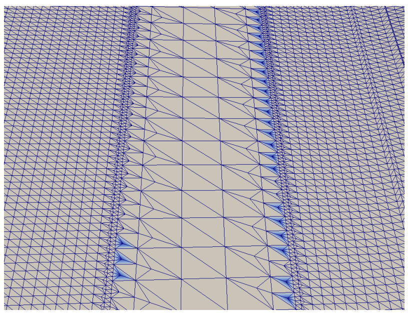


(h) *Square Bay* with a minimum of 40 vertices.

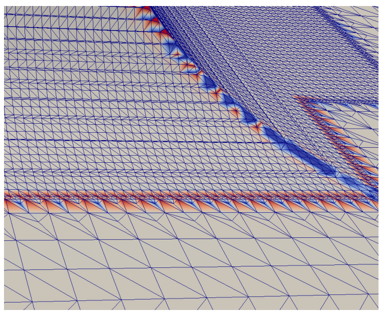
Figure 5.5: Gaussian curvature calculated with *jet-fitting*. The colors indicate the rate of the curvature. *Jet-fitting* only takes a certain number of vertices into consideration to approximate the Weingarten map ,thus , it is possible that ,locally, the curvatures of two adjacent vertices are curved in opposite directions.



The colors indicate the curvature of the vertices in the mesh *Smooth Bay*

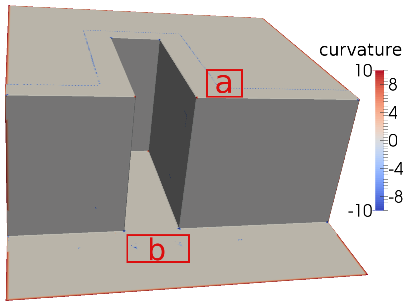


(a) Locally flat 1-ring neighbourhood of a vertex with a Gaussian curvature $\neq 0$.

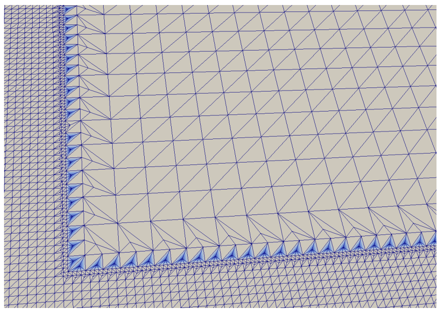


(b) Locally flat 1-ring neighbourhood of a vertex with a Gaussian curvature $\neq 0$.

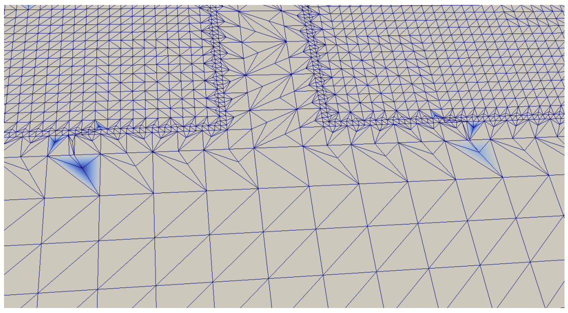
Figure 5.6: Error (see Definition 5.1.1) in Gaussian curvature calculation with 6 points *Smooth Bay*.



The colors indicate the curvature of the vertices in the mesh *Square Bay*

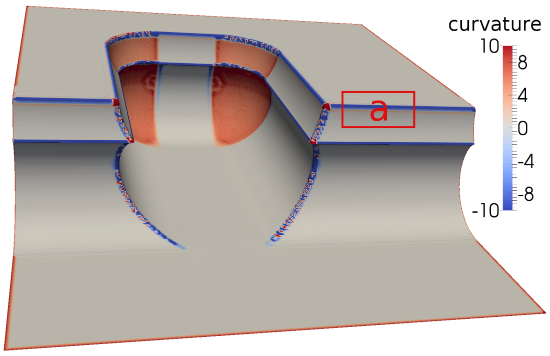


(a) Locally flat 1-ring neighbourhood of a vertex with a Gaussian curvature $\neq 0$.

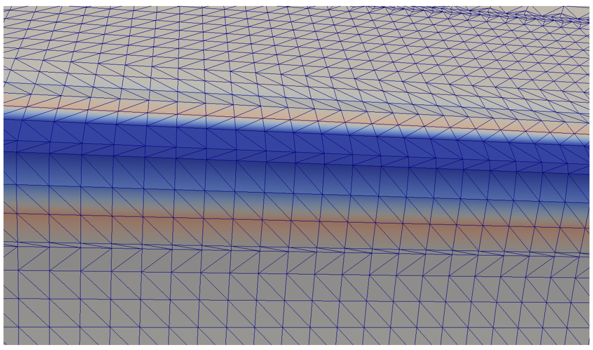


(b) Locally flat 1-ring neighbourhood of a vertex with a Gaussian curvature $\neq 0$.

Figure 5.7: Error (see Definition 5.1.1) in Gaussian curvature calculation with 6 points *Square Bay*.

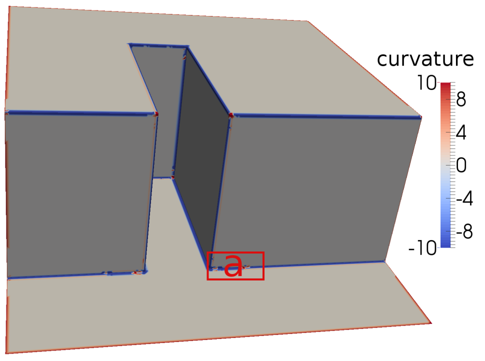


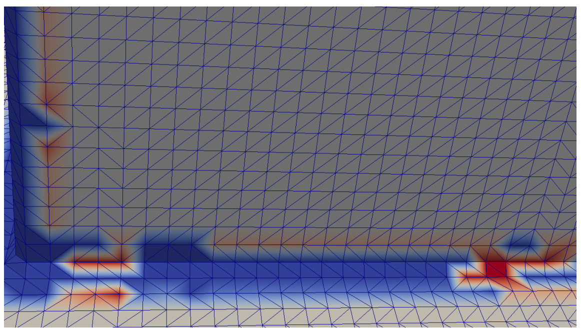
The colors indicate the curvature of the vertices in the mesh *Smooth Bay*



(a) Vertices that get a Gaussian curvature $\neq 0$ that should be 0.

Figure 5.8: Error (see Definition 5.1.1) in Gaussian curvature calculation with 18 points *Smooth Bay*.

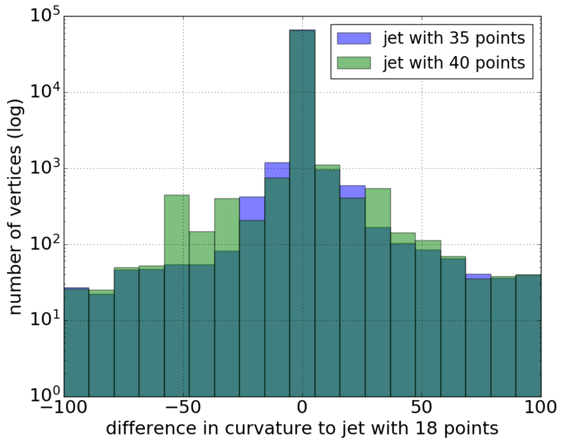




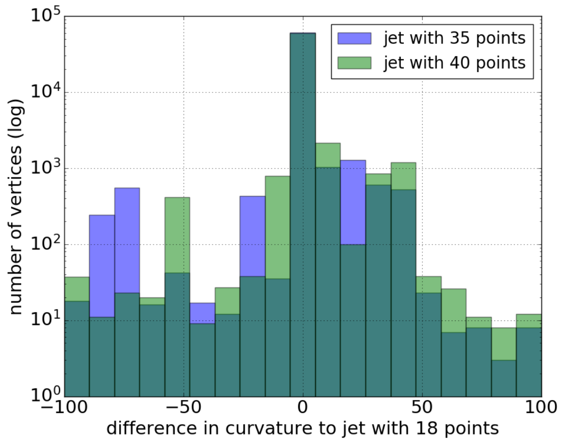
The colors indicate the curvature of the vertices in the mesh *Square Bay*

(a) Vertices that get a Gaussian curvature $\neq 0$ that should be 0.

Figure 5.9: Error (see Definition 5.1.1) in Gaussian curvature calculation with 18 points *Square Bay*.

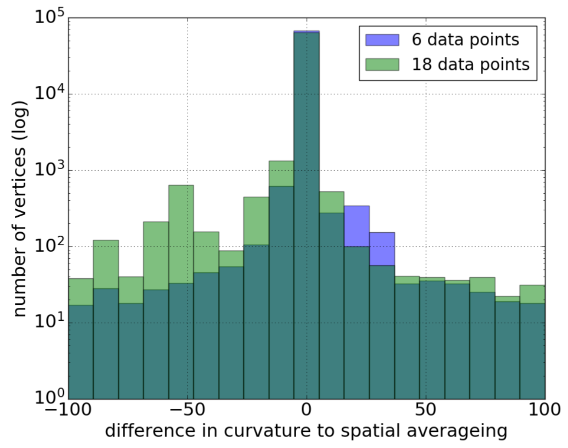


(a) *Smooth Bay*

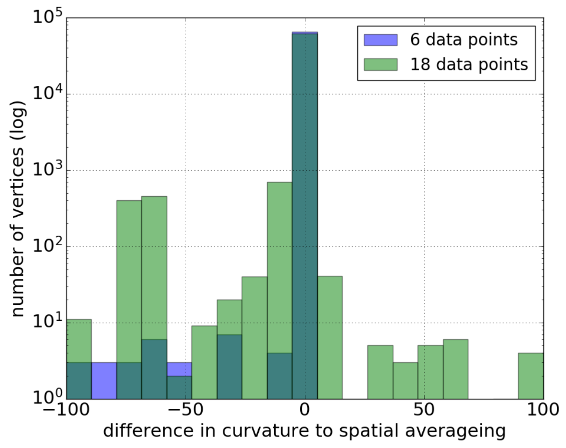


(b) *Square Bay*

Figure 5.10: Differences in the Gaussian curvature between jets with 18, 35, and 40 points.



(a) *Smooth Bay*



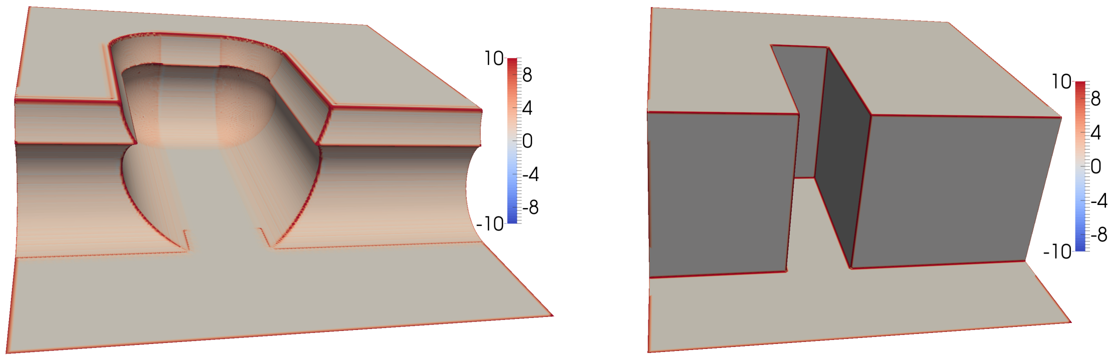
(b) *Square Bay*

Figure 5.11: Differences in the Gaussian curvature between *spatial averaging* and *jet-fitting*.

5.1.2.2 Mean Curvature

The evaluation and comparison of the results from the mean curvature calculation is treated in a similar way as the Gaussian curvature calculation. In Figure 5.4 the results from *spatial averaging* and in Figure 5.5 the results from *jet-fitting* are displayed with an increasing number of points.

It has to be noted that *spatial averaging* only calculates the absolute value of the mean curvature, so information about the direction of the curvature is not present. However, for the investigations in this work only the information about the magnitude of the curvature is needed since it gives information about the complexity of the geometry; the direction of the curvature only provides information about the type of geometry.



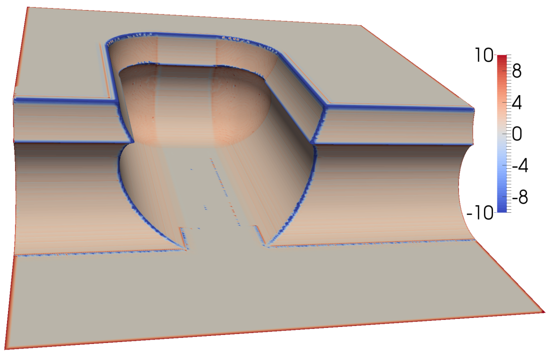
(a) *Smooth Bay* (b) *Square Bay*

Figure 5.12: Mean curvature calculated with spatial averaging. The colors indicate the rate of the curvature.

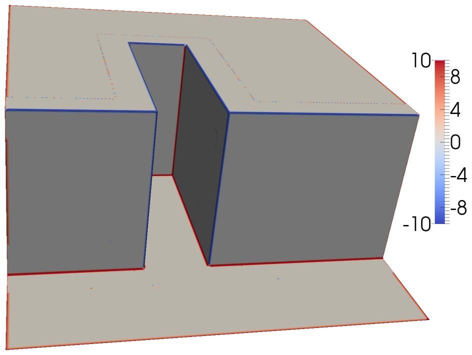
Both previously discussed errors, the error were vertices that are part of a flat part of the mesh get a curvature value greater zero when using 6 points (see Figure 5.6 and Figure 5.7), and the error were the vertices that are part of a locally flat part of the mesh but are near a ridge get a curvature greater zero (see Figure 5.8 and Figure 5.9), are also present in the calculation of the mean curvature. As argued in the previous section 18 data points provide a good basis for the comparison of the two approaches. However, the mean curvature calculated with *spatial averaging* has an error (see Figure 5.14) that is not present when calculating the curvature with *jet-fitting*. The vertex from which the curvature is calculated has one coordinate which value has an order of magnitude of 10^{-3} and the value of the same coordinate of the vertices in its 1-ring neighbourhood is 10^{-17}. Hence, *spatial averaging* calculates a curvature for this vertex. This error is due to the provided data and, thus, has to be handled as a numerical error.

To ensure that the values of the mean curvature are comparable between *spatial averaging* and *jet-fitting* the absolute value of the mean curvature is used to create the histograms shown in Figure 5.15 and Figure 5.16.

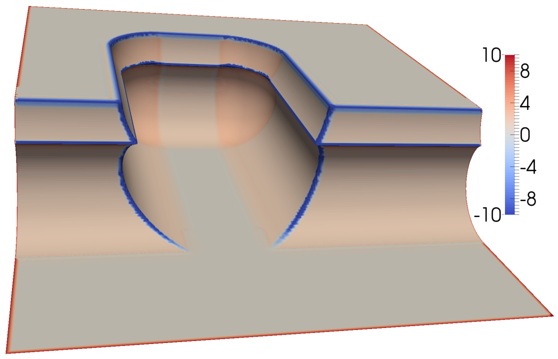
Unlike with the calculation of the Gaussian curvature there have not been any outliers in the calculation of the difference in mean curvature. This is due to the fact that on sharp ridges the product of the two principle curvatures grows faster than their sum. The results in Figure 5.15 suggest that a limited amount of accuracy is gained when the number of points provided to the jet-fitting approach is increased.



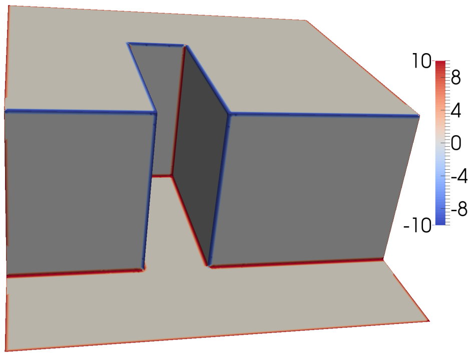
(a) *Smooth Bay* with a minimum of 6 vertices.



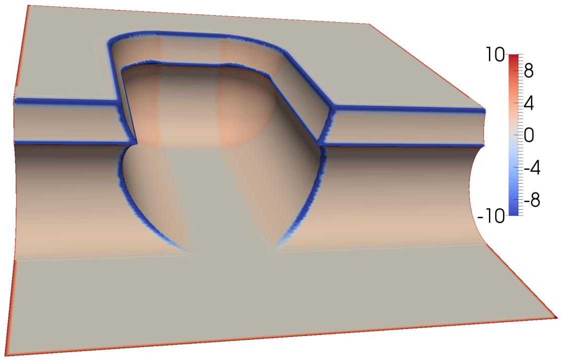
(b) *Square Bay* with a minimum of 6 vertices.



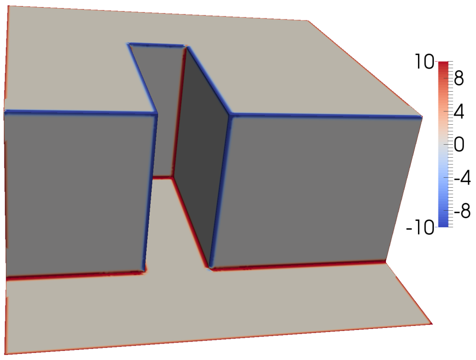
(c) *Smooth Bay* with a minimum of 18 vertices.



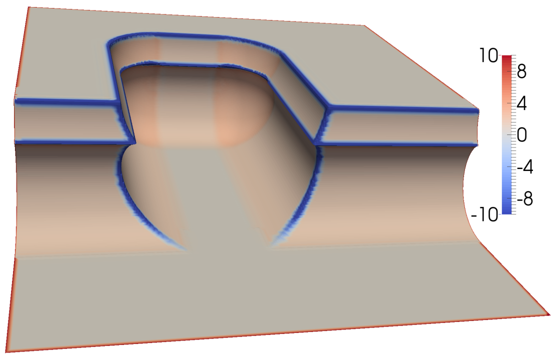
(d) *Square Bay* with a minimum of 18 vertices.



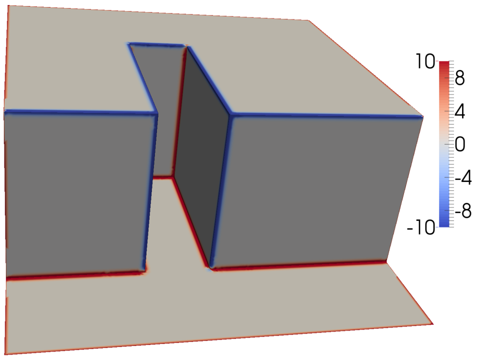
(e) *Smooth Bay* with a minimum of 35 vertices.



(f) *Square Bay* with a minimum of 35 vertices.

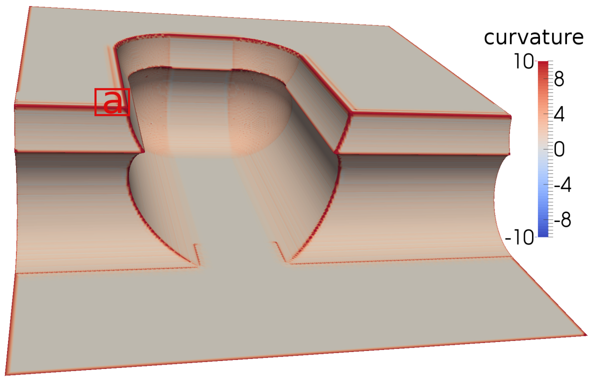


(g) *Smooth Bay* with a minimum of 40 vertices.

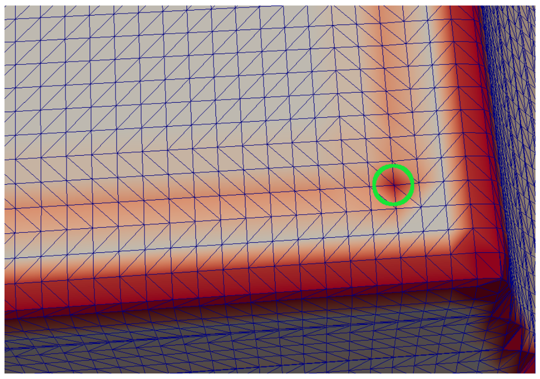


(h) *Square Bay* with a minimum of 40 vertices.

Figure 5.13: Mean curvature calculated with jet-fitting. The colors indicate the rate of the curvature.



The colors indicate the curvature of the vertices in the mesh *Smooth Bay*



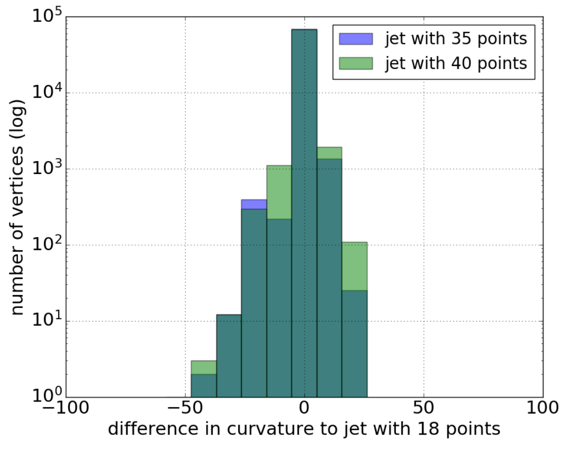
Zoom of highlighted region in left figure (a): Vertex (green circle) gets a curvature $\neq 0$ because of a small difference in in one coordinate.

Figure 5.14: Error (see Definition 5.1.1) in mean curvature calculation when using *spatial averaging Smooth bay.*

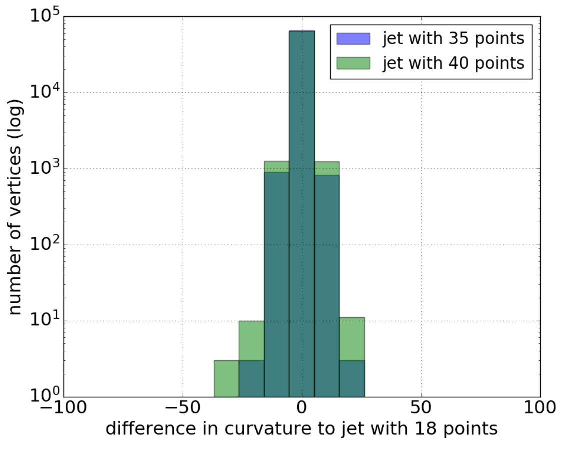
Figure 5.16 shows that only about $2,000$ vertices have a difference in the calculated value of the absolute mean curvature between spatial averaging and jet-fitting.

These investigations lead to the conclusion that if the direction of the mean curvature is needed, *jet-fitting* provides a more straightforward solution. However, when only the rate of the curvature is needed *spatial averaging* provides comparable accuracy with a faster calculation time.

Choosing a preferred method to calculate the mean curvature of a vertex is not as easy as with the Gaussian curvature. However, *spatial averaging* was chosen for several reasons. The quality of the curvature calculation of this approach is comparable to the quality of *jet-fitting* which is provided with at least 18 vertices. The calculation time of *spatial averaging* is at least one order of magnitude faster then *jet-fitting*. A drawback of *spatial averaging* is that it only calculates positive values for the mean curvatures. However, this is not relevant for this work, since the rate of the curvature is the important metric and its direction does not provide useful information. When *Jet-fitting* is provided with a larger number of points, it starts smoothing the surface and detects vertices which lie in a flat plane but are near a ridge as curved vertices, which is counterproductive.

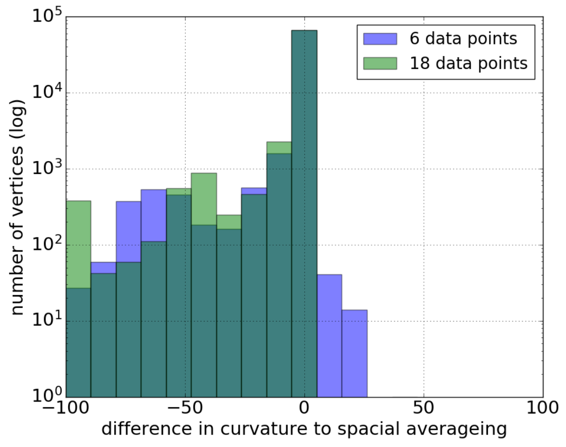


(a) *Smooth Bay*

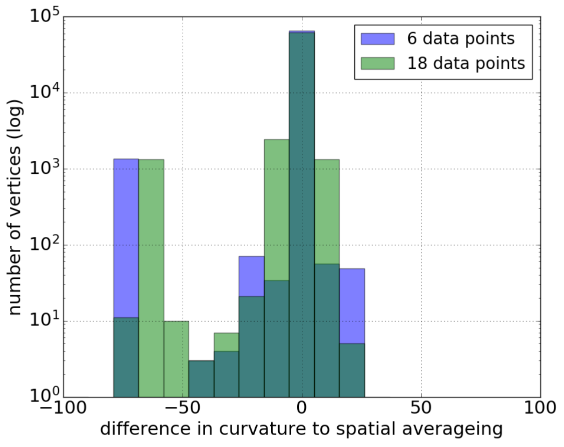


(b) *Square Bay*

Figure 5.15: Differences in the mean curvature between jets with 18, 35, and 40 points.



(a) *Smooth Bay*

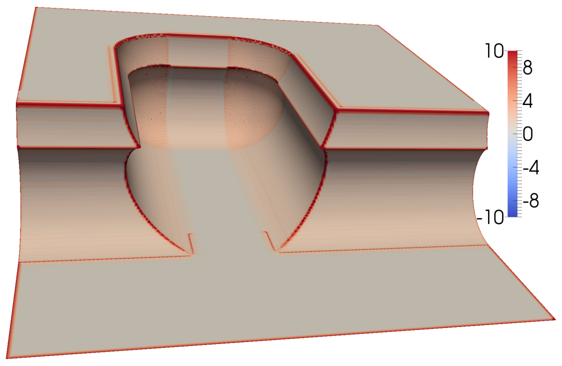


(b) *Square Bay*

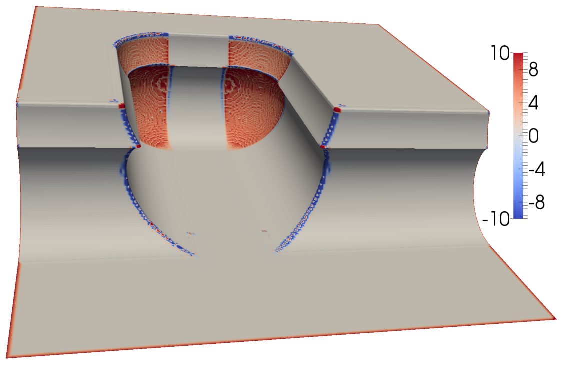
Figure 5.16: Differences in the mean curvature between *spatial averaging* and *jet-fitting*.

5.2 Feature Detection

In the previous section the quality and calculation time of the mean and Gaussian curvature calculation was discussed. Figure 5.17 and Figure 5.18 show the results of the mean and Gaussian curvature values for both test meshes calculated with *spatial averaging*. These suggest that the vertices of a mesh can be separated into different sets based on their curvature values. This section argues how these metrics can be used to detect parts of a mesh with different geometries.

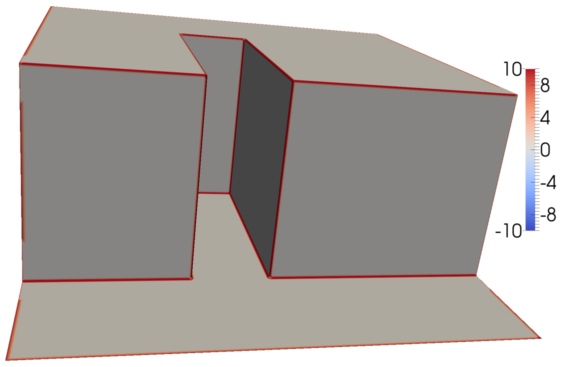


(a) Mean curvature

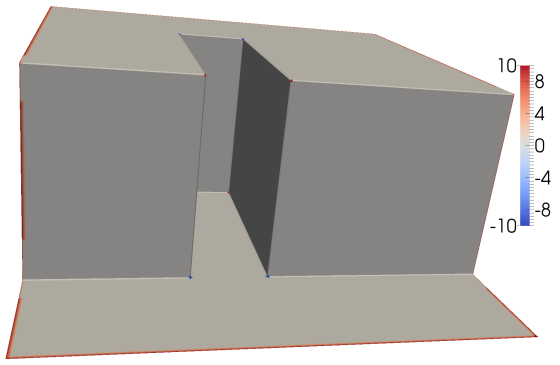


(b) Gauss curvature

Figure 5.17: Curvatures of the mesh *Smooth Bay*. The colors indicate the curvature of the vertices calculated with spacial averaging.



(a) Mean curvature



(b) Gauss curvature

Figure 5.18: Curvatures of the mesh *Square Bay*. The colors indicate the curvature of the vertices calculated with spacial averaging.

Let $\tau \in \mathbb{R}$ be a user defined number. The value of τ depends on the underlying mesh. In the second part of this section it is discussed how τ is determined for *Smooth Bay* and *Square Bay*. From now on vertices with at least one absolute principal curvature larger than τ are called **feature vertices** and vertices were both absolute principal curvatures are smaller than τ are called **flat vertices**. This convention allows the vertices in a mesh to be assigned to two disjunct sets: The set of flat vertices $\mathcal{M}_{\text{flat}}$, and the set of feature vertices $\mathcal{M}_{\text{feature}}$.

Algorithm 2 shows how the vertices in a mesh can be assigned to one of the previously defined sets.

```
input  : mesh $\mathcal{M}$, tolerance $\in \mathbb{R}$
output: $\mathcal{M}_{\text{flat}}$, $\mathcal{M}_{\text{feature}}$
set $\mathcal{M}_{\text{flat}} \leftarrow \emptyset$
set $\mathcal{M}_{\text{feature}} \leftarrow \emptyset$
foreach vertex $v$ in $\mathcal{M}$ do
    // $\kappa_1(v)$ and $\kappa_2(v)$ denote the 2 principle curvatures of $v$
    if ($v$ is NOT part of the border of $\mathcal{M}$) then
        if $|\kappa_1(v)| \geq tolerance \vee |\kappa_2(v)| \geq tolerance$ then
            $\mathcal{M}_{\text{feature}} \leftarrow \mathcal{M}_{\text{feature}} \cup v$;
        else
            $\mathcal{M}_{\text{flat}} \leftarrow \mathcal{M}_{\text{flat}} \cup v$;
        end
    end
end
```

Algorithm 2: Partition of vertices into $\mathcal{M}_{\text{flat}}$ and $\mathcal{M}_{\text{feature}}$.

In Algorithm 2 border vertices are not assigned to one of the sets $\mathcal{M}_{\text{flat}}$ or $\mathcal{M}_{\text{feature}}$. During the simplification process the border vertices have to be taken into consideration to ensure that the shape of the mesh stays in tact, so a method is needed to assign these vertices to one of the two previously defined sets. However, as previously discussed, vertices that are part of the border of a mesh have no curvature so the following method is used: The 1-ring neighbourhood of each border vertex is analysed and border vertices whose 1-ring neighbourhood only consists of vertices that are in $\mathcal{M}_{\text{flat}}$ are assigned to this set. In all other cases the border vertex is assigned to $\mathcal{M}_{\text{feature}}$ (see Algorithm 3).

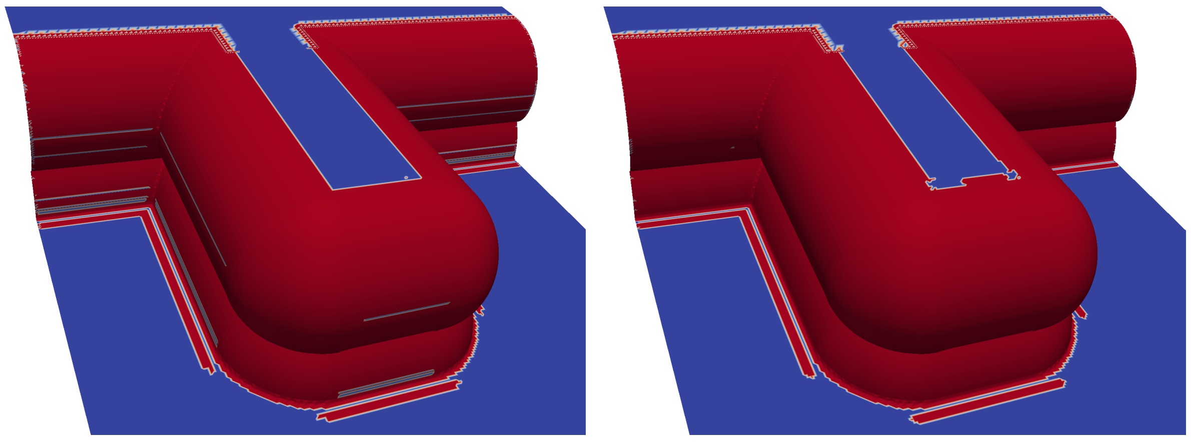
```
input  : mesh $\mathcal{M}$, $\mathcal{M}_{\text{flat}}$, $\mathcal{M}_{\text{feature}}$
output: $\mathcal{M}_{\text{flat}}$, $\mathcal{M}_{\text{feature}}$
foreach border vertex $v$ in $\mathcal{M}$ do
    foreach vertex $v_i$ in $N_1(v)$ do
        if $v_i$ is a feature $\wedge$ $v_i$ not a boarder vertex then
            $\mathcal{M}_{\text{feature}} \leftarrow \mathcal{M}_{\text{feature}} \cup v$;
            GOTO next vertex;
        end
    end
    $\mathcal{M}_{\text{flat}} \leftarrow \mathcal{M}_{\text{flat}} \cup v$;
    next vertex:
end
```

Algorithm 3: Adding border vertices to the partition.

Algorithm 3 requires a partitioning of all inner vertices of the mesh, thus, it has to be computed after Algorithm 2. As can be seen in Figure 5.18 the mesh *Square Bay* has huge areas which have zero curvature and it has only a few sharp ridges. Due to this fact the tolerance value does not impact the assignment of the vertices as much as the assignments of the vertices of the mesh *Smooth Bay* so only the latter mesh is used for the further discussion in this section.

The partition of the vertices with four different tolerances is shown in Figure 5.19.

When comparing Figure 5.19(a) and Figure 5.19(d) the impact of the chosen tolerance on the quality of the partitioning of the vertices is observable. However, neither produces a satisfying partition, i.e., with no errors. Figure 5.20 shows all vertices with a ratio between the flat vertices and all vertices in the 1-ring neighbourhood larger than zero: All vertices which are on the border between the sets $\mathcal{M}_{\text{flat}}$ and $\mathcal{M}_{\text{feature}}$ of the mesh. A genuine border vertex between these two sets should have a ratio between 0.4 and 0.6, because approximately half of the vertices in the 1-ring neighbourhood should lie in $\mathcal{M}_{\text{flat}}$ and $\mathcal{M}_{\text{feature}}$. If this is not the case the border between the regions is rigid, resulting in bad triangles during the simplification process. It is obvious that there have to be border vertices with a lower or higher ratio in an arbitrary mesh, but Figure 5.20 shows that there is a significant number of such vertices.


(a) $\tau = 0.5$ (b) $\tau = 0.1$

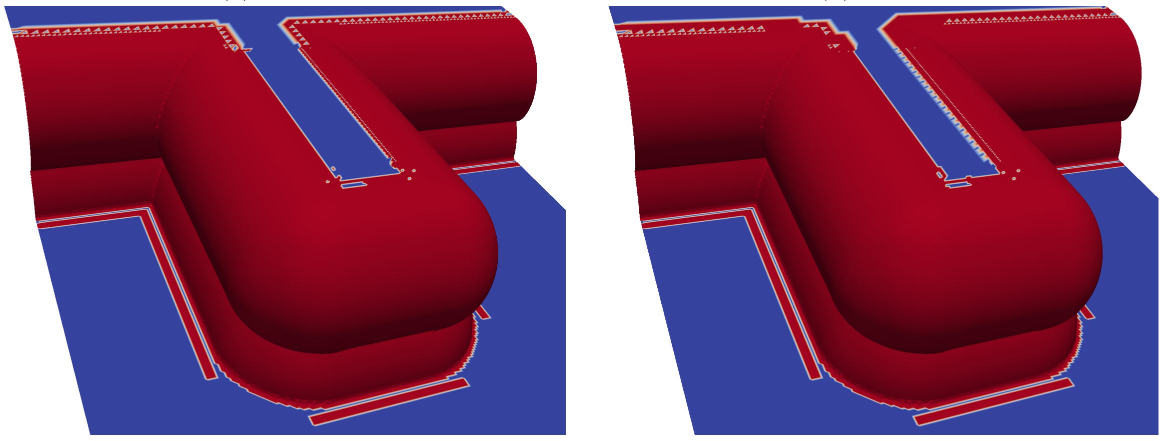

(c) $\tau = 0.001$ (d) $\tau = 0.0001$

Figure 5.19: Vertex partitioning of the mesh *Smooth Bay* with different tolerances $\tau \in \mathbb{R}$. If both principal curvatures of a given vertex are less than τ this vertex is put into $\mathcal{M}_{flat}$ (blue areas) otherwise it is put into $\mathcal{M}_{feature}$ (red areas).

Figure 5.21 shows the errors encountered during the feature detection process.

It can be seen in Figure 5.22(a) that if the tolerance is too high, as in the case of *Smooth Bay* $\tau = 0.5$, big areas of a curved part of the mesh are detected as flat planes. This is the result of triangles with at least one very short edge.

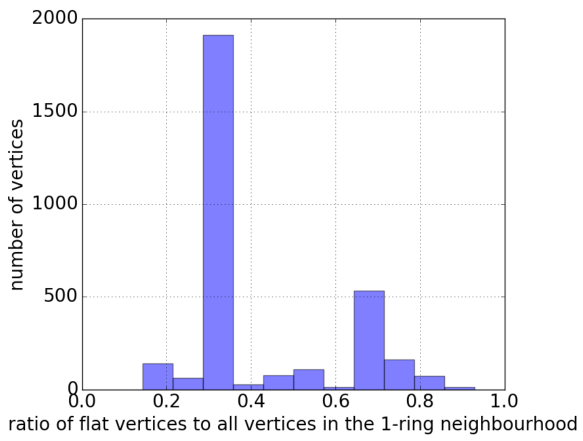


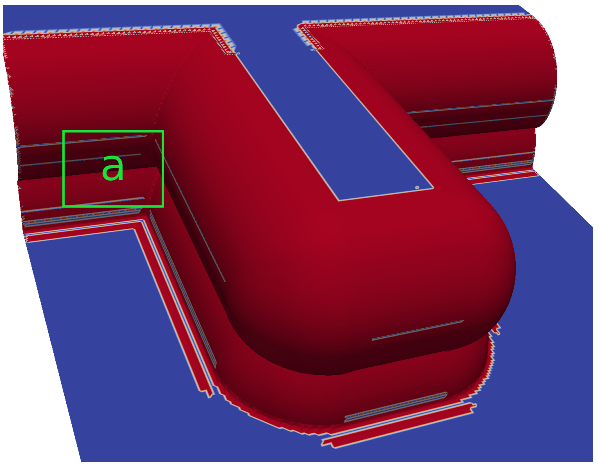
Figure 5.20: Quality of the partition with $\tau = 0.1$.

One way to deal with these errors is to reduce the tolerance. The second way is to use the information gained from the calculation of the transition region as described in Section 5.3.2 and use it to fix these errors.

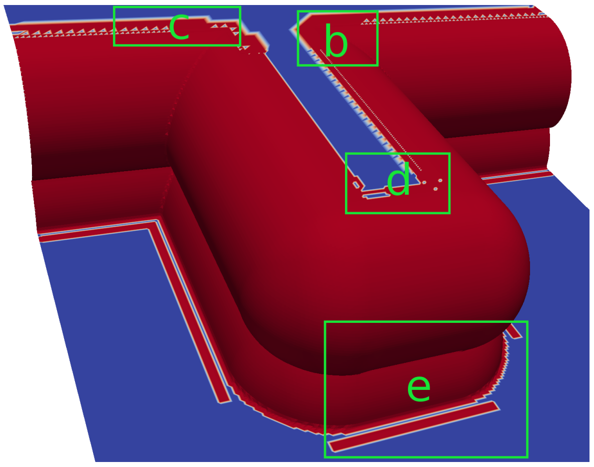
Figure 5.22(b) shows that if the tolerance value is too small, areas of the mesh with a very small curvature are detected as features. This implies that setting a very small value for the tolerance is not a feasible solution.

As discussed in Section 3.2, the curvature of a vertex is calculated by approximating an operator and a suitable surface patch around this vertex. If the vertices that make up the 1-ring neighbourhood of a vertex lie in the same plane, the curvature of this vertex is correctly calculated as zero. However, the other vertices in the 1-ring neighbourhood of this vertex can have different curvatures. These errors are shown in Figure 5.22(c) and Figure 5.22(d). They occur due to inaccuracies during the discretization process of the surface. These vertices have to be detected and their set allocation has to be adapted accordingly. A solution to fix these errors is introduced in Section 5.2.1.

The error shown in Figure 5.22(e) occurs when one coordinate of a vertex is near the minimum value of a double precision variable and the same coordinate in an adjacent vertex is just a small number, for example, the vertices $v_1 = (0, 0, 10^{-300})$ and $v_2 = (0.5, 0, 10^{-3})$. This error is again due to the discretization of the surface. It is fixed by adding the flat vertices between the two feature vertices to the set of feature vertices as described in Section 5.3.2.

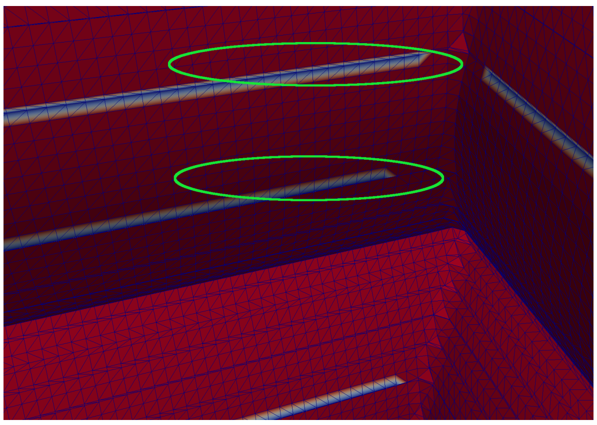


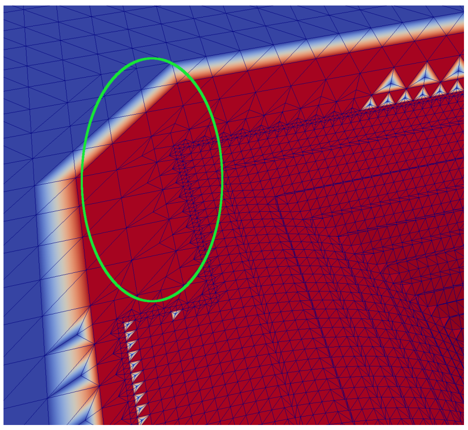
(a) $\tau = 0.5$

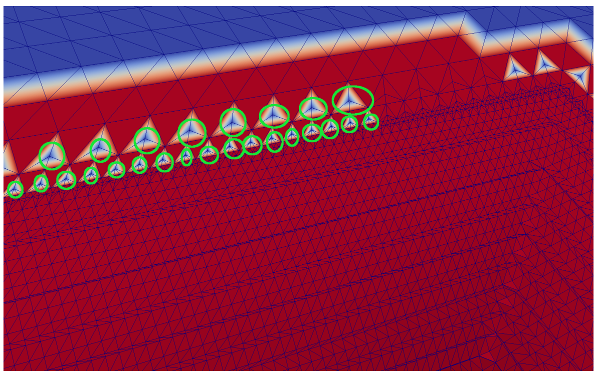


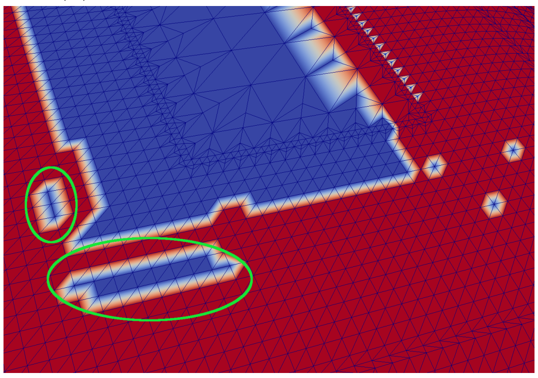
(b) $\tau = 0.0001$

Figure 5.21: Errors in the feature detection (*Smooth Bay*). The highlighted areas (a-e) refer to the zoomed areas shown in Figure 5.22.


(a) Triangles with a short edge.


(b) Flat vertices detected as feature.


(c) Error in a single vertex.


(d) Holes in features.

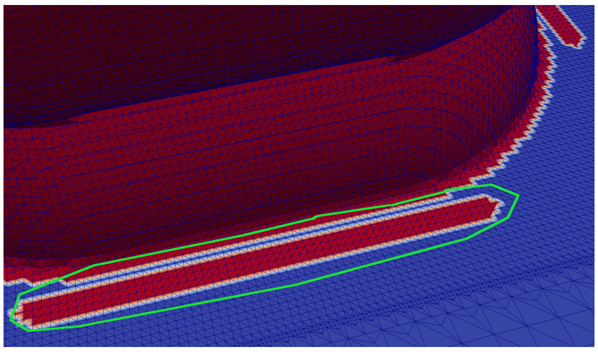

(e) Small differences in point data.

Figure 5.22: Zoomed-in errors (green circles) in the feature detection (*Smooth Bay*).

5.2.1 Filling Inaccurate Surface Patches

In the previous section it was discussed that when only the curvature of a given vertex is considered, the partition of the vertices contained errors. This is a consequence of the shape and position of triangles in the mesh. Hence, a solution is needed to find these inaccurate subsets of vertices and change their assignment from $\mathcal{M}_{\text{flat}}$ to $\mathcal{M}_{\text{feature}}$. It is straightforward to identify vertices as seen in Figure 5.22(c) and change their assignment. For holes in the partition (see Figure 5.22(d)) a more sophisticated approach has to be taken. The set $\mathcal{M}_{\text{flat}}$ has to be divided into subsets of a certain defined size that are surrounded by vertices of the set $\mathcal{M}_{\text{feature}}$. Algorithm 4 shows how this partitioning problem is solved. It can be used for a single vertex or a previously defined number of vertices.

```
input  : mesh $\mathcal{M}$, $\mathcal{M}_{\text{flat}}$, set $\mathcal{M}_{\text{feature}}$, $size \in \mathbb{R}$
output: $\mathcal{M}_{\text{flat}}$, $\mathcal{M}_{\text{feature}}$
set current-hole ← $\emptyset$;
set current-hole-tmp ← $\emptyset$;
foreach vertex $v$ in $\mathcal{M}$ do
    if $v \in \mathcal{M}_{flat}$ then
        current-hole $\cup\, v$;
        current-hole-tmp $\cup\, v$;
    end
    // in this context |·| denotes the cardinality of a set
    while $|current\text{-}hole| \leq size + 1 \wedge current\text{-}hole\text{-}tmp \neq \emptyset$ do
        $w$ ← get vertex from the set current-hole-tmp;
        current-hole-tmp ← current-hole-tmp$\setminus w$;
        foreach vertex $w_i$ in $N_1(w)$ do
            if $w_i$ is a flat vertex then
                current-hole $\cup\, w_i$;
                current-hole-tmp $\cup\, w_i$;
            end
        end
    end
    if $|current\text{-}hole| \leq size$ then
        $\mathcal{M}_{\text{feature}} \leftarrow \mathcal{M}_{\text{feature}} \cup$ current-hole;
        $\mathcal{M}_{\text{feature}} \leftarrow \mathcal{M}_{\text{flat}} \setminus$current-hole;
    end
    current-hole ← $\emptyset$; current-hole-tmp ← $\emptyset$;
end
```

Algorithm 4: Filling inaccurate surface patches.

After applying Algorithm 4 to the previously calculated partitioning the results displayed in Figure 5.23 were produced. Figure 5.24 shows that the number of vertices, were the 1-ring neighbourhood has an uneven ratio of vertices between the set $\mathcal{M}_{flat}$ and $\mathcal{M}_{feature}$, thus creating ridges in the partition, is reduced by half. This shows that filling the inaccurate surface patches has increased the quality of the feature detection. This improved version of the partition is used for the remainder of this work.

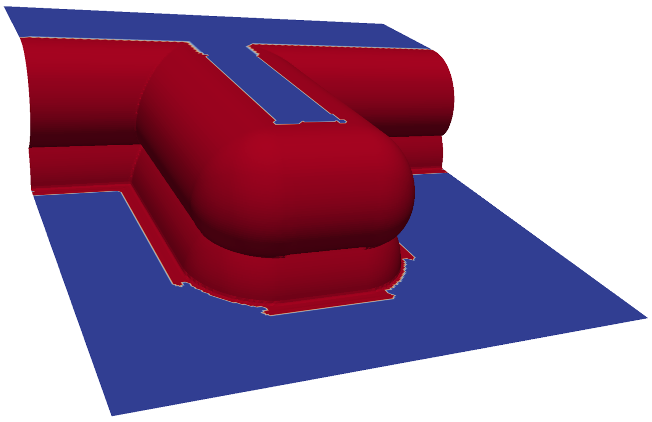


Figure 5.23: *Smooth Bay* with fixed errors.

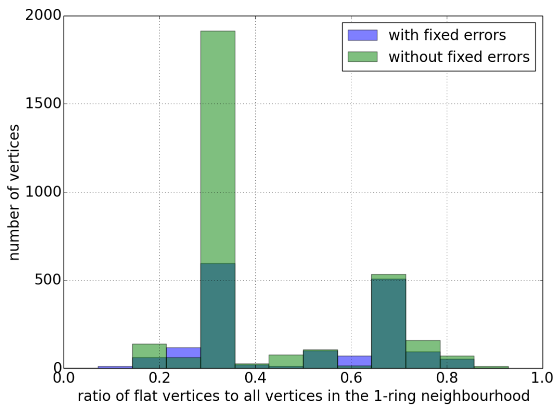


Figure 5.24: Quality of the partition with $\tau = 0.1$ after fixing the errors.

5.3 Curvature Regions of a Surface

The partitioning divides the mesh into regions, as indicated in Figure 5.23. From now on the set of flat vertices $\mathcal{M}_{\text{flat}}$ is called the **flat region** of the mesh, and the set of feature vertices $\mathcal{M}_{\text{feature}}$ is called the **feature region** of the mesh. Edges were both vertices are in the set of $\mathcal{M}_{\text{flat}}$ are called **flat edges**, and edges were both vertices are in the set of $\mathcal{M}_{\text{feature}}$ are called **curved edges**.

5.3.2 Definition
Let $\mathcal{M}$ be a mesh, $V_{\mathcal{M}}$ be the set of the vertices of $\mathcal{M}$, and let $f(v) : V_{\mathcal{M}} \to \{0, 1\}$ be a function. Then a **region** R of $\mathcal{M}$ is defined as

$$R := \{v \in V_{\mathcal{M}} | f(v) = 1\}.$$

The edges of a mesh that connect two different regions play a significant role in the further investigations.

5.3.3 Definition
Let $\mathcal{M}$ be a Mesh, $E_{\mathcal{M}}$ the set of edges in $\mathcal{M}$, and R_1, R_2 be 2 regions of $\mathcal{M}$. Then the set $CE \subset E_{\mathcal{M}}$ is called the **set of critical edges** if

$$CE := \{e \in E_{\mathcal{M}} | (e_v \in R_1 \wedge e_w \in R_2) \vee (e_w \in R_1 \wedge e_v \in R_2)\},$$

Elements of CE are called **critical edges**.

This Section discusses the simplification of these regions of the mesh with different approaches as well as different parameters, e.g., region specific placement functions or region specific stop conditions. The study of the results of this simplification approach illustrates some problems with the element quality of certain elements of the simplified mesh. A possible solution to this problem is proposed in Section 5.3.2.

In the discussion in the following section the scale of a mesh is needed so the following definition is given:

5.3.4 Definition
Let $\mathcal{M}$ be a mesh, $l_{avg}(\mathcal{M})$ the average edge length of $\mathcal{M}$, and $l_{BB}(\mathcal{M})$ the maximum edge length of the *Bounding Box* [12] of $\mathcal{M}$.

Then the **scale** of a mesh is defined as:

$$\frac{l_{avg}(\mathcal{M})}{l_{BB}(\mathcal{M})}.$$

5.3.1 Simplification of Regions

When simplifying the flat regions of a mesh it is clear that the local geometry of the mesh is planar. When calculating the new position of a vertex the attention can be focused on the shape of the triangles around this vertex and the preservation of the border of the mesh if the collapsed edge was a border edge. This can be achieved by minimizing the distance from each edge in the 1-ring neighbourhood of the new vertex v. Let $\operatorname{dist}(v, w) := \|v - w\|$ denote the distance between the two vertices v and w. Then the minimization problem can be formulated as follows

$$\sum_{v_i \in N_1(v)} \operatorname{dist}(v, v_i)^2 \to \min.$$

The edge that yields the best shape for all incident triangles after it is collapsed is chosen as the edge to collapse. The position of the new vertex is calculated via a minimization problem.

On the other hand, when simplifying the feature regions of the mesh the Lindstrom-Turk simplification algorithm is applied (see Section 2.2.2).

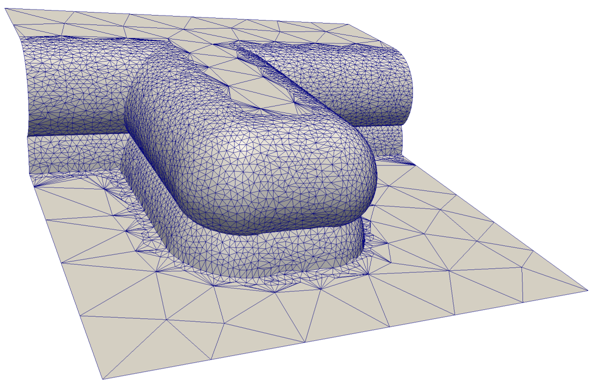
Remark: Both of the methods discussed above can be implemented by using the implementation of the Lindstrom-Turk algorithm in CGAL [4].

Since the flat region of the mesh can be simplified to a larger degree than the feature region, without the loss of geometric information, the maximum edge length for each region is taken as the termination function (see Algorithm 5). The maximum edge length for each region is a user supplied parameter, which has to be chosen in concordance with the scale of the supplied mesh (see Definition 5.3.4) and the desired level of simplification.

```
input  : edge $e$ of the mesh $\mathcal{M}$, $l \in \mathbb{R}$
output: stop the simplification process
// $\|e\| := dist(e_v, e_w)$ were $e_v$ and $e_w$ are the two vertices of the edge $e$
if $\|e\| \leq l$ then
  | false;
else
  | true;
end
```

Algorithm 5: Edge length termination function.

When applying the simplification techniques discussed above with different parameters on the two test meshes, the results shown in Figure 5.25 are obtained. Together with the histograms in Figure 5.26 it is shown that this way of simplifying the mesh leads to bad triangles (see Section 2.1.2) at the border between the flat and feature region of the mesh. The edges that are responsible for the bad triangles are the critical edges; handling those will be discussed in the next section.


(a) Curved region of *Smooth Bay* simplified until the edge length of curved edges is 0.03 and the length of flat edges is 0.3.

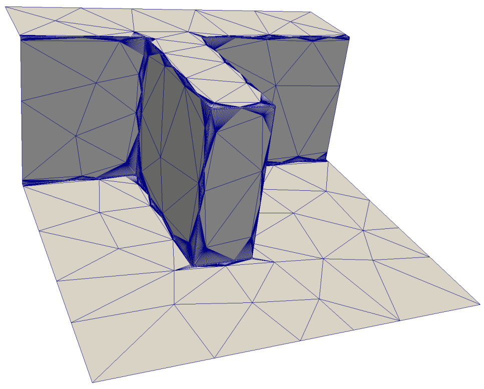
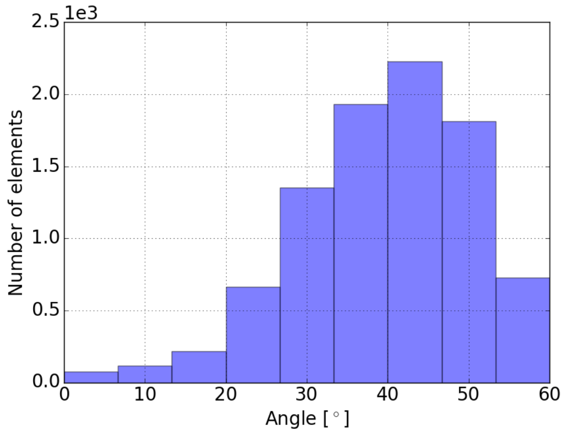
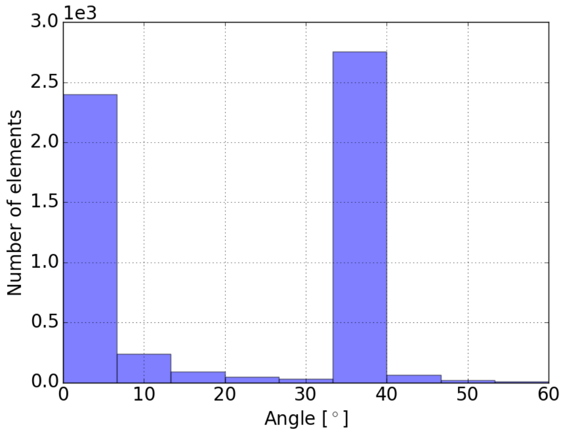

(b) Flat region of *Square Bay* simplified until the edge length of flat edges is 0.3.

Figure 5.25: Simplification of the flat and feature regions. The edge lengths are chosen in accordance with the scale of the mesh (see Definition 5.3.4).



(a) *Smooth Bay* curved length 0.03 flat length 0.3 minimum angles.



(b) *Square Bay* flat length 0.3 minimum angles.

Figure 5.26: Triangle quality after the simplification.

5.3.2 Transitions between Regions

The previous section discussed how the critical edges between the flat and feature region of the mesh lead to bad triangles after the simplification process. To create a smoother transition, in the sense of triangle quality, a third mesh region is introduced: The **transition region** $\mathcal{M}_{\text{transition}}$. The transition region creates a buffer between the elements of the flat and feature regions of the mesh. The idea is to find suitable edge lengths between the average edge length of the flat and feature region to avoid creating triangles that have one short edge or one large angle.
Before describing the process of creating the transition region, additional notations have to be introduced.

5.3.5 Definition
Let $\mathcal{M}$ be a mesh, $V_{\mathcal{M}}$ be the set of all vertices in $\mathcal{M}$, and $E_{\mathcal{M}}$ the set of all edges in $\mathcal{M}$. Then a **path** P of length n is defined as

$$P = (v_1, v_2, \dots, v_n) \in V_{\mathcal{M}} \times V_{\mathcal{M}} \times \cdots \times V_{\mathcal{M}},$$

were v_i is connected to v_{i+1} through an edge $e \in E_{\mathcal{M}}$.

5.3.6 Definition
Let P be a path of length n in $\mathcal{M}$. Then the **distance** of the path P is defined as

$$d(P) := \sum_{i=1}^{n-1} \text{dist}(v_i, v_{i+1}).$$

The calculation of the transition region starts with a collection phase in which the flat vertices of all critical edges from the flat region to the feature region are stored in a set. After the collection phase ends, the transition region ”grows” from the feature region into the flat region until each vertex of $\mathcal{M}_{\text{flat}}$ has a distance value associated with it (see Algorithm 6).

Remark: Until this point only sets were needed to describe the algorithms in this work. In Algorithm 6 a more complex data structure, that provides additional functionality, is needed. To that end, a Hash-List L of vertices is defined as follows: The elements in a Hash-List are ordered. So the first element of a Hash-List can be accessed *L.first()* and an element can be added at the end of the Hash-List *L.pushback(element)*. Additionally, elements of the Hash-List can be accessed as *L(v)*, were v is a vertex.

It was mentioned in Section 5.2 that the calculation of the transition region is used to fix errors in the feature detection (see Figure 5.22(e)). Each vertex in the list after the collection phase is checked and added to $\mathcal{M}_{\text{feature}}$, if it is only surrounded by other feature vertices or vertices in the set after the collection phase (see Algorithm 7).

Figure 5.27 shows the transition region calculated with Algorithm 6 for both test meshes. Now that the transition regions are calculated the simplification of the transition regions remains to be discussed.

```
input  : mesh $\mathcal{M}$, set $\mathcal{M}_{\text{feature}}$, set $\mathcal{M}_{\text{flat}}$
output: $\mathcal{M}_{\text{transition}}$
Hash-List transition-region $\leftarrow \emptyset$;
Hash-List transition-distances $\leftarrow \emptyset$;
set $\mathcal{M}_{\text{transition}} \leftarrow \emptyset$;
foreach critical edge e in $\mathcal{M}$ do
    transitiona-region.pushback(flat vertex of $e$);
    transition-distances(flat vertex of $e$) $\leftarrow$ dist($e_v, e_w$);
end
while transition-region $\neq \emptyset$ do
    $v \leftarrow$ transition-region.first();
    transitiona-region $\leftarrow$ transitiona-region$\backslash v$;
    $\mathcal{M}_{\text{transition}} \cup v$;
    foreach vertex $v_i$ in $N_1(v)$ do
        if $v_i \notin \mathcal{M}_{feature}$ then
            if transition-distances($v_i$) $\leq$ transition-distances(v) + dist($v, v_i$) then
                transition-distances($v_i$) $\leftarrow$ transition-distances($v$) + dist($v, v_i$);
            end
            transitiona-region $\leftarrow$ transitiona-region $\cup v_i$;
        end
    end
end
```

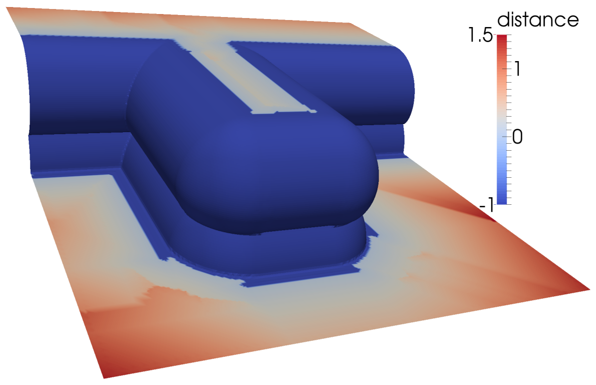
Algorithm 6: Calculating the transition region.

```
input  : mesh $\mathcal{M}$, set $\mathcal{M}_{\text{flat}}$, set $\mathcal{M}_{\text{feature}}$
output: $\mathcal{M}_{\text{flat}}$, $\mathcal{M}_{\text{feature}}$
set transition-region $\leftarrow \emptyset$;
foreach critical edge e in $\mathcal{M}$ do
    transitiona-region $\cup$ flat vertex of $e$;
end
foreach vertex v in transitiona-region do
    foreach vertex $v_i$ in $N_1(v)$ do
        if $v_i \notin$ transitionare-region $\vee v_i \notin \mathcal{M}_{feature}$ then
            GOTO next vertex;
        end
    end
    $\mathcal{M}_{\text{flat}} \backslash v$;
    $\mathcal{M}_{\text{feature}} \cup v$;
    next vertex:
end
```

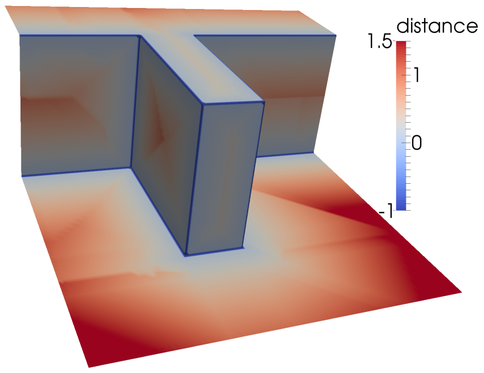
Algorithm 7: Fixing errors in the feature detection.

Starting from the critical edges of the mesh, the transition regions are simplified with a certain edge length $l_1 \in \mathbb{R}$.

After this first simplification step the transition regions are simplified with another, larger edge length $l_2 \in \mathbb{R}$. This time the starting positions for the simplification process are not the critical edges of the mesh, but the edges of the mesh that are a certain distance $l_1 * s$ away from the critical edges of the mesh. The parameter $s \in \mathbb{R}$ is a user supplied parameter, which has to be chosen with respect to the scale of the mesh (see Definition 5.3.4) and the desired element quality. This process repeats until the transition regions are simplified with all provided lengths $l_1, l_2, \ldots, l_n$. Algorithm 8 and Figure 5.28 shows the process and some results, respectively.



(a) Transition region *Smooth Bay.*



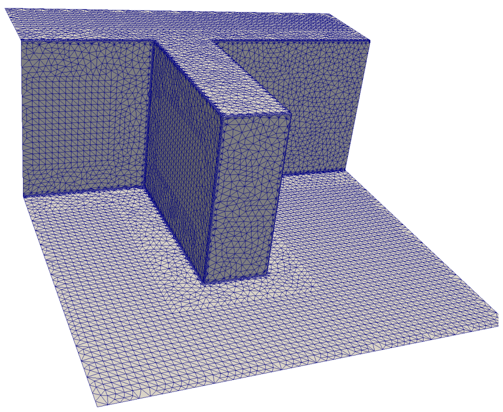
(b) Transition region *Square Bay.*

Figure 5.27: Transition region of both test meshes. The dark blue areas with the value -1 are $\mathcal{M}_{\text{feature}}$, the coloured areas transitioning from blue to red are the transition regions. The distance is defined as in Definition 5.3.5.

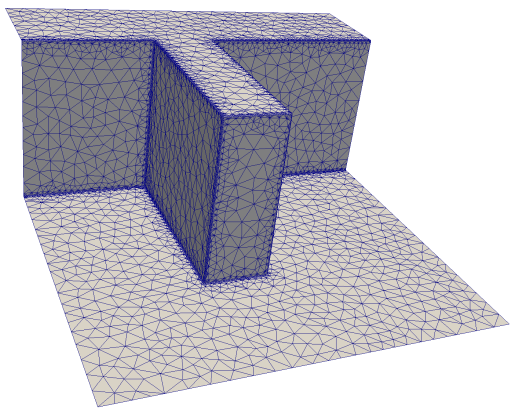
```
input  : mesh $\mathcal{M}$, set $\mathcal{M}_{\text{transition}}$, list edge-length, transition-size $\in \mathbb{R}$
output: mesh $\mathcal{M}_{simplified}$
start = 0.0;
foreach length l in edge-length do
    simplify $\mathcal{M}_{\text{transition}}$ with an edge length of $l$ starting at distance start ;
    start = start + transition-size $*l$
end
```

Algorithm 8: Simplifying the transition region.

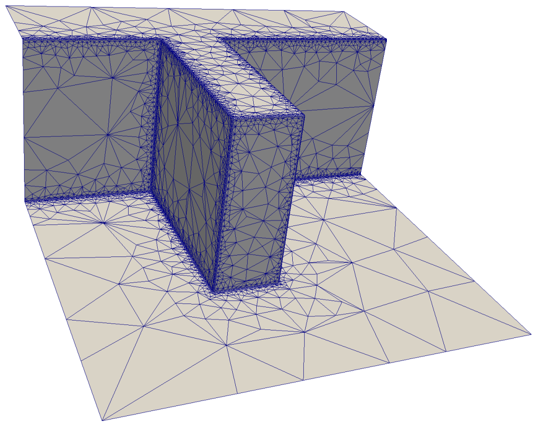
When the methods discussed in this section are applied to the two test meshes the results shown in Figure 5.29 are produced. In Figure 5.30 it can be seen that the triangle quality significantly improves due to the transition region. In the next Chapter 6.1 the results from this method are evaluated and more examples are given.



(a) $l_1 = 0,03$

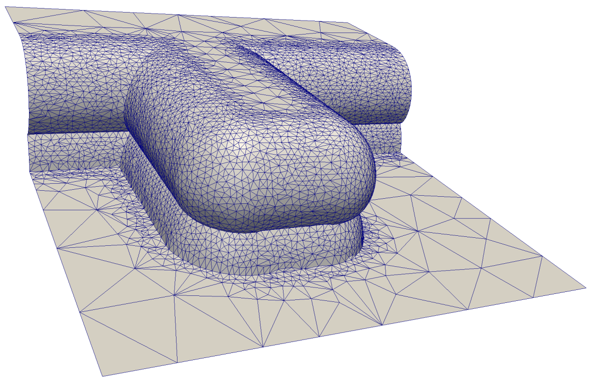


(b) $l_1 = 0,03$, $l_2 = 0,09$

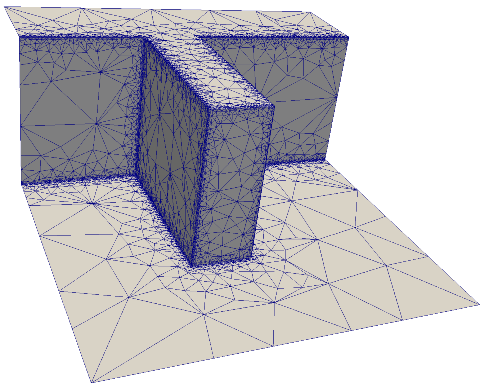


(c) $l_1 = 0,03$, $l_2 = 0,09$, $l_3 = 0,3$

Figure 5.28: Visual example of the successive simplification described in Algorithm 8 on the mesh *Square Bay*. l_1, l_2, l_3 are the maximum edge length of the three regions used in this example.

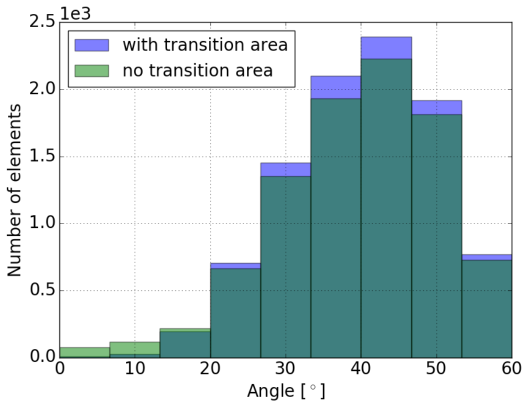


(a) *Smooth Bay* simplified with transition region.

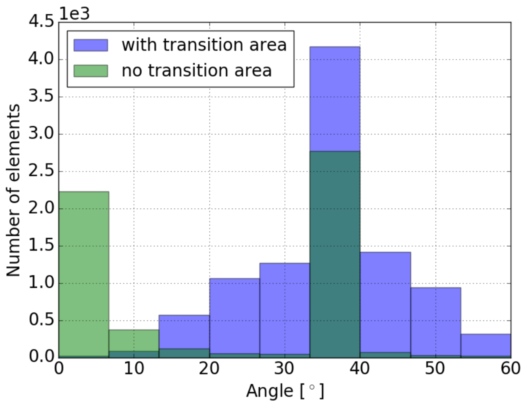


(b) *Square Bay* simplified with transition region (see Figure 5.28(c)).

Figure 5.29: Example of of both test meshes simplification with transition region.



(a) *Smooth Bay*



(b) *Square Bay*

Figure 5.30: Minimal angle distribution after simplification with transition region.

Chapter 6

Results and Discussion

This chapter discusses the results gained from the approach of subdivided surface simplification presented in Chapter 5, which will be further referred to as *region simplification*. The approach presented in the previous chapter is an extension of the Lindstrom-Turk simplification algorithm. Both methods are compared with respect to execution time and resulting mesh quality. Finally, a short summary is given together with potential further improvements of the method.

6.1 Surface Mesh Simplification Comparison

This section discusses several applications of the *region simplification* approach and compares them with the Lindstrom-Turk simplification approach.

The investigations in this section are structured in the following way. At first, only the flat regions of the meshes are simplified and appropriate transition regions are calculated to ensure a good element quality. These results are then compared to the Lindstrom-Turk simplification strategy. To guarantee a baseline for a fair comparison between these two approaches the mesh simplified with the Lindstrom-Turk approach is simplified until it reaches the same amount of vertices as the *region simplification*.

Although the number of vertices in the simplified meshes is the same, the number of elements differs. This happens because an edge collapse of an inner edge of the mesh removes two elements of the mesh (see Section 2.2.1). However, an edge collapse of a border edge of the mesh only removes one element, since there is only one element that can be removed. Thus, the Lindstrom-Turk simplification process removes more elements than *region simplification*, because it removes a larger amount of inner edges.

Finally, four quality metrics of both meshes are compared: the minimum angle distribution, the radius ratio, the distance of each vertex in the simplified mesh to the original mesh, and the execution time of both approaches. The results are averaged over 100 runs. Afterwards, the findings from those metrics are discussed and used to motivate further investigations.

6.1.1 Square Bay Mesh

The first investigation, using the *Square Bay* mesh, uses the parameters shown in Table 6.1. The number in the first column in Table 6.1 is the value for 'transition-size' in Algorithm 8. The number in the second column is the minimum edge length to stop the simplification of the feature region (see Algorithm 5).

Region size	Feature region	Flat region 1	Flat region 2	Flat region 3	Flat region 4
1.5	0.0	0.03	0.06	0.12	0.25

Table 6.1: Investigation 1 *Square Bay* parameters for region simplification.

	Region simpl.	Lindstrom-Turk simpl.
elements	10,617	10,500
vertices	5,353	5,353

Table 6.2: Investigation 1 *Square Bay*, elements and vertices in the mesh after the simplification.

The last four numbers in the columns of Table 6.1 are the entries in the list 'edge-length' in Algorithm 8. These parameters were chosen in such a way that the elements of the mesh maintain a good shape (see Section 2.1.2). This yields the mesh shown in Figure 6.1(a) with the number of elements shown in Table 6.2.

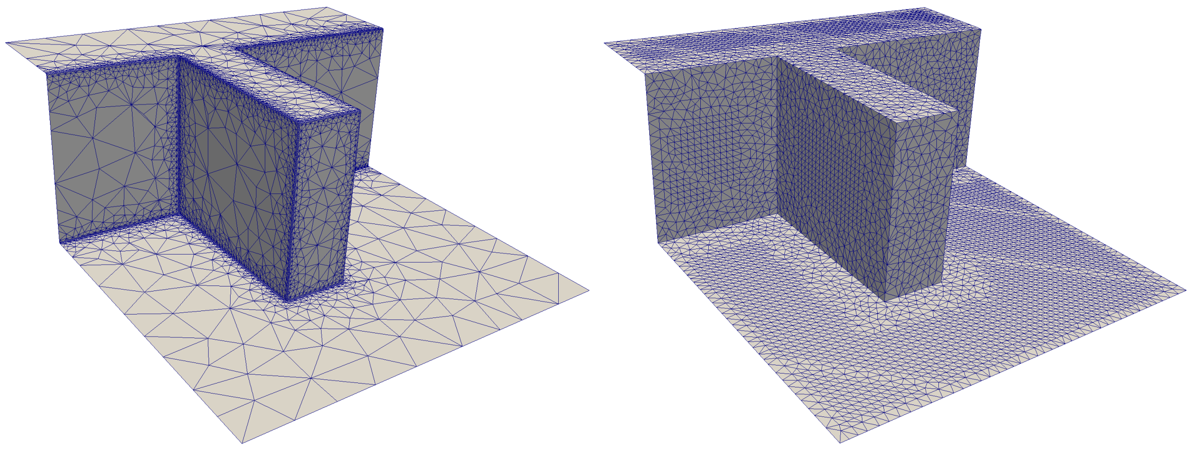
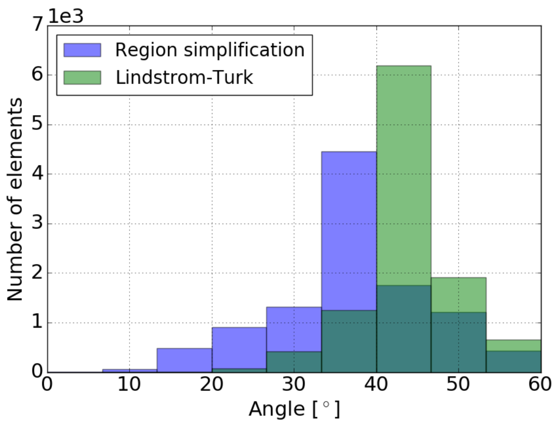
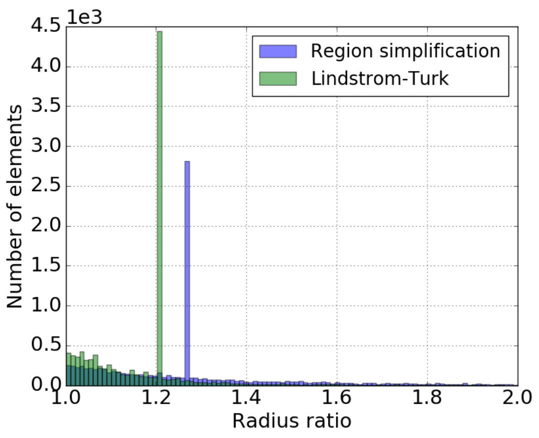

(a) Region simplification (b) Lindstrom-Turk simplification

Figure 6.1: Investigation 1 *Square Bay* output.

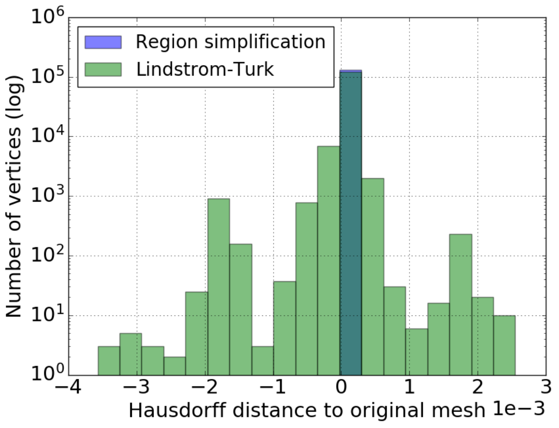
Figure 6.2 compares the metrics of the results gained from both simplification processes. The approach of Lindstrom-Turk changes the geometry of the mesh, this can be seen in Figure 6.2(c) since the Lindstrom-Turk simplification has vertices with a Hausdorff distance greater than 0 to the original mesh. On the other hand *region simplification* has a Hausdorff distance of 0 to the original mesh, and thus does not change the geometry of the original mesh. This behaviour is a consequence of the fact that the feature region (see Section 5.3) of the mesh is not simplified. Additionally the transition region (see Section 5.3.2) prevents the formation of bad elements like needles. However, Figure 6.2(a) shows that on average *region simplification* has a lower element quality. The elements with the lower quality are the elements that connect two different regions and thus, cannot maintain a higher element quality. The calculation time is about 30 % in favour of the Lindstrom-Turk simplification (see Figure 6.3).



(a) Minimum angles of all elements.



(b) Radius ratio of all elements.



(c) Vertex distance to the original mesh.

Figure 6.2: Investigation 1 *Square Bay* quality.

In this first investigation the *region simplification* managed to preserve the geometry of the original mesh and reduce the number of vertices in the mesh by about 92 %. In the next investigation the feature region is simplified to get an overall lower amount of vertices.

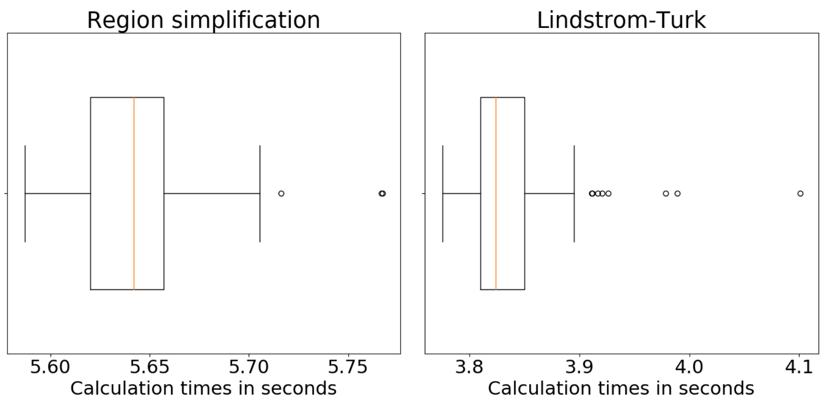


Figure 6.3: Investigation 1 *Square Bay* calculation times.

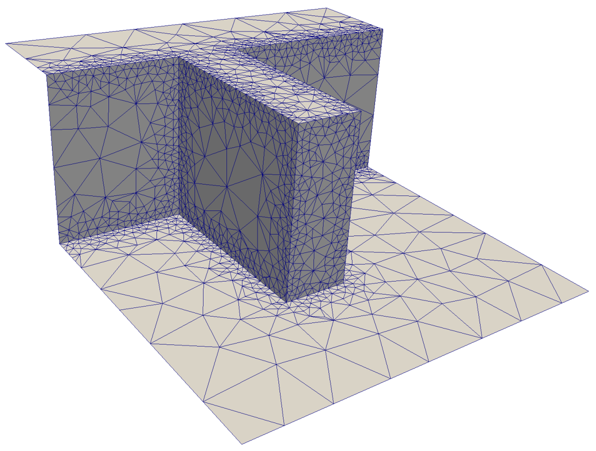
In the next investigation the feature region of the mesh *Square Bay* is simplified with the parameters shown in Table 6.3. In Table 6.4 the remaining number of elements can be seen, and Figure 6.4(a) displays the mesh after the simplification

Region size	Feature region	Flat region 1	Flat region 2	Flat region 3	Flat region 4
1.2	0.04	0.06	0.1	0.2	0.3

Table 6.3: Investigation 2 *Square Bay* parameters for region simplification.

	Region simpl.	Lindstrom-Turk simpl.
elements	3,610	3,546
vertices	1,838	1,838

Table 6.4: Investigation 2 *Square Bay*, elements and vertices in the mesh after the simplification.


(a) Region simplification

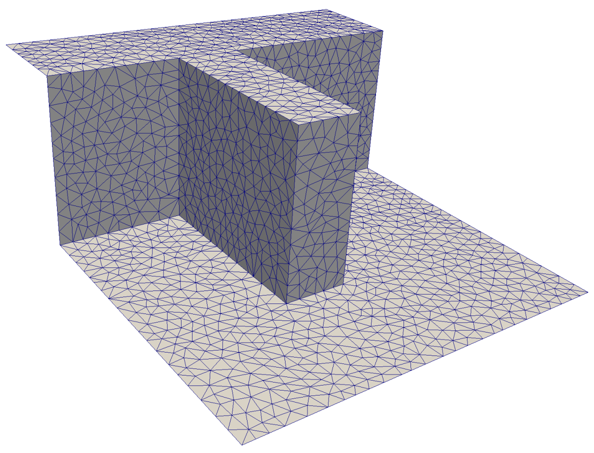
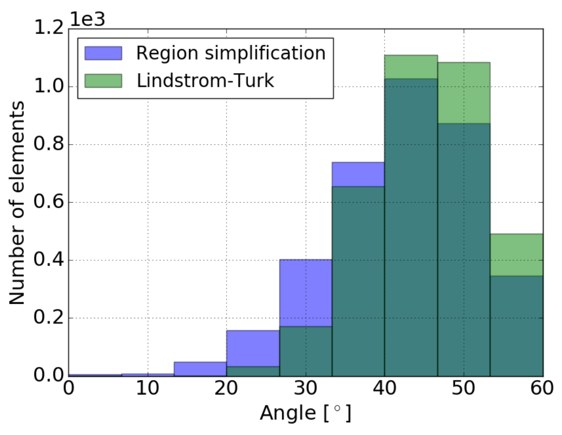
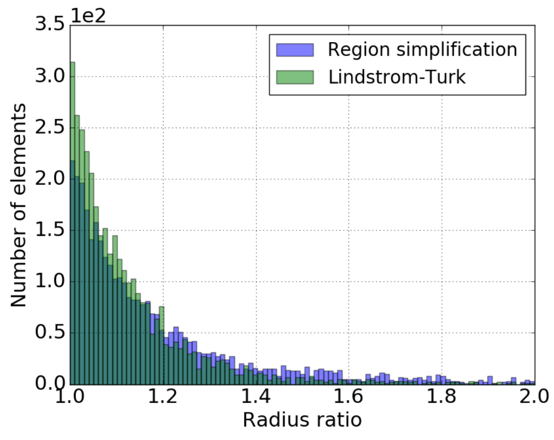

(b) Lindstrom-Turk simplification

Figure 6.4: Investigation 2 *Square Bay* output.

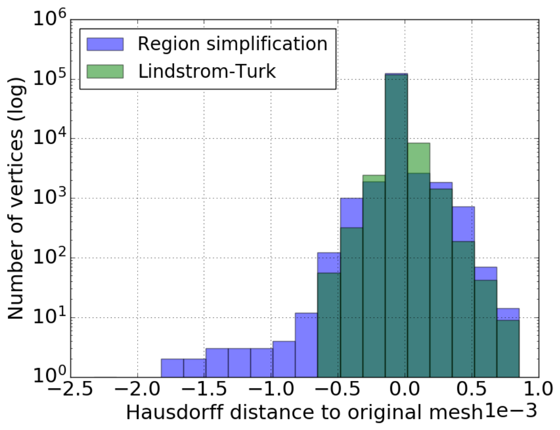
Figure 6.5, and in particular Figure 6.5(c) clearly show that the previously preserved geometry of the original mesh is no longer sustained when simplifying the feature region of the mesh *Square Bay*. The element quality improved with respect to the first investigation (see Figure 6.5(a) and 6.5(b). However, this is because the minimum edge length of the feature and transition regions are closer together. Figure 6.6 depicts that the difference in the average calculation time dropped to about 16 % in favour of the Lindstrom-Turk simplification.



(a) Minimum angles of all elements.



(b) Radius ratio of all elements.



(c) Vertex distance to the original mesh.

Figure 6.5: Investigation 2 *Square Bay* quality.

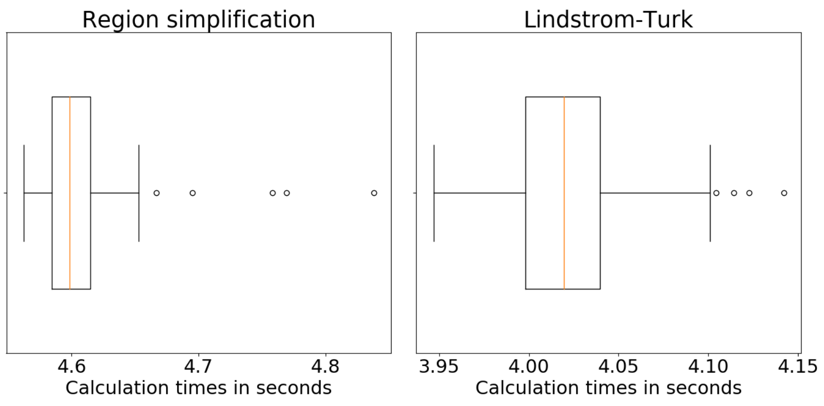


Figure 6.6: Investigation 2 *Square Bay* calculation times.

6.1.2 Smooth Bay Mesh

The mesh *Smooth Bay* originally has 57,058 feature vertices whereas the previous mesh, *Square Bay*, has 2,765 which is a significantly higher number. This fact leads to a greater variety in possible results. The mesh shown in Figure 6.7(a) was simplified with the parameters shown in Table 6.5.

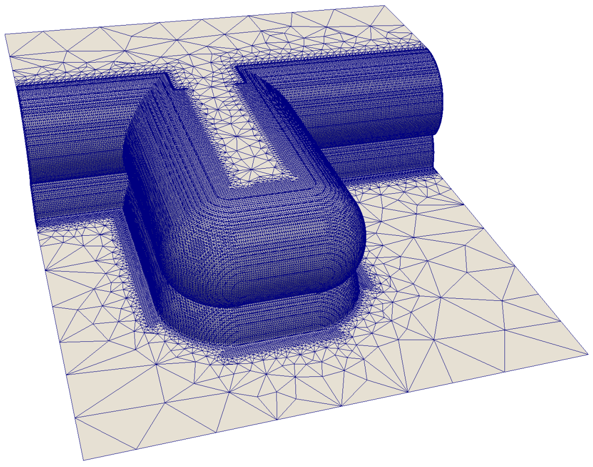
Region size	Feature region	Flat region 1	Flat region 2	Flat region 3	Flat region 4
1.5	0.0	0.02	0.06	0.12	0.25

Table 6.5: Investigation 1 *Smooth Bay* parameters for region simplification.

The remaining number of elements and vertices are shown in Table 6.6.

	Region simpl.	Lindstrom-Turk simpl.
elements	115,790	115,493
vertices	58,183	58,183

Table 6.6: Investigation 1 *Smooth Bay*, elements and vertices in the mesh after the simplification.


(a) Region simplification

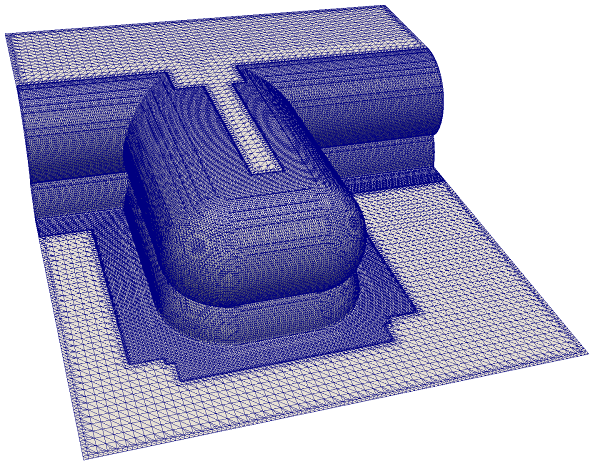
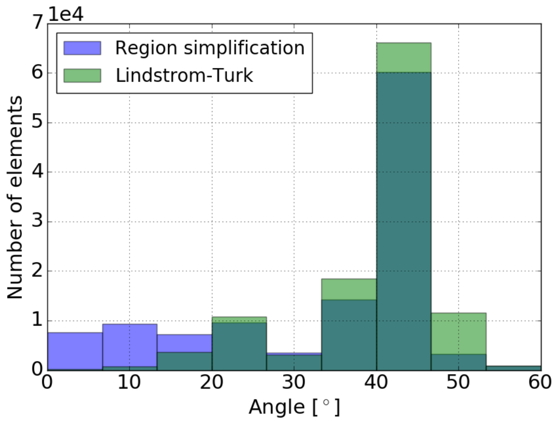
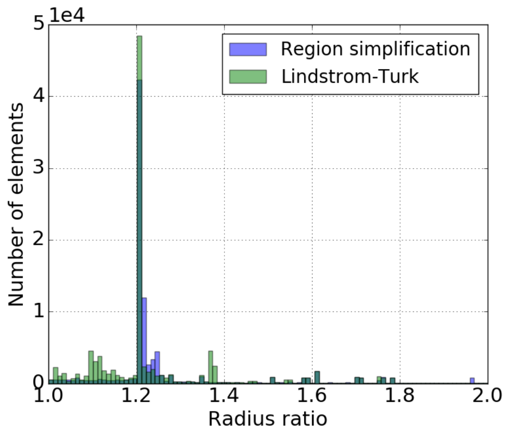

(b) Lindstrom-Turk simplification

Figure 6.7: Investigation 1 *Smooth Bay* output.

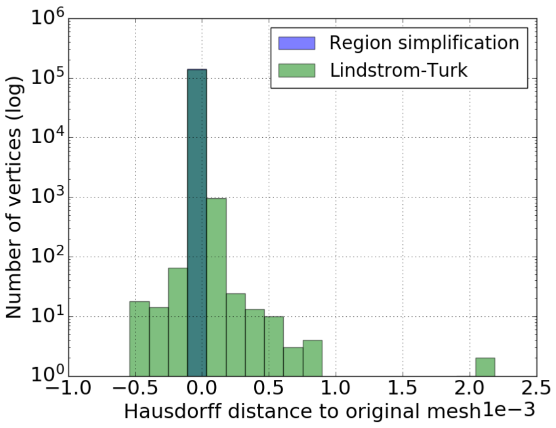
Figure 6.8 depicts key metrics of the simplification processes. Figure 6.8(a) shows that the simplified mesh has several elements with a minimum angle smaller than 20 degree. These are bad elements and are already part of the original mesh *Smooth Bay* (see Figure 6.9). They are not created by the simplification, but are a consequence of the discretization process. However, when considering Figure 6.8(c) it can be seen, as expected since the feature region is not simplified, that the *region simplification* keeps the original geometry in tact. Figure 6.10 shows that the difference in calculation time is about 60 % in favour of the Lindstrom-Turk simplification approach. This is the case, because the feature detection needs additional computation time during the *region simplification* that is not needed by the Lindstrom-Turk simplification.



(a) Minimum angles of all elements.



(b) Radius ratio of all elements.



(c) Vertex distance to the original mesh.

Figure 6.8: Investigation 1 *Smooth Bay* quality.

These results suggest that in the case of *Smooth Bay* the overall element quality could be improved by simplifying the feature region of the mesh with a very small edge length, while maintaining a low distance to the original mesh.
Based on these insights, Figure 6.11(a) depicts another investigation using the parameters as listed in Table 6.7:

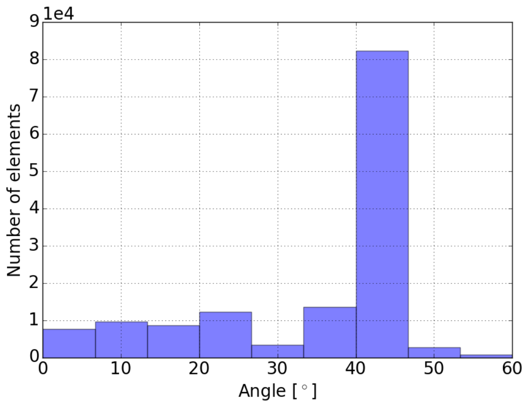


Figure 6.9: Minimum angles of the original mesh *Smooth Bay*.

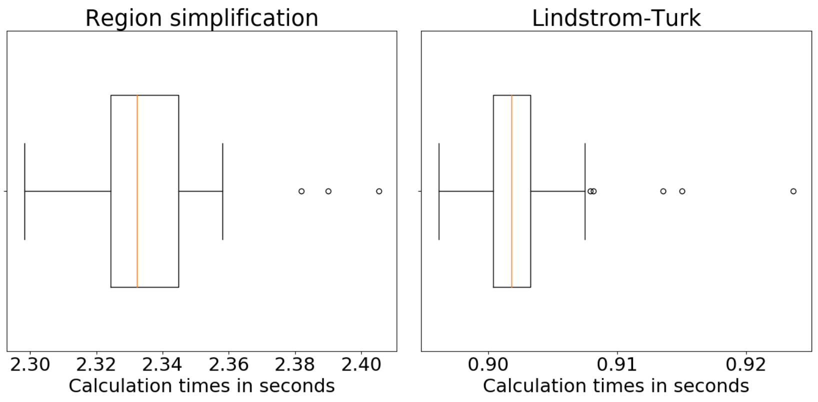


Figure 6.10: Investigation 1 *Smooth Bay* calculation times.

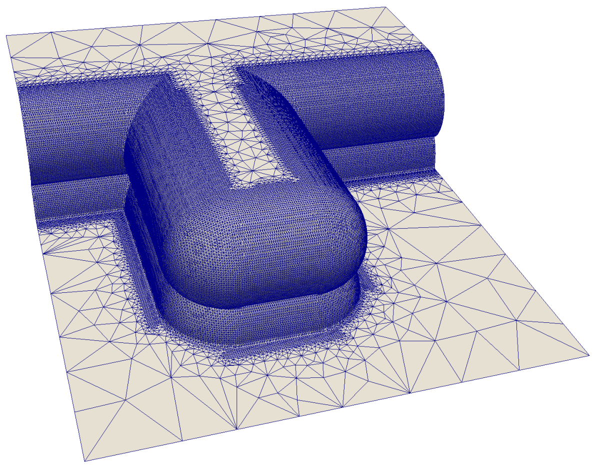
Region size	Feature region	Flat region 1	Flat region 2	Flat region 3	Flat region 4
1.5	0.009	0.02	0.06	0.12	0.25

Table 6.7: Investigation 2 *Smooth Bay* parameters for region simplification.

The remaining number of elements and vertices are shown in Table 6.8.

	Region simpl.	Lindstrom-Turk simpl.
elements	79,369	78,994
vertices	39,798	39,798

Table 6.8: Investigation 2 *Smooth Bay*, elements and vertices in the mesh after the simplification.


(a) Region simplification

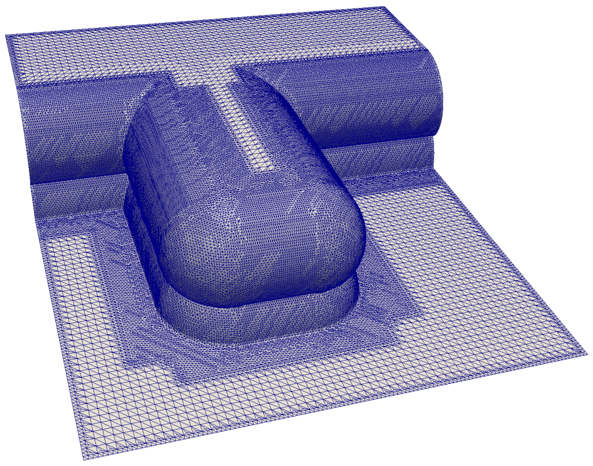
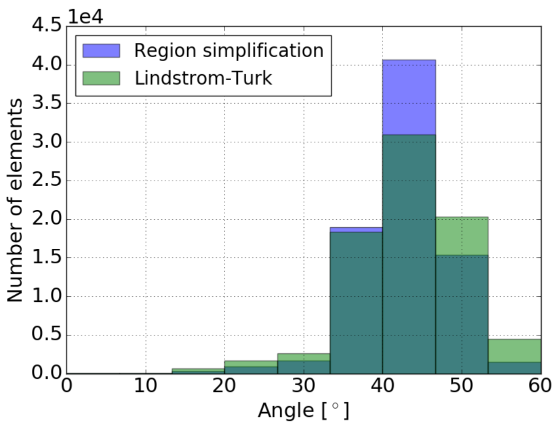

(b) Lindstrom-Turk simplification

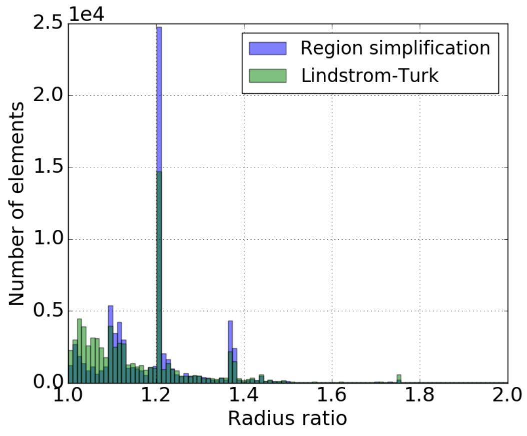
Figure 6.11: Investigation 2 *Smooth Bay* output.

The results of this investigation are shown in Figure 6.12. Figure 6.12(a) proves that the element quality has improved and that the distance to the mesh with no simplification in the feature region increased only by a small amount (see Figure 6.13). The average difference in calculation time is approximately 30 % in favour of the Lindstrom-Turk simplification (see Figure 6.14). This investigation proves that *region simplification* preserves the original geometry of the mesh in more details and additionally improves the element quality.

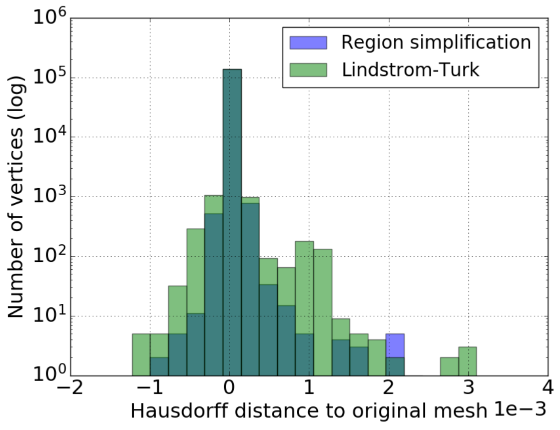
Considering the results of the last two investigations the number of vertices was only reduced by 18 and 44 %, respectively. Hence, in the next investigation, the focus will be on reducing the number of vertices and elements.



(a) Minimum angles of all elements.



(b) Radius ratio of all elements.



(c) Vertex distance to the original mesh.

Figure 6.12: Investigation 2 mesh *Smooth Bay* quality.

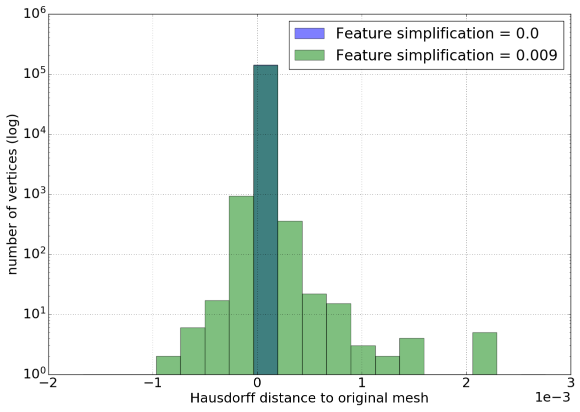


Figure 6.13: Distance between simplified and not simplified feature region of the mesh *Smooth Bay*.

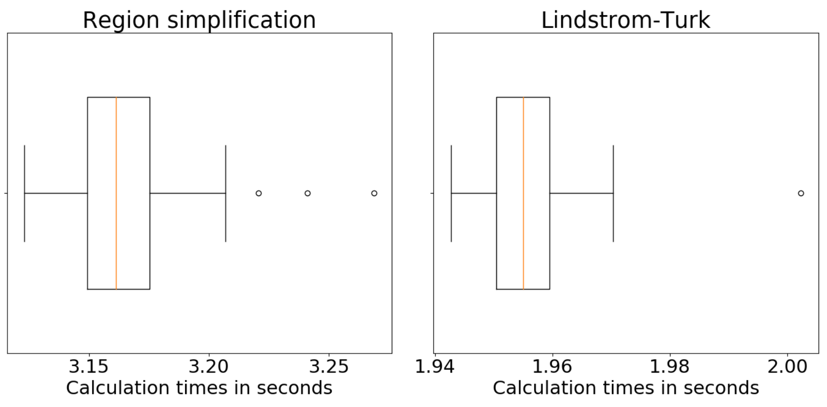


Figure 6.14: Investigation 2 *Smooth Bay* calculation times.

The mesh *Smooth Bay* has a large ratio of feature vertices, approximately 79 %. To reduce the overall number of elements by a larger amount, this region has to be simplified to a higher degree. Figure 6.15(a) shows the mesh simplified with the parameters shown in Table 6.9:

Region size	Feature region	Flat region 1	Flat region 2	Flat region 3
1.3	0.04	0.06	0.12	0.25

Table 6.9: Investigation 3 *Smooth Bay* parameters for region simplification.

The remaining number of elements and vertices are shown in Table 6.10.

	Region simpl.	Lindstrom-Turk simpl.
elements	6,284	6,179
vertices	3,186	3,186

Table 6.10: Investigation 3 *Smooth Bay*, elements and vertices in the mesh after the simplification.

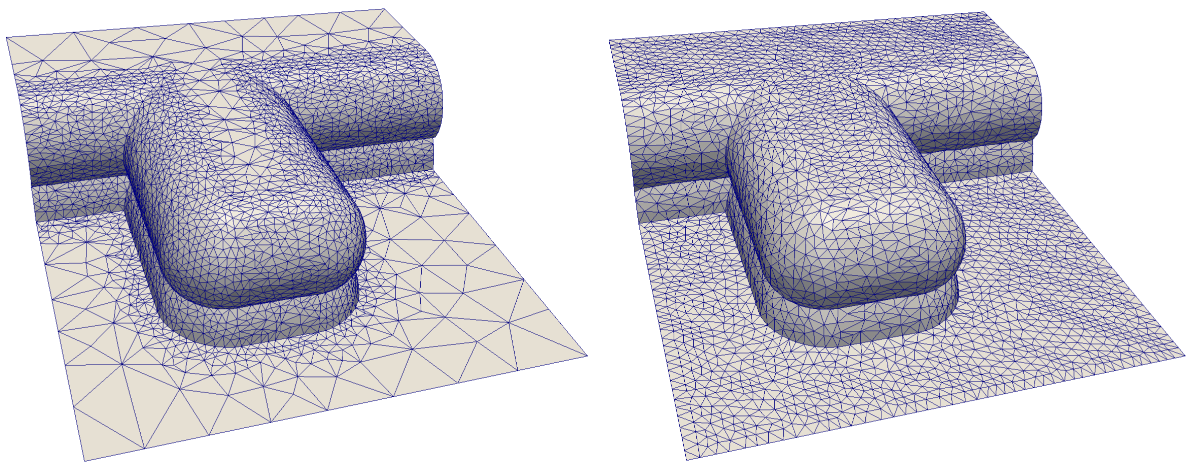
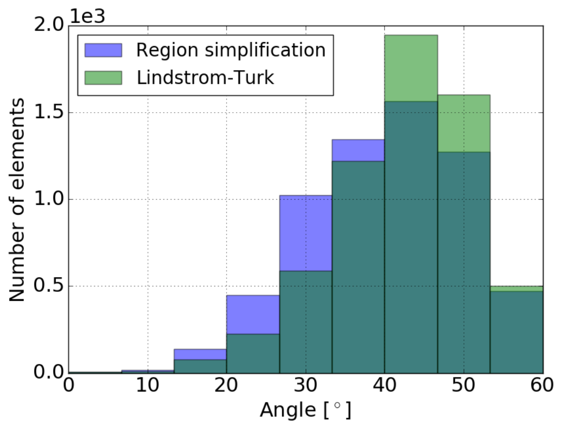

(a) Region simplification (b) Lindstrom-Turk simplification

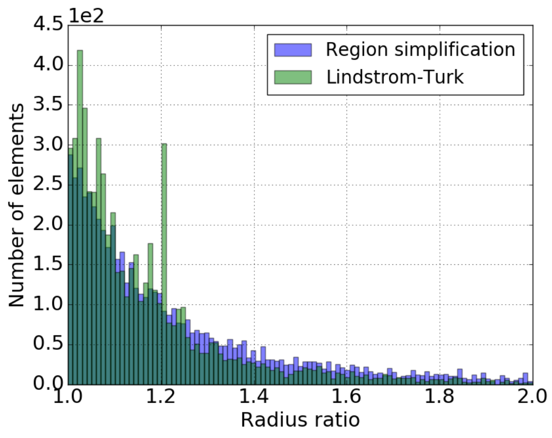
Figure 6.15: Investigation 3 *Smooth Bay* output.

Figure 6.16 displays the results of this investigation. The element quality is not as high as with the Lindstrom-Turk simplification as seen in Figure 6.16(a). However, a lower distance to the original mesh is maintained (see Figure 6.16(c)). The lower element quality is the result of the minimal edge lengths between the feature and the transition regions. Figure 6.17 shows that the difference in calculation time is still about 15 % faster when considering the best case for the approach of Lindstrom and Turk and the worst case for region simplification. However, when the average case is considered the difference in calculation time is lower than 8 %.

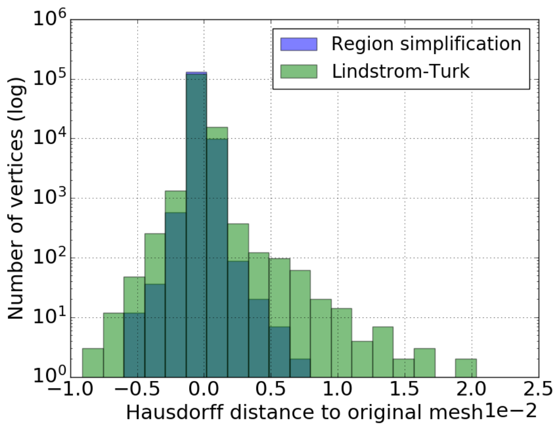
When studying all previous histograms showing the distances to the original mesh *Smooth Bay*, it can be seen that the differences in the distance to the original geometry shrinks for both methods. The next investigation thus simplifies the mesh in such a way that the feature region of the mesh is simplified by a high degree to show the limitations of *region simplification*.



(a) Minimum angles of all elements.



(b) Radius ratio of all elements.



(c) Vertex distance to the original mesh.

Figure 6.16: Investigation 3 *Smooth Bay* quality.

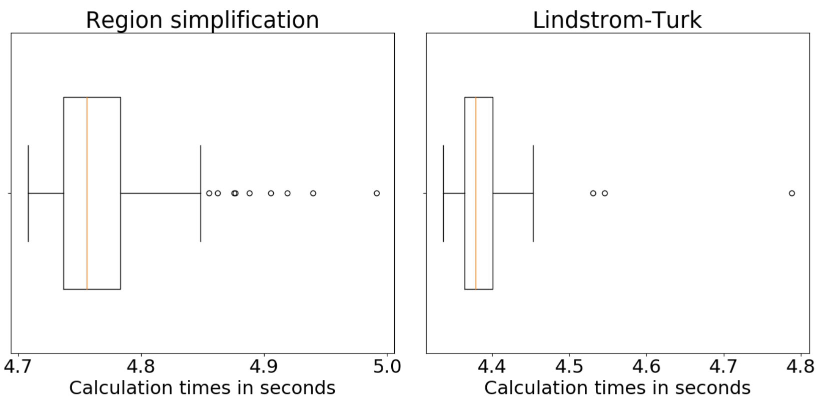


Figure 6.17: Investigation 3 *Smooth Bay* calculation times.

In this investigation, the parameters listed in Table 6.11 are used. These were chosen in such a way that the total number of vertices in the simplified mesh is about 1 to 2 % of the number vertices in the original mesh (see Figure 6.18(a) and Table 6.12).

Region size	Feature region	Flat region 1	Flat region 2	Flat region 3
1.0	0.075	0.09	0.18	0.3

Table 6.11: Investigation 4 *Smooth Bay* parameters for region simplification.

The remaining number of elements and vertices are shown in Table 6.12.

	Region simpl.	Lindstrom-Turk simpl.
elements	1,891	1,846
vertices	974	974

Table 6.12: Investigation 4 *Smooth Bay*, elements and vertices in the mesh after the simplification.

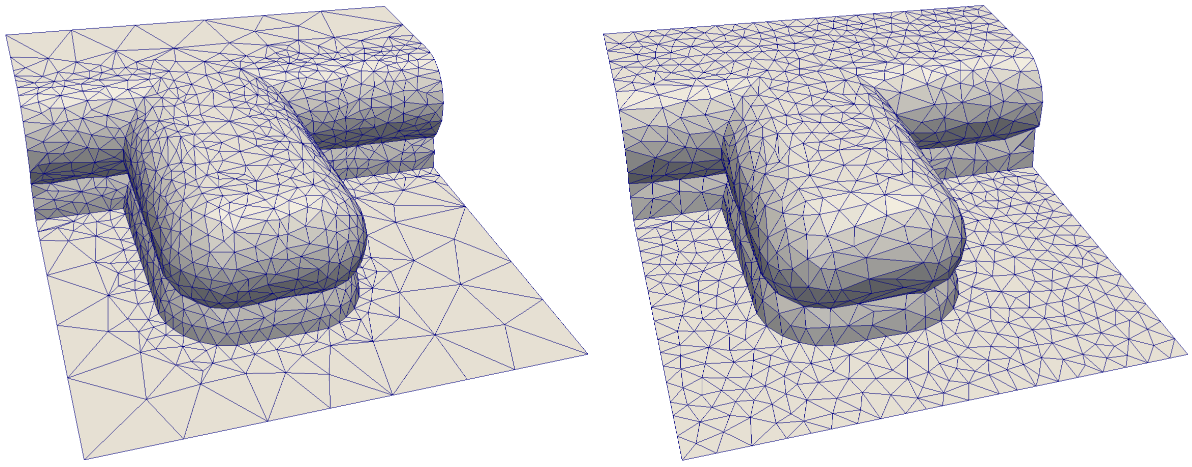
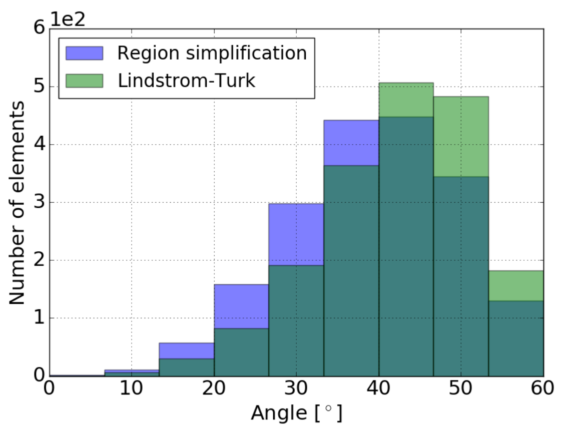
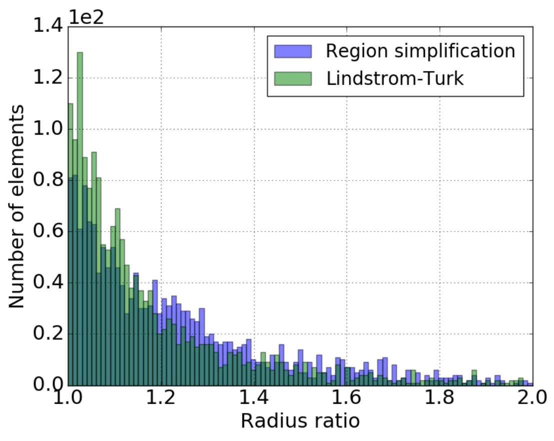

(a) Region simplification (b) Lindstrom-Turk simplification

Figure 6.18: Investigation 4 *Smooth Bay* output.

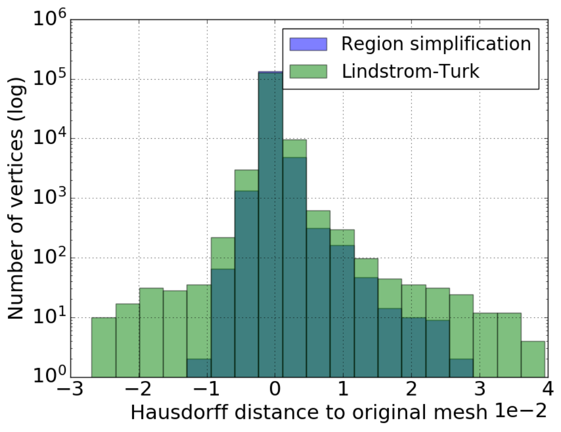
Figure 6.19 shows the results of this investigation. The approach of Lindstrom and Turk achieves a higher element quality (see Figure 6.19(a) and Figure 6.19(b)). The difference in the distance to the original mesh of the two approaches is nearly identical (see Figure 6.19(c)). Figure 6.20 shows that the average difference in calculation time is 7 % in favour of the Lindstrom-Turk simplification process. This investigation shows that if the feature region is vastly simplified *region simplification* cannot preserve the original geometry in greater detail.



(a) Minimum angles of all elements.



(b) Radius ratio of all elements.



(c) Vertex distance to the original mesh.

Figure 6.19: Investigation 4 *Smooth Bay* quality.

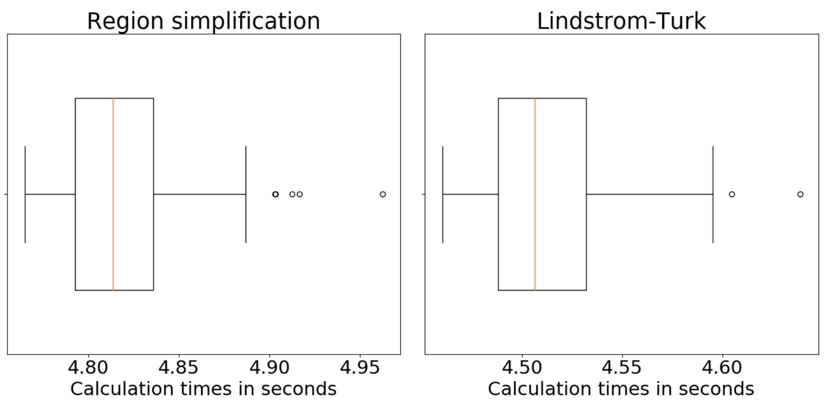


Figure 6.20: Investigation 4 *Smooth Bay* calculation times.

6.1.3 Results Without Transition Region

One of the primary goals of this work was to maintain a high element quality in the mesh after the simplification process. For this purpose the transition region (see Section 5.3.2) was introduced. However, this approach of surface mesh simplification also offers the possibility to ignore element quality in favour of an overall lower number of elements while maintaining geometric features with their original resolution (see Figure 6.21 and Figure 6.22). These meshes were obtained by using *region simplification* with the parameters shown in Table 6.13 and Table 6.15.

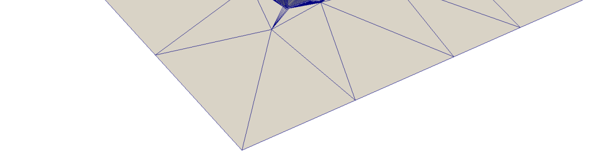
Feature region	Flat region
0.0	0.5

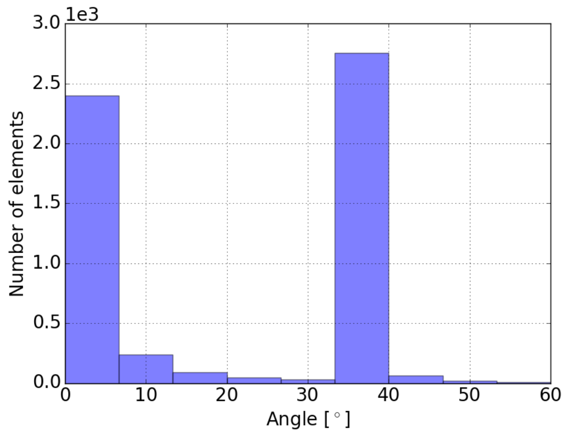
Table 6.13: Parameters for *Square Bay* with no transition region.

The remaining number of elements and vertices are shown in Table 6.14.

	Region simpl.
elements	5,641
vertices	2,840

Table 6.14: No transition region *Square Bay*, elements and vertices in the mesh after the simplification.


(a) Simplified mesh



(b) Element quality

Figure 6.21: Investigation with no transition region using the mesh *Square Bay*.

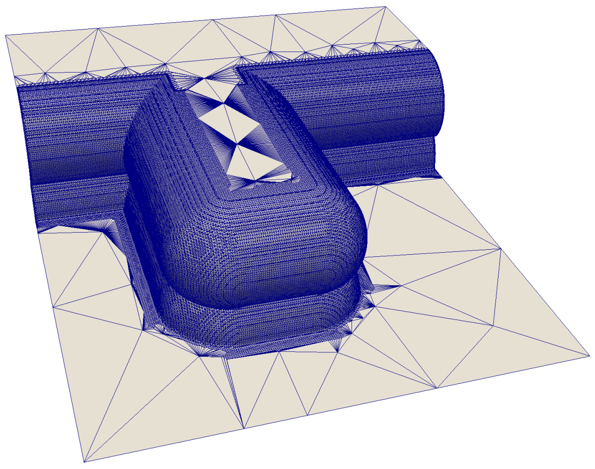
Feature region	Flat region
0.0	0.5

Table 6.15: Parameters for *Smooth Bay* with no transition region.

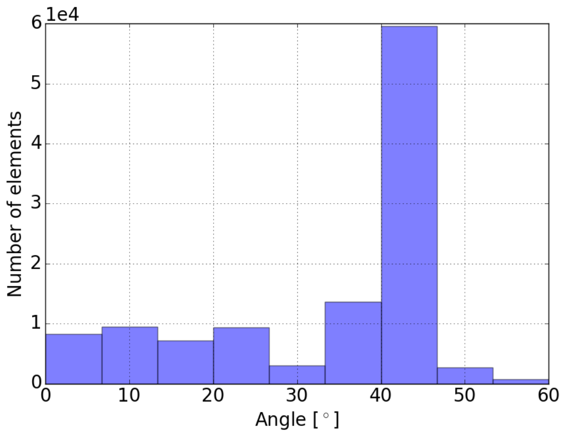
The remaining number of elements and vertices are shown in Table 6.16.

	Region simpl.
elements	113,732
vertices	57,109

Table 6.16: No transition region *Smooth Bay*, elements and vertices in the mesh after the simplification.



(a) Simplified mesh



(b) Element quality

Figure 6.22: Investigation with no transition region using the mesh *Smooth Bay*.

6.1.4 Discussion

Since the mesh *Smooth Bay* has a larger amount of feature vertices the graphs in this section are created with data gained from only this mesh. The feature region in the mesh *Square Bay* is so small, that if this region is simplified much information about the geometry is lost (see Figure 6.5(c)). However, the conclusions drawn about the transition and flat regions of the mesh are also valid for *Square Bay*.

All investigations in the previous section show that *region simplification* preserves geometric features with a higher resolution when the mesh is simplified to the same number of vertices than using the Lindstrom-Turk simplification (see Figure 6.25). Depending on the application the mesh is needed for, the number of elements that need to be removed from the feature region has to be adjusted. This influences the overall geometry of the mesh which is better preserved by using *region simplification*.

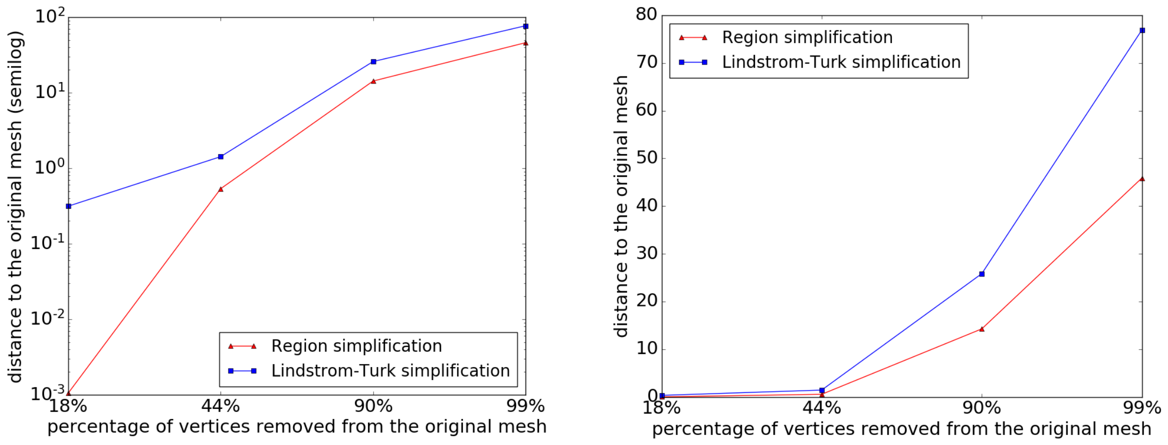
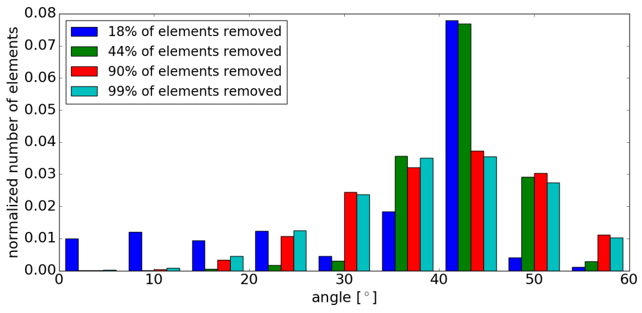


Figure 6.23: Accumulated absolute distance to the original mesh compared to the degree of simplification *Smooth Bay*.

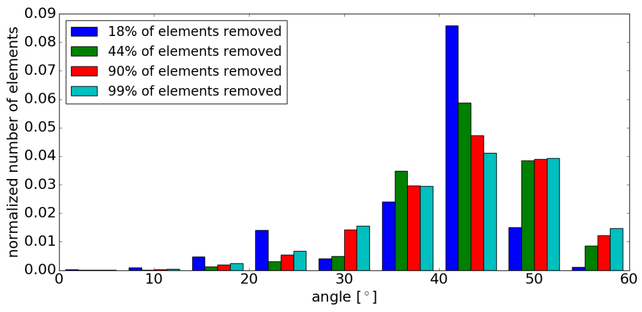
Region simplification creates a transition region between high and low simplification regions. This improves the overall number of elements that can be removed, but this process also has the side effect that some elements have to have a lower quality since they have to connect different regions of the mesh with different edge sizes. Figure 6.24 shows that *region simplification* creates more elements with a minimum angle between 10 and 35 degree than the Lindstrom-Turk simplification. It is also possible that this approach creates a few elements with a very low quality, but they can be fixed in a post processing step further discussed in Section 7.

In terms of calculation time the Lindstrom-Turk simplification process is faster than *region simplification* (see Figure 6.23). However, this was expected since *region simplification* is an extension of the Lindstrom-Turk simplification approach which simplifies certain parts of the mesh with different parameters. This additional computation time stems from the fact that the mesh has to be analyzed and split into different regions. Furthermore, the difference in average computation time was at most two seconds which is not even a difference of one order of magnitude between the computation time of both approaches. The largest differences in calculation time occurs in the investigations that do not simplify the feature region of the mesh.

This is due to the fact that in these investigations only the flat regions of the mesh are simplified, which in turn increases the impact of the feature detection on the overall calculation time.



(a) Region simplification



(b) Lindstrom-Turk simplification

Figure 6.24: Minimum angle of the mesh *Smooth Bay* after the simplification.

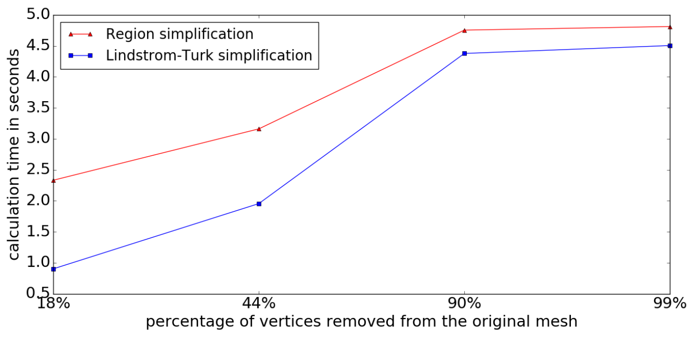


Figure 6.25: *Smooth Bay* calculation time compared to the degree of simplification.

Chapter 7

Summary and Outlook

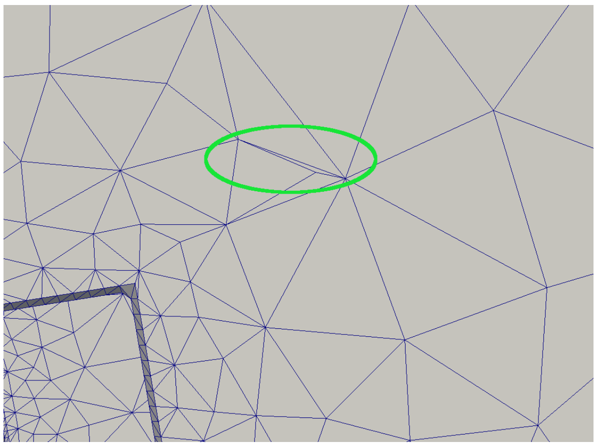
In this work an extension of the Lindstrom and Turk simplification algorithm has been introduced and discussed. This extension categorizes the vertices of a mesh into two regions based on their geometric properties, and simplifies these regions with different strategies to minimize the distance (see Section 2.2.3) to the original geometry. To maintain a high element quality for all mesh elements an additional region was introduced (see Section 5.3.2) that creates a transition between the previously created regions.

Regarding future work: The simplification approach developed in this work is an extension of the Lindstrom and Turk simplification algorithm. However, since this approach allows to apply different simplification strategies to different regions of the mesh, it would be of interest to investigate the effects of other simplification strategies on $\mathcal{M}_{feature}$, for example, the strategies presented in [11] or [34].

At the moment the transition region is calculated trough a breadth-first search. Depth-first search or other search algorithms could be investigated whether they yield improved transition regions.

In some cases the transition region has bad elements (see Figure 7.1). A post processing step could be developed that would fix those by changing the position of one vertex of the element. This process would have a computational complexity of $\mathcal{O}(n)$, were $n \in \mathbb{N}$ is the number of vertices in the mesh $\mathcal{M}$ and would increase the overall runtime of the algorithm by a constant amount of time.

It is possible to detect a third set of vertices, the *significant features* (see Figure 7.2(a)), and incorporate it into the algorithm. After detecting a feature vertex it is tested if the absolute mean curvature or the absolute Gaussian curvature is greater than a set tolerance. If one of these conditions is true the vertex is put into the set of *significant feature vertices*. The computation time of this additional region would only increase the runtime by a constant amount and thus would not effect its computational complexity. This additional region would improve the distance to the original mesh with an overall lower amount of elements in the mesh.


(a) Mesh *Square Bay*

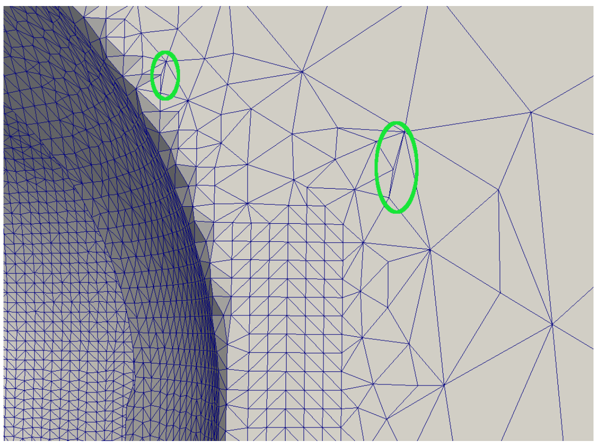
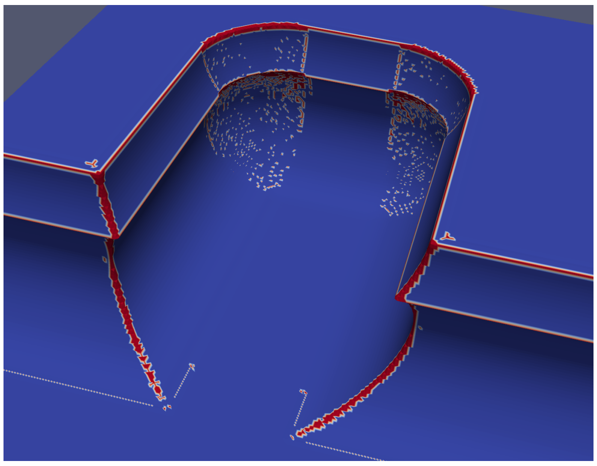

(b) Mesh *Smooth Bay*

Figure 7.1: Examples of bad elements in the transition region.


(a) Tolerance 10.0

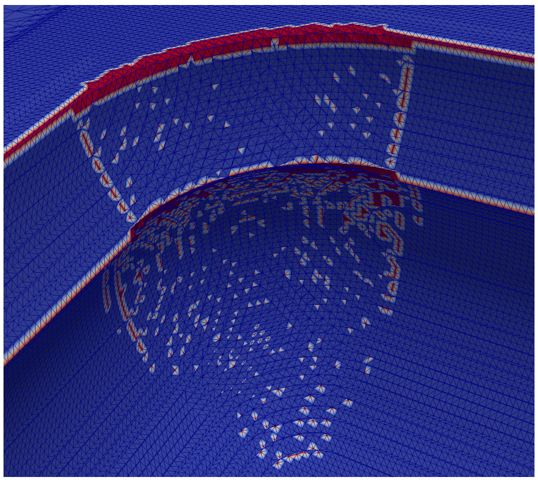

(b) Errors in the detection algorithm.

Figure 7.2: Significant features of *Smooth Bay*.

Bibliography

[1] D. Bommes, L. Bruno, N. Pietroni, E. Puppo, C. Silva, M. Tarini, and D. Zorin. "State of the Art in Quad Meshing." In: *Proceedings of the Annual Conference of the European Association for Computer Graphics (Eurographics)*. 2012.

[2] S. Brenner and R. Scott. *The Mathematical Theory of Finite Element Methods (Texts in Applied Mathematics)*. Springer, 2007. ISBN: 0387759336.

[3] J. W. Bruce and P. J. Giblin. *Curves and Singularities: A Geometrical Introduction to Singularity Theory.* 2nd ed. Cambridge: Cambridge University Press, 1992. ISBN: 9781139172615.

[4] F. Cacciola. "Triangulated Surface Mesh Simplification." In: *CGAL User and Reference Manual.* 4.12.1. CGAL Editorial Board, 2018.

[5] M. P. Do Carmo. *Differential Geometry of Curves and Surfaces.* 2nd ed. New York: Dover Publications, 2016. ISBN: 9780486806990.

[6] F. Cazals and M. Pouget. "Estimating Differential Quantities Using Polynomial Fitting of Osculating Jets." In: *Computer Aided Geometric Design* 22.2 (2005), pp. 121–146. DOI: `10.1016/j.cagd.2004.09.004`.

[7] P. Cignoni, C. Montani, and R. Scopigno. "A Comparison of Mesh Simplification Algorithms." In: *Computers & Graphics* 22.1 (1998), pp. 37–54. DOI: `10.1016/S0097-8493(97)00082-4`.

[8] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr. "Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow." In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH).* 1999, pp. 317–324. DOI: `10.1145/311535.311576`.

[9] U. Dierkes, S. Hildebrandt, A. Küster, and O. Wohlrab. *Minimal Surfaces I.* Springer-Verlag, 1991. ISBN: 3540531696. DOI: `10.1007/978-3-662-02791-2_3`.

[10] I. Farmaga, P. Shmigelskyi, P. Spiewak, and L. Ciupinski. "Evaluation of Computational Complexity of Finite Element Analysis." In: *Proceedings of the 11th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM).* 2011, pp. 213–214.

[11] M. Garland and P. S. Heckbert. "Surface Simplification Using Quadric Error Metrics." In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH).* 1997, pp. 209–216. ISBN: 0897918967. DOI: `10.1145/258734.258849`.

[12] B. Gill and H.-P. Sariel. "Efficiently Approximating the Minimum-Volume Bounding Box of a Point Set in Three Dimensions." In: *Journal of Algorithms* 38 (2001), pp. 91–109. DOI: `10.1006/jagm.2000.1127`.

[13] H. Havlicek. *Lineare Algebra für Technische Mathematiker.* Heldermann, 2008. ISBN: 9783885381167.

[14] L. Hogben. *Handbook of linear algebra.* CRC Press, Taylor & Francis Group, 2016. ISBN: 9781138199897. DOI: `10.1201/b16113`.

[15] L. Kettner. "3D Polyhedral Surface." In: *CGAL User and Reference Manual.* 4.12.1. CGAL Editorial Board, 2018.

[16] S. -J. Kim, C. -H. Kim, and D. Levin. "Surface Simplification Using a Discrete Curvature Norm." In: *Computers & Graphics* 26.5 (2002), pp. 657–663. DOI: `10.1016/S0097-8493(02)00121-8`.

[17] P. Lindstrom and G. Turk. "Fast and Memory Efficient Polygonal Simplification." In: *Proceedings of the Conference on Visualization.* IEEE Computer Society Press, 1998, pp. 279–286. DOI: `10.1109/VISUAL.1998.745314`.

[18] S. H. Lo. "A New Mesh Generation Scheme For Arbitrary Planar Domains." In: *International Journal for Numerical Methods in Engineering* 21.8 (1985), pp. 1403–1426. DOI: `10.1002/nme.1620210805`.

[19] W. E. Lorensen and H. E. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm." In: *Computer Graphics* 21.4 (1987), pp. 163–169. DOI: `10.1145/37402.37422`.

[20] P. L. Manstetten. "Efficient Flux Calculations for Topography Simulation." Doctoral Dissertation. TU Wien, 2018.

[21] P. Manstetten, J. Weinbub, A. Hössinger, and S. Selberherr. "Using Temporary Explicit Meshes for Direct Flux Calculation on Implicit Surfaces." In: *Procedia Computer Science* 108 (2017), pp. 245–254. DOI: `https://doi.org/10.1016/j.procs.2017.05.067`.

[22] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr. "Discrete Differential-Geometry Operators for Triangulated 2-Manifolds." In: *Visualization and Mathematics III.* Ed. by Hans-Christian Hege and Konrad Polthier. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 35–57. ISBN: 9783662051054. DOI: `10.1007/978-3-662-05105-4_2`.

[23] M. Pharr and G. Humphreys. *Physically Based Rendering: From Theory to Implementation.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004. ISBN: 9780128007099.

[24] P. P. Philippe and J. B. Timothy. "Analysis of Triangle Quality Measures." In: *Mathematics of Computation 72.* 2003, pp. 1817–1839. DOI: `10.1090/S0025-5718-03-01485-6`.

[25] K. Polthier and M. Schmies. "Straightest Geodesics on Polyhedral Surfaces." In: *Proceedings of the International Conference and Exhibition on Computer Graphics and Interactive Techniques (SIGGRAPH) Courses.* 2006, pp. 30–38. ISBN: 1595933646. DOI: `10.1145/1185657.1185664`.

[26] M. Pouget and F. Cazals. "Estimation of Local Differential Properties of Point-Sampled Surfaces." In: *CGAL User and Reference Manual.* 4.12.1. CGAL Editorial Board, 2018.

[27] K. J. Renze and J. H. Oliver. "Generalized Surface and Volume Decimation for Unstructured Tessellated Domains." In: *Proceedings of the IEEE Virtual Reality Annual International Symposium (VR).* 1996, pp. 111–121. ISBN: 0818672951. DOI: `10.1109/VRAIS.1996.490518`.

[28] F. Rudolf. "Symmetry- and Similarity-Aware Volumetric Meshing." Doctoral Dissertation. TU Wien, 2016.

[29] F. Rudolf, J. Weinbub, K. Rupp, and S. Selberherr. "The Meshing Framework ViennaMesh for Finite Element Applications." In: *Journal of Computational and Applied Mathematics* 270 (2014), pp. 166–177. DOI: `10.1016/j.cam.2014.02.005`.

[30] P. V. Satheesh. *Unreal Engine 4 Game Development Essentials : Master the Basics of Unreal Engine 4 to Build Stunning Video Games.* Packt Publishing, 2016. ISBN: 1784398454.

[31] J. A. Sethian. *Level Set Methods and Fast Marching Methods.* Cambridge Univ. Press, 2007. ISBN: 9780521645577.

[32] J. A. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science.* Cambridge University Press, 1999. ISBN: 9780521642040.

[33] J. R. Shewchuk. "What is a Good Linear Element? Interpolation, Conditioning, and Quality Measures." In: *Proceedings of the International Meshing Roundtable (IMR), pages 115-126, 2002.* 2002.

[34] Z. Shi, K. Qian, K. Yu, and X. Luo. "A New Triangle Mesh Simplification Method with Sharp Feature." In: *Proceedings of International Conference on Digital Home (ICDH).* 2018, pp. 324–328. DOI: 10.1109/ICDH.2018.00063.

[35] R. Straub. "Exact Computation of the Hausdorff Distance between Triangular Meshes." In: *Proceedings Eurographics Short Papers.* 2007. DOI: 10.2312/egs.20071023.

[36] The CGAL Project. *CGAL User and Reference Manual.* 4.12.1. CGAL Editorial Board, 2018.

[37] *ViennaMesh.* 2016. URL: http://viennamesh.sourceforge.net/.

[38] W. Schroeder. *The Visualization Toolkit.* 4th ed. Kitware, 2006. ISBN: 9781930934191.

[39] K. Weiler. "Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments." In: *IEEE Computer Graphics and Applications* 5.1 (1985), pp. 21–40. DOI: 10.1109/mcg.1985.276271.

[40] Y. -C. Wu and Y. -R. Jhan. *3D TCAD Simulation for CMOS Nanoeletronic Devices.* Springer Singapore, 2018. ISBN: 9811030669.