Technischen Universität München
Institut für Informatik
Lehrstuhl für Computer Graphik und Visualisierung

Growing Surface Structures
an Iterative Refinement Surface Reconstruction Approach
Deduced from Artificial Neural Networks

Hendrik Annuth

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universiät München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzende: Univ.-Prof. Gudrun Johanna Klinker
Prüfer der Dissertation:
1. Univ.-Prof. Dr. Rüdiger Westermann
2. Univ.-Prof. Dr. Heinrich Müller, Technische Universität Dortmund
3. Prof. Dr. Christian-Arved Bohn, Fachhochschule Wedel

Die Dissertation wurde am 17.02.2014 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 22.05.2014 angenommen.

Abstract

This thesis presents a high quality surface reconstruction approach based on an iterative refinement strategy deduced from artificial neural networks. The presented approach is composed of three novel developments extending the capabilities as well as the concept of the growing cell structures algorithm.

An introduction to the associated fields of research is given and the problem of surface reconstruction from unorganized points is discussed in detail. A state of the art overview of all major surface reconstruction methods and their performance is presented. A detailed introduction to the development stages of the growing cell structures algorithm is given, which enables localizing algorithm properties for possible modification.

The distinctions between the growing cell structures and the growing neural gas algorithm are discussed.

The presented filter chain concept enables easy editability of the algorithm behavior and adds additional flexibility. The twist solving method efficiently and reliably fixes inconsistently oriented surface areas within the reconstruction process. The potential of the novel data structures used to solve this problem for other computer graphics applications is demonstrated. The growing cell structures algorithm concept is redesigned to incorporate an explicit surface representation into its learning scheme. This introduces novel possibilities to direct the algorithm's approximation behavior.

The final algorithm, composed of the combined presented developments, is evaluated in comparison to classical reconstruction algorithms as well as in reconstructing extremely challenging point clouds. The algorithm proves to be an efficient all-round high quality approach. Finally, the presented results and their implications for future work are discussed.

Kurzfassung

In dieser Dissertation wird ein qualitativ hochwertiges Oberflächenrekonstruktionsverfahren vorgestellt, welches auf einer iterativen Verfeinerungsstrategie basiert, die von künstlichen neuronalen Netzen abgeleitet ist. Der vorgestellte Ansatz setzt sich aus drei neuen Entwicklungen zusammen, welche sowohl die Leistungsfähigkeit als auch die Konzeption des Growing Cell Structures Algorithmus erweitern.

Die zugehörigen Forschungsfelder der Arbeit werden vorgestellt und das Problem der Oberflächenrekonstruktion von unorganisierten Punkten wird im Detail dargestellt. Alle wesentlichen Oberflächenrekonstruktionsmethoden werden in einem aktuellen Überblick gemeinsam mit deren Eigenschaften präsentiert. Durch die detaillierte Vorstellung der Entwicklungsstadien des Growing Cell Structures Algorithmus wird die Lokalisierung von Algorithmuseigenschaften für potenzielle Modifikationen ermöglicht.

Die Unterschiede von Growing Cell Structures zu Growing Neural Gas werden diskutiert.

Das präsentierte Filterkettenkonzept ermöglicht die einfache Editierbarkeit des Algorithmus und macht diesen zusätzlich flexibler. Die Verdrehungsauflösungsmethode korrigiert inkonsistente Oberflächenorientierungen innerhalb des Rekonstruktionsprozesses auf effiziente und zuverlässige Weise. Das Potenzial innerhalb der zur Lösung verwendeten Datenstrukturen wird demonstriert indem diese zusätzlich für andere Computergraphik-Anwendungen eingesetzt werden. Durch eine Neukonzeption des Growing Cell Structures Algorithmus wird eine explizite Flächenrepräsentation in dessen Lernschema mit einbezogen. Dies eröffnet neue Möglichkeiten bei der Steuerung des Approximationsverhaltens des Algorithmus.

Der finale Algorithmus, welcher sich aus den vorgestellten Entwicklungen zusammensetzt, wird durch Vergleiche mit klassischen Rekonstruktionsalgorithmen evaluiert und durch die Rekonstruktion von extrem anspruchsvollen Punktwolken. Der Algorithmus erweist sich dabei als ein effizienter und qualitativ hochwertiger Ansatz, der sich für viele Einsatzfelder eignet. Die Arbeit wird durch eine Diskussion über die erzielten Ergebnisse und deren Bedeutung für die zukünftige Forschung abgeschlossen.

Contents

Chapter 1

Introduction

In this chapter, an introduction to *surface reconstruction* and *artificial neural networks* (ANN) – the associated fields of research of this thesis – is given. The structure of the thesis is explained and its contribution is presented.

1.1 Surface Reconstruction

Due to the rapid development in 3D scanning technology, real world objects can be scanned faster, more accurately and at higher resolutions. State of the art laser scanning devices are able to acquire a hundred million points with one single scan. This allows creating high quality virtual representations of these objects as shown in Fig. 1.1.

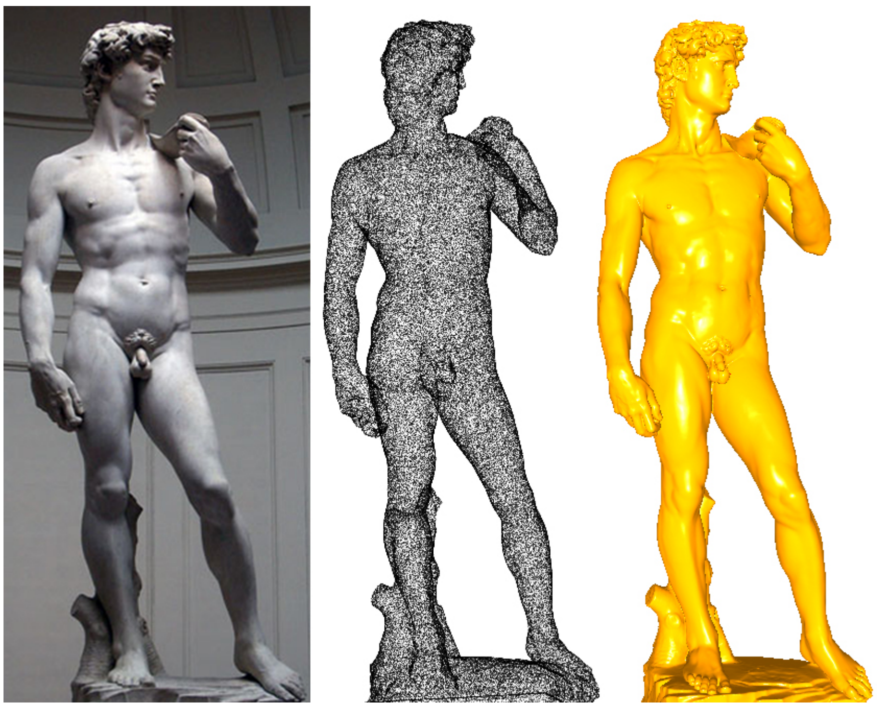


Figure 1.1: A photograph of *Michelangelo's David* (left); A point cloud of the David statue (middle); A surface fitted into that point cloud (right).

Application Cases: The digitalization of physical surfaces has many different applications. In archaeology and crime scene investigation, sites can be analyzed independent of their location and permanently preserved through time.

Architecture and plant manufacturing involve accurate construction planning and precise measuring which can be efficiently accomplished by introducing digital surface models. In medical applications, memory intensive volume models or contour data from stacks of pictures are reconstructed for visualization purposes. In films, reconstructed surfaces are applied as support structures for manually created models. Computer games use 3D scanning technology to incorporate real world objects and environments into their otherwise virtual worlds.
Reverse engineering is a process where existing products or physical prototypes are used as a template to create virtual models for production processes. These models are then used for *computer-aided design* (CAD) and in *computer-aided manufacturing* (CAM). The reconstruction is usually only the first step in a process where the surface is additionally segmented, elements are classified, a parametric surface is fitted and additional constraints are recognized and added to the model description. The complete process is called *digital shape reconstruction* [VF05]. This application case is currently gaining additional attention due to the growing popularity of 3D printing devices.
In quality control – often seen as part of reverse engineering – the digital representation of a physically created object is used to verify the accuracy of a production process. This is especially important for complex shaped objects, for which conventional measuring techniques are impractical.
Mobile robots can profit from surface reconstruction in different contexts, such as path planning, localization, scene interpretation, and grasping. Robotic maps are the basis for all actions of a mobile robot.
Scanning Devices: The applied scanning devices are as numerous as the described application cases. They differ in their accuracy, the complexity of their scanning process, their limitations due to their physical working principle and the reconstruction relevant information in their data output.
By using a touch probe, points, or contours on a physical surface are determined mechanically, which is very accurate, but leads to an extremely slow scanning process. Medical applications usually need to scan the inside of a body rather than its surface. Scanning techniques such as *computed tomography* (CT), *magnetic resonance imaging* (MRI), and *ultrasound imaging* are used. These techniques often produce stacks of pictures that represent cross-sections of the body, which lead to *contour data* as input data for the reconstruction process.
Multi-view stereo (MVS) methods use a collection of images as input data for the reconstruction process. There are different techniques to create 3D information from those 2D images. *Binocular, trinocular*, and *multi-baseline methods*, also called *single shot methods*, seek to create a single *depth map* by identifying corresponding *feature points* on two or three images [SS02].
However, MVS is mostly associated with methods that use a higher number of images taken from different points of view to create 3D models of greater complexity [SCD$^+$06]. They use *photo-consistency measure* to find correlating pixels in the images. Usually they also include a visibility model that retrieves 3D information based on the visibility or the occlusion of certain features in the images.
Structured light measurement devices project a calibrated reference pattern into the scene. Naturally, the surface of the present object distorts this pattern. Based on the difference between referenced and sensed pattern, the 3D coordinates of samples on the object surfaces are determined. These methods can achieve very high resolutions and accuracy. The scanning process normally produces *range images*, which already represent oriented partial surface segments.
A technique which also uses the projection of a structured light pattern, typically a grit, is *time of flight* (TOF). TOF has a very high scanning frequency at the expense of scanning accuracy. Thus, it is used in many real-time applications.

The first prominent use of such systems in practical applications was the *Stanford Digital Michelangelo Project* [Lev99]. While these systems deliver point clouds of high quality, they need a very controlled setup and have a limited sensing area. Thus, they are normally used to capture single objects rather than large scenes. As these devices depend on sensing the reference pattern, they are sensitive to additional light sources that outshine the pattern. Hence, they are mostly used in indoor environments.

The most versatile applicable scanning devices are *3D laser scanners*, which use a laser beam that samples a surface point by point. The device determines the distance of the scanner to a surface point by measuring the time it takes the light to be reflected from the surface to the scanner. Together with the current beam direction, this defines a 3D position. To produce a cloud of points, a mechanical system using rotating mirrors directs the laser beam around two axes to create a spherical sampling.

In most real world applications, one single scan is insufficient to capture an entire scene. Normally, several scans are performed and united into a single point cloud. There are various techniques for integrating point clouds of different scans into a global coordinate system. It can either be done manually, which is quite a time-consuming task or geo-reference markers such as spheres can be placed in the scene. These markers are then detected in the different point clouds and enable an automatic or semi-automatic alignment process.

The development of 3D laser scanners has shown a tremendous improvement over the recent years. A 3D laser scanner combines a high accuracy with a time efficient and straightforward scanning process. It has a large scanning range and its physical scanning principle exposes only few limitations such as shiny object surfaces, which reflect the emitted signal, or dark surfaces, which absorb the laser light, and thus create low or even not sampled surfaces. The sampling density is generally non-uniform. Due to the consistent spherical sampling, objects that are closer to the scanner are captured with a higher density than objects that are far away. Additionally, when integrating several scans, interlaced areas have higher sample densities than non-interlaced areas.

Unorganized Point Clouds: The result of a 3D laser scan is an *unorganized point cloud*. Only spatial point positions are known and no additional information is given, such as relations between points or the orientation of a point. Since unorganized point clouds carry a minimum of information, data from most scanning devices can be transformed easily into this format, but not vice versa. Algorithms working with unorganized points are therefore usable quite universally. However, not using all available information might compromise the quality of the produced result.

Surface Reconstruction Motivation: An unorganized point cloud already has many applications even before it is transformed into a surface. When scanning a physical surface, the representation is digital and can therefore very easily be copied, transferred and permanently retained. Point clouds can also be used directly in point-based computer graphics applications [KB04].

A point cloud, however, does not represent a coherently defined *subspace* in 3D space as a surface does, which limits its application. Surface models enable very sophisticated measuring processes, such as distances based on intersection and projection operations, distances on the surface or volume calculations.

In visualization, lighting calculations involve entry and exit angles of light beams reflected from surfaces. This makes the gradient of a surface necessary. Using common texturing techniques initially requires the parameterization of a surface. Closed surfaces which do not self-intersect can be transferred to volumetric models, such as tetrahedron meshes [TWAD09]. Such models can be exposed to simulated forces such as tension and pressure.

Although a surface model is a more complete representation of a physical surface than a point cloud, it is generally more memory efficient. Complex shaped surface areas, such as thin and curved ones, need more samples to be represented adequately. Since scanning processes cannot be adapted in their resolution, most other surface areas are overrepresented by many redundant samples.
An unorganized point cloud may include noise, outliers, non-uniform sample densities and insufficiently sampled structures and holes. Since many of these problems are inherently ambiguous and mutually dependent, surface reconstruction approaches as well as pre- and post-processes suggested to the problem are numerous. Although the problem has great importance in different application cases and has been dealt with intensively since the early eighties [Boi84], a generally satisfactory solution to the process has not yet been found.

1.2 Artificial Neural Networks

The beginning of *artificial intelligence* (AI) research was characterized by great optimism. In 1967 Marvin Minsky, cofounder of the term itself, was quoted to have said, "within a generation ... the problem of creating 'artificial intelligence' will be substantially solved." ([JHB95], p. 109).
At that time AI research was directly associated with the attempt to rebuild or even exceed human intelligence, which is today an AI subdiscipline termed "strong AI". The strong AI problem was basically approached from two directions.
One was to examine the results of human intelligence and to reproduce those results within a process. This *top-down* approach led to so called "expert systems". A second approach was to recreate the building blocks of natural intelligent systems. Here, AI is created due to emergence, where many simple units create complex behavior as a result of their interactions.
The inspiration for this *bottom-up* approach came from the increasing understanding of biological neural networks. Which was expressed, for instance, in the *Hebbian theory* postulated by Donald Hebb in 1949 [Heb49]. Biological neural networks are built of relatively simple base units, the neurons, that can only emit and receive simple signals, but if connected in networks and in vast numbers, are able to create something as complex as human behavior. In 1951, Marvin Minsky created the first ANN at Princeton University.
With passing time, the possibility to actually create a strong AI became increasingly unlikely. In 1982, Marvin Minsky wrote, "The AI problem is one of the hardest science has ever undertaken." ([Kol82], p. 1237). However, ANNs had proven to be a useful tool.
Statistical Learning Theory: In computation, ANNs are used due to their "learning capability". In *statistical learning theory*, learning describes the ability to build a predictive function derived from given examples. Common tasks are *function approximation*, *regression analysis*, *classification*, *pattern recognition*, *dimensionality reduction*, *clustering*, and *feature extraction*.
Input Data: ANNs typically use *Euclidian distance* as a metric for data comparison. Thus, in order to apply them to a problem at hand, the given *input data* (examples) has to be represented as numerical vector data. The input vectors represent a *sampling* of the function of interest in the *input data space*. Since ANNs are known to be very stable and robust they are often used for tasks which are characterized by complex, blurred and incomplete input data.
Output: ANNs can be distinguished by the model they use for representing the created predictive function. One is to divide the input data space by a series of *hyperplanes* using the scalar product. In a *perceptron* [MP88], for instance, input data is multiplied by a series of *weight vectors*. This associates an input sample (vector) with a certain fraction of space, which is also known as a *subspace*. Such subspaces are again associated with one of several predefined classes in classification applications.

When using hyperplanes, processes such as perceptrons model subspaces by defining the boundaries between them. The number of hyperplanes which an input sample passes, defines the complexity in which the input data space can be subdivided.
A second concept is to directly model subspaces by placing *reference vectors* into the input data space. Here, subspaces are defined by single reference vectors or groups of them. An input sample is then associated with the reference vector with the smallest Euclidian distance. The complexity of created subspaces is determined by the number of reference vectors used to shape them.
Supervised Learning: There are different learning concepts available, depending on the given input data and the task to fulfill. *Supervised learning* is strongly connected to the task of classification. Here, every input sample comes with an associated output.
In a *training phase*, input samples are presented to the algorithm which then predicts an output for them. Since the actual output values are known, a feedback can be created to "supervise" the learning process. This feedback is represented by a *distance metric*, which expresses how correct or incorrect the predicted output for a given input sample is. This distance metric could, for instance, be the square distance between predicted and actual output.
The training phase is finished when a certain threshold for the error in these predictions is reached. The process can then be used to make predictions for input samples of unknown output.
Unsupervised Learning: In *unsupervised learning*, input samples come without associated output values and the learning process therefore has to be "unsupervised". Such processes find hidden structures in the input data to allow for predictive capabilities. A typical task is clustering. Clustering is also referred to as "automatic classification", since it is equivalent to classification, but without predefined classes.
While in supervised learning correctly determined classes or outputs are of interest, unsupervised learning focuses on the differentiation and determination of significant aspects within the input data and their visualization. This includes tasks such as pattern recognition, dimensionality reduction, and feature extraction.
Competitive Learning: *Competitive learning* is characterized by the "competition" of multiple units for presented input samples. If a unit in this competition "wins" for a given input sample, it is modified. The modification increases the likelihood for this unit to win again for the given and for alike input samples. Gradually, these modifications structure the units to represent *clusters* of input samples. Thus, the approach represents an unsupervised learning technique.
There are hyperplane based implementations modifying weight vectors to segment the input space as well as distance based approaches, where reference vectors compete and represent clusters.
A distance based approach is categorized as *hard competitive learning* if only one reference vector (unit) is modified when an input sample or several input samples are presented. This learning scheme is also known as *winner-take-all.* The *k-means clustering* [Mac67] and the *Linde-Buzo-Gray* (LBG) [LBG80] approach use this learning scheme. In clustering, clusters of input samples are represented by a single reference vector. In vector quantization, data compression is achieved by representing many input vectors as one single *prototype vector.*
In comparison to hard competitive learning, reference vectors in *soft competitive learning* are connected to one another creating a graph. Not only the winning reference vector is modified to better match an input sample, but also connected neighbors in the graph. The learning scheme is also known as *winner-take-most.* Since reference vectors are not moved independently, their movements are stabilized by one another, creating smoother distributions. Depending on the restrictions of a graph's connectivity, additional applications are enabled.
A graph can be used to additionally recognize the underlying continuous aspects of the recreated subspace of the input data, as in regression analysis. *Neural gas* (NG) [MS91, MS94] and

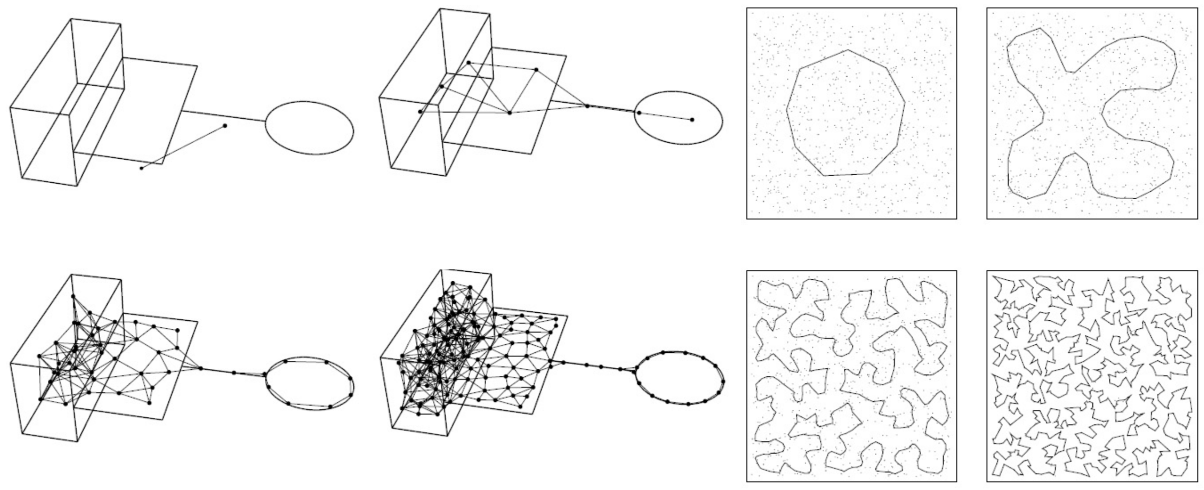


Figure 1.2: Approximation series of a subspace of multiple dimensions with GNG (left) (from [Fri95]). Series of approximate results for the *travelling salesman problem* with GCS (right) (from [FW91]).

growing neural gas (GNG), for instance, produce graphs of arbitrary connectivity. If a subspace is modeled with such a graph, its connectivity can resemble structures of arbitrary dimension. If a subspace has the characteristics of a curve (1D), it can connect reference vectors as line segments, it can connect them to triangles in case of a surface (2D), tetrahedrons in case of a volume (3D) and even to simplices of higher dimension (see Fig. 1.2). However, if only one particular dimension is desired its sole construction cannot be guaranteed.
The *self-organizing map* (SOM) [Koh82] uses a static connectivity that remains unchanged throughout the process. Normally, a square shaped 2D grid is used, since it is easy to parameterize by width and height. The grid is adapted to resemble the input data subspace. The input samples, which might originate from high dimensional space, can then be projected onto the two dimensional *grid-space*. The input data is then easier to visualize or analyze as in dimensionality reduction and pattern recognition.
In *growing cell structures* (GCS) [Fri93], the connectivity of the graph is not static. However, the graph is restricted to only include structures of a predefined dimension. In [FW91], GCS was applied to approximate a result for the *travelling salesman problem* and was restricted to solely include line segments (see Fig. 1.2).
Which soft competitive learning approach is selected for a certain task depends on the expectations toward the resulting graph. If used for surface reconstruction, *e.g.*, a graph needs to expose the structure of a 2D surface.

1.3 Structure

In the following section, a contents overview is given and general information on shown illustrations, algorithm representations, complexity analyses, used point clouds, measured values, and the used test hardware is presented.

1.3.1 Overview

In chapter 1, an introduction to the fields of research associated with this thesis is given, an overview of contents and general information on the structure of the thesis is provided, and finally, its contribution is presented.

Chapter 2 presents the main problem addressed in this thesis. The specific surface reconstruction scenario is defined, the problems based on the input data – unorganized points – and aspects concerning the output – the surface model – are analyzed and a state of the art overview of all major reconstruction methods is given.
When presenting the GCS algorithm in chapter 3, different development stages of the algorithm toward a surface reconstruction approach are shown, novel computer graphics uses of the algorithm additional to reconstruction are demonstrated, the GCS algorithm is distinguished from the GNG approach, and different parameter settings are tested for both algorithms.
Chapter 4 introduces the novel filter chain concept to the GCS algorithm. The filters – used to generalize the way algorithm behavior can be edited – are presented alongside the concept.
If a surface is reconstructed with the GCS approach it might expose inconsistently oriented surface areas. A method to resolve such "twists" in the surface, which is based on a novel set of data structures, is presented in chapter 5.
Chapter 6 introduces *growing surface structures* (GSS), a conceptual redesign incorporating an explicit surface representation into the learning scheme of the GCS approach. An implementation of this new algorithm concept is presented and evaluated.
In chapter 7, all presented algorithm developments are evaluated in a final combined reconstruction algorithm. This algorithm is evaluated in comparison to classical reconstruction algorithms as well as in reconstructing a series of extremely challenging point clouds.
In chapter 8, the findings of this thesis are summarized and discussed, the challenges of future work are considered, and finally, the thesis is concluded by an epilog.

1.3.2 Figures

Illustrations: The algorithm presented in this thesis uses a mesh based surface representation (see section 3.3.4). In illustrations, meshes are presented in different render modes and schematic representations. An overview of these types is illustrated in Fig. 1.3.
Photographs: The photograph of "Michelangelo's David" in Fig. 1.1 is distributed under the *GNU Free Documentation License* [Fre14], the photograph of the "Farm Building in Ethiopia" in Fig. 7.5 was provided by the Hamburg HafenCity University, the photograph of the "Kornhaus Bridge" in Fig. 7.6 was given into public domain by the author, and the photograph of the "Office" of the laser scanning division of the criminal investigation department in Hamburg in Fig. 7.7 was provided by that same department.

1.3.3 Algorithms

Algorithms are presented as a schematic set of instructions. Such schematic representations might include minimized images of actual illustrations of algorithm aspects to further improve orientation and to underline specific algorithm parts. The representation also highlights if an algorithm is based on a previously presented algorithm and which of its instructions are added or removed from the previous algorithm. This presentation scheme is exemplarily outlined in Alg. 1.

1.3.4 Complexity Analysis

All upcoming discussions concerning the complexity of presented algorithms are done under certain realistic use case assumptions if not explicitly stated otherwise.

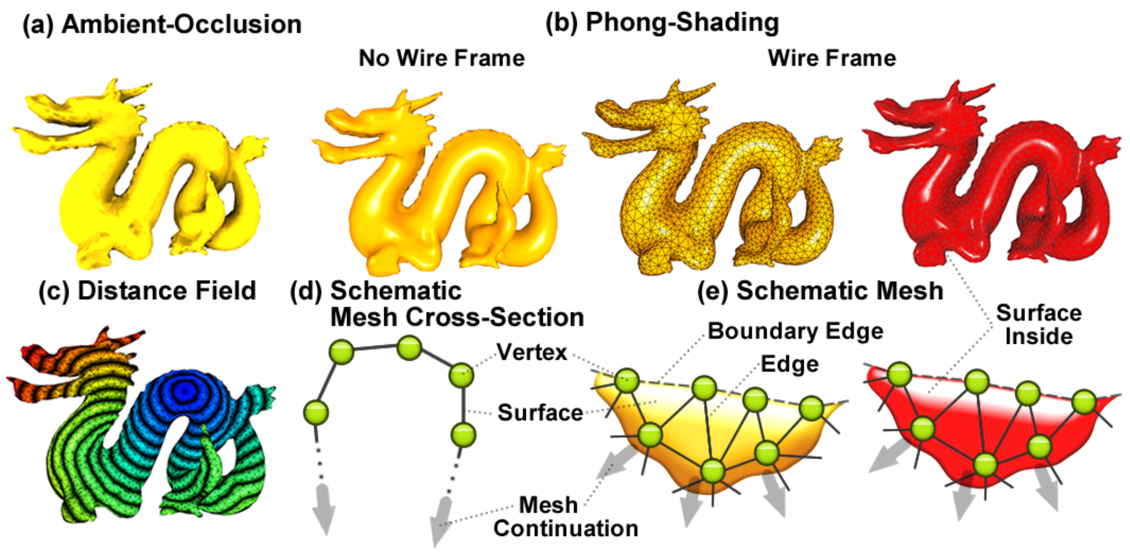


Figure 1.3: An illustration of different mesh representations: *(a)* Ambient-Occlusion is used to emphasize surface depth; *(b)* Phong-Shading is used as the default representation. A wire frame is used to show triangle shapes; *(c)* Distance fields are represented with a color gradient from a starting point (blue) over an intermediate state (green) to the farthest distant points (red). Superimposed on the gradient are contours (black) representing a coherent space of equal distance; To simplify the presentation of certain mesh aspects they are shown as *(d)* a schematic mesh cross section. To precisely demonstrate mesh modifications *(e)* a schematic mesh topview is used. The representations *(b)* and *(e)* both use a color code to differentiate mesh outside (yellow) and inside (red).

This concerns the distribution of input samples, which are generally assumed to represent a surface sampling. Such distributions might include severe imperfections, such as non-uniform sample distributions, noise, and outliers, but exclude cases where distributions actually sample a volume instead of a surface or the majority of all samples in a distribution are compressed in a small spatial region or other equally unfavorable but artificially creatable distributions. To assume such scenarios for real laser scanned data, even under worst-case conditions, is considered unreasonable.
When complexity statements involving mesh-editing operations, the number of edges for vertices is assumed constant and unrelated to the overall mesh size. With edge adding and removing operations applied constantly and evenly on the mesh – given a normal sample distribution as explained above – it is unreasonable to assume a vertex might accumulate a number of connections that is related to the mesh size. A theoretical analysis to mathematically evaluate the

Algorithm 1 This algorithm example is given to explain the different elements used in this thesis to present algorithm concepts.

1: This is a new algorithm instruction which is explained in the corresponding algorithm section.
2: This algorithm instruction is unchanged and has been explained in an early algorithm stage or is part of an entirely new algorithm, where almost all instructions are new.
~~This algorithm instruction was performed in an early algorithm stage. It is exchanged for a new instruction or not needed any longer.~~

A specific algorithm part

3: Instruction of this specific algorithm part
(Specific algorithm parts are additionally represented by visual identifiers, in this example, a blue cogwheel. Their meaning is explained in the corresponding section.)

4: **An already presented specific algorithm part**

unlikelihood of this aspect, given the number and configurability of the presented algorithms would, however, exceed the scope of this thesis.
Thus, complexity statements refer to sensible surface reconstruction cases with appropriate sample distributions and reasonable assumptions about the mesh construction process.

1.3.5 Measurements

The following reappearing measurements are taken. Measurements not shown in this enumeration are explained in the corresponding section.
Time: The time a process took. Normally, it is itemized in "hours : minutes : seconds".
Dist and Dist^2: The average distance or average square distance between the created surface and the samples in the point cloud. For better comparability all point clouds are normalized so that the diagonal of their bounding box equals one. To be in a convenient scope, results for average distance have been multiplied by 10^4 and results for average square distance by 10^7.
Equilaterality: The average percentage of equilaterality for triangles. To calculate the equilaterality of a triangle, first its actual surface area is calculated. Then this surface area is divided by the surface area of a hypothetical triangle. This hypothetical triangle is an equilateral triangle built with the longest edge of the initial triangle. If triangles are equilateral this division results in 100% equilaterality and in smaller values otherwise.
Valence[5;6;7]: The average percentage of vertices having a valence of five, six, or seven. The valence of a vertex is determined by the number of connections it has to other vertices. The ideal value is six, since it is best to shape equilateral triangles, allowing for six divisions with an angle of 60°.

1.3.6 Point Clouds

The point clouds used in this thesis were kindly provided by the Stanford University Computer Graphics Laboratory, the Hamburg HafenCity University, and the laser scanning division of the criminal investigation department in Hamburg.

1.3.7 Test Hardware

In this thesis, different novel algorithm developments for the GCS algorithm are presented. The properties of major developments are constantly proven in practical experiments. For the committed experiments the following hardware was used:

A *Dell*®Precision M6400 Notebook with *Intel*®Core 2 Extreme Quad Core QX9300 (2.53GHz, 1066MHz, 12MB) processor with 8GB 1066 MHz DDR3 Dual Channel RAM

1.4 Contribution

All publications cited in this section were published as part of the scientific work of this thesis. No texts or pictures in this thesis were copied from prior scientific work, neither were contents copied from collaborative, *i.e.*, not *first author* publications. In the following, the scientific contributions produced as part of the thesis are presented.

In this thesis, a detailed analysis of surface reconstruction from unorganized points is presented. A state of the art overview of all major surface reconstruction methods and their performances on these problems is given. This enables a more differentiated perspective on the correlation between a reconstruction method and its performance. The presented overview allows for a novel perspective on reconstruction methods and the strengths and weaknesses they lead to.

The GCS algorithm is presented as a surface reconstruction approach including all its development stages prior to this thesis, such as the *smart growing cells* (SGC) approach, which is summarized in [AB12c]. The differences as well as the similarities of the GCS and the GNG approach are analyzed and experimentally proven [AB12a].

The versatility of the GCS concept is demonstrated by showing novel additional uses for computer graphics applications, such as point cloud filters and mesh-processing algorithms. When combined, those applications can be used to visualize complex virtual scenes [AB11] with efficient rendering techniques, such as *normal mapping*.

The filter chain concept generalizes the way algorithm behavior can be edited in order to account for individual application cases. This introduces new flexibility to the GCS algorithm to automatically exchange, test, and potentially even automatically adjust algorithm behavior toward given input data.

A surface reconstructed with the GCS approach might expose inconsistently oriented surface areas. The twist solving method presented in this thesis efficiently and reliably removes this limitation from the GCS algorithm [AB12b]. This method is achieved with novel data structures. Additional computer graphics application cases for these data structures, such as *geodesic distance* calculations, are presented to demonstrate the potential within the introduced processing strategy [AB14a].

Growing surface structures (GSS) represent a redesigned GCS algorithm concept, which explicitly incorporates the created surface into the algorithms learning scheme [AB13, AB14b]. This introduces entirely novel possibilities to direct the approximation behavior of the algorithm. The capabilities of the new concept are proven by evaluating an implemented surface reconstruction approach.

A final algorithm composed of the combined presented developments is compared to classical reconstruction algorithms [WALH13] and used to reconstruct especially challenging models. The algorithm is proven to be a universal, flexible, and robust high quality surface reconstruction approach, which outperforms classical approaches. With the newly added developments a novel approach is introduced which can be modified toward desired reconstruction behaviors, creates soundly oriented surfaces, and quickly adapts to given topologies even in extremely challenging reconstruction scenarios.

Chapter 2

Problem Analysis

This chapter includes a detailed analysis of unorganized point based surface reconstruction. It starts with the definition of the general problem, then focuses on problems within the input data, then presents properties concerning the output surface model and finally gives an overview of reconstruction methods to solve the problem.

2.1 Problem – Reconstruction from Unorganized Points

Surface reconstruction creates a 2D subspace $\mathscr{S}$ in 3D space $\mathbb{R}^3$ that represents a digital equivalent of a real world physical surface $\mathscr{S}_{phy}$. How well $\mathscr{S}$ resembles $\mathscr{S}_{phy}$ can be measured as the Euclidian distance of corresponding points on both surfaces. Minimizing these distances is known as the *surface fitting problem* (see section 2.3.1.4).
The more substantial problem, however, is to create $\mathscr{S}$ in a way that points actually have a unique correspondence on both surfaces. This requires $\mathscr{S}$ to be a topologically correct representation of $\mathscr{S}_{phy}$ (see section 6.1.3).
For surface reconstruction from unorganized points the information available about $\mathscr{S}_{phy}$ is a finite collection of surface samples $\mathscr{P} = \{\mathbf{p}_1 ... \mathbf{p}_n\}$ in 3D space $\mathbb{R}^3$. This information is inconclusive, since points cannot uniquely define a surface unless an additional set of rules is given. Thus, the task to determine $\mathscr{S}_{phy}$ from $\mathscr{P}$ is considered an *ill-posed* problem (see Fig. 2.1).

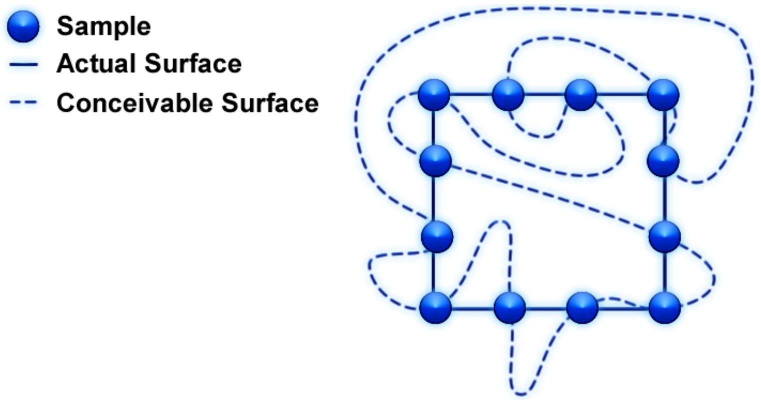


Figure 2.1: Different surfaces that both interpolate a group of samples. The samples could theoretically be a sampling of both surfaces.

The task is to search for a subspace $\mathscr{S}$ that is the most probable under the condition that $\mathscr{P}$ originates from scanning $\mathscr{S}_{phy}$. Since $\mathscr{P}$ result from a physical measuring process it contains im-

perfections. These imperfections and their influence on the reconstruction process are explained in detail in section 2.
The most basic reconstruction problem, however, remains even if every sample p is measured flawlessly $\mathbf{p} \in \mathscr{P} \Rightarrow \mathbf{p} \in \mathscr{S}_{phy}$ and the samples are uniformly distributed over $\mathscr{S}_{phy}$.
Surface reconstruction methods seek to find implications about the unknown surface $\mathscr{S}_{phy}$ in $\mathscr{P}$. Those implications are neighborhood relations in $\mathscr{P}$ from which *on-surface* neighborhoods can be derived. The problem is $\mathscr{P}$ being only accessible by 3D search queries, while on-surface neighborhoods following the 2D distribution of the yet unknown surface (see Fig. 2.2).

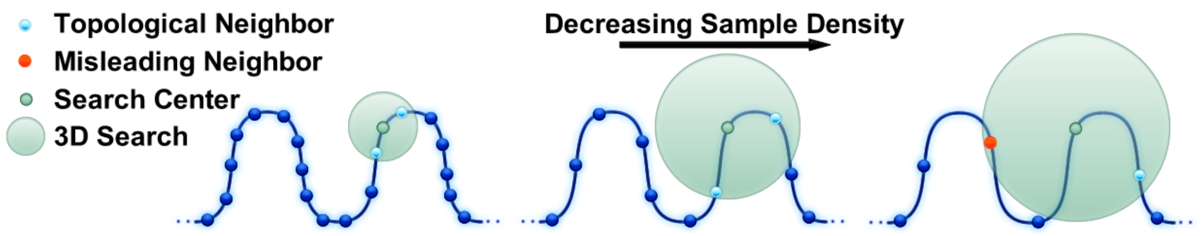


Figure 2.2: On-surface neighborhoods are investigated with 3D search queries: First, a high surface sampling, where the two on-surface neighbors to a certain sample can be easily found (left). Then the same surface with a lower sampling. The search space for the two closest neighbors has grown (middle). And at last, a low sampling where the two closest neighbors are not the correct topological (on-surface) neighbors (right).

The degree of difficulty of a reconstruction process lies in the imperfections and ambiguities in $\mathscr{P}$ and the surface model quality criteria that need to be satisfied by $\mathscr{S}$.

2.2 Input – Unorganized Points

In a realistic unorganized point based surface reconstruction scenario, $\mathscr{P}$ is obtained by an actual scanning process. Such scanned data contains imperfections, in contrast to synthetic or pre-processed data. Fig. 2.3 illustrates common challenges within unorganized point data. In the following, these challenges are explained and possible solutions are presented.

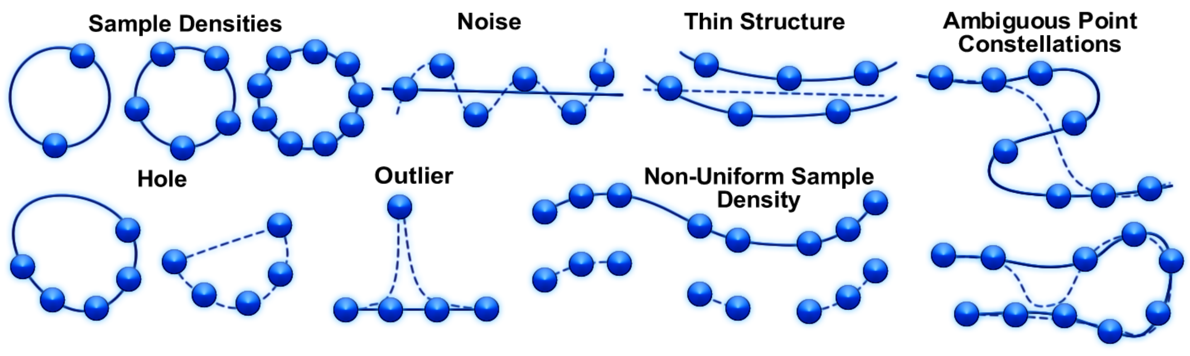


Figure 2.3: Common challenges for a reconstruction process within an unorganized point cloud.

2.2.1 Sample Set Size

Assuming every sample $\mathbf{p}$ is correctly measured $\mathbf{p} \in \mathscr{P} \Rightarrow p \in \mathscr{S}_{phy}$ and the samples in $\mathscr{P}$ are uniformly distributed over $\mathscr{S}_{phy}$, then more samples deliver more information about $\mathscr{S}_{phy}$. The more information is available about $\mathscr{S}_{phy}$, the easier the reconstruction. With a dense sampling,

however, the size of $\mathscr{P}$ becomes a problem. Especially spatial data structures such as *octrees* or *k-d trees* require huge memory resources.
If the input data is highly redundant, data reduction can be applied. Randomly selecting a sample subset is a straightforward approach. A more sophisticated approach is to insert the samples into an *octree* of fixed minimum voxel size. Then the center of every leaf node or an existing sample closest to that center is used to create a *resampling*. This also produces a uniform sampling (see section 2.2.2).
Large scanning areas, however, can involve huge amounts of samples $|\mathscr{P}|$ that cannot be reduced without compromising information about the investigated surface. In order to process such point clouds, they can be separated into single pieces, which are then reconstructed by a method that can handle open surfaces. Finally, those pieces are put together, as in [CLK09]. Algorithms such as the *co-cone* [DGH01] (see section 2.4.1.1) and *Poisson surface reconstruction* [BKBH09] (see section 2.4.3.4) have been modified to reconstruct huge point clouds by avoiding overloading the system memory, also known as an "out-of-core" solution.

2.2.2 Non-Uniform Sample Densities

Laser scanning does not produce uniform sample distributions over $\mathscr{S}_{phy}$. With the same angular resolution surfaces more distant to the scanning device are sampled with lower resolution. Also, the relative orientation of a surface to the scanner influences the sampling. Orthogonal surfaces are sampled denser than tilted ones.
Certain materials also have an effect on the sample density: transparent, dark, or reflective materials lead to very low or non-sampled areas. When several scans of the same scene are superimposed, the overlapping areas have a higher sample density than non-overlapping areas.
If a reconstruction approach assumes a fixed distance between samples, two problems can occur: too low assumptions lead to surface areas existing in $\mathscr{S}_{phy}$ but missing in $\mathscr{S}$; and too high assumptions lead to surface areas included in $\mathscr{S}$ that are non-existent in $\mathscr{S}_{phy}$.
To create a uniform sampling $\mathscr{P}$ can be added into an octree as described in section 2.2.1. Also, a temporary surface can be fitted into $\mathscr{P}$. Based on that surface a new evenly distributed sampling can be created [ABCO$^+$01]. This obviously already involves a surface reconstruction technique.

2.2.3 Holes

Scanners can only register samples on surfaces in sight. Occluded areas, also known as "scanner shadows", are not sampled. Materials that are not reflecting the laser beam back to the scanner due to absorption or non-diffuse reflection also cause holes. For single objects this can be avoided by taking scans from different positions and "powdering" problematic surfaces. For large scanning areas, however, holes are inevitable.
If a reconstruction aims to create a *solid object* (see section 2.3.4), any hole needs to be closed. Algorithms which guaranty solid objects often include hole filling mechanisms. In many cases, those approaches can be applied as post-processes on previously created surfaces that exposes holes [SSZCO10, SLS$^+$06].
If an object in contrast is desired to be non-solid, the distinction between desired and undesired holes becomes very hard. For those cases different hole filling or completion processes are needed [SWK07, CBM$^+$03].
Hole filling can be differentiated in the general ability to close holes in a surface and more sophisticated approaches that extrapolate the progression of missing geometry [SDK09, SACO04].

2.2.4 Noise

When reconstructing a real world object, $\mathscr{P}$ is the result of a physical measuring process. It is therefore bound to the precision and technical imperfections of the used scanning device. Noisy data points are displaced from $\mathscr{S}_{phy}$ and thus reduce accuracy. Noise results from several factors such as temperature, surface material, and measurement distance. Thus, point clouds usually do not have a unique noise pattern. This makes it difficult to compensate noise related distortions without causing the loss of surface detail.
Evaluating a surface fitting result by taking the distances of the points in $\mathscr{P}$ to $\mathscr{S}$ hardly makes sense in the presence of noise. If a reconstructed surface $\mathscr{S}$ mimics given noise, the supposed improved fitting that represents $\mathscr{S}_{phy}$ is actually less accurate.
Although ideal noise filters would depend on the local noise patterns, most noise removing techniques assume a constant noise pattern and a constant noise distribution. A common noise removal technique is to move a new set of points into the median of local point subsets of the initial points. This smoothes the entire point distribution, but thereby also generally reduces geometrical detail [LCOLTE07]. This approach has been improved by making the smoothing process dependent on local sample distributions [HLZ$^+$09].
Also, the surface of a noise resistant reconstruction process can be used to create a resampling [ABCO$^+$01]. In [JBS$^+$06], $\mathscr{P}$ is seen as a *probability distribution* for which in a resampling the most likely sample positions are estimated by using *Bayesian statistics*.

2.2.5 Outliers

While noise only represents a distortion of the sampling, outliers represent false information about $\mathscr{S}_{phy}$. The reason why a sample is being seen as false can vary and thus also the meaning of the term “outlier”.
Since the laser of a laser scanner is not infinitely thin, the edge of a surface and the surface behind it can both be hit partially by the laser. The measuring principle then creates a sample between both surfaces that represents neither of them. Also, the laser might hit a dust grain or an insect flying through the scene while scanning.
Outliers are commonly removed by statistical analysis of the sample distribution. In [RMB$^+$08], outliers are eliminated by calculating the mean distance of a sample to its k nearest neighbors. Based on this information, a mean distribution for the whole point cloud is computed. The mean and standard deviation of this distribution are then used to identify and remove outliers.
In addition, undesired objects such as cars, people, or technical equipment captured in a scanned scene, also referred to as “ghost geometry”, can be classified as outliers. Such structures consist of sparse but correctly registered surface samples, which are afterwards defined as *false* samples. The differentiation between a desired low sample surface and an undesired geometry of this kind is very hard. To deal with this type of outlier an algorithm needs to be adjustable to account for individually desired behavior.

2.2.6 Unrecognizable Surface Structures

Sparse sampled geometry might be entirely unrecognizable when investigating neighboring points in $\mathscr{P}$. This often occurs when scanning thin and fragile structures such as cables, wires and sheet metals, sharp features, and vegetation. Scanning these areas with special attention is often impractical when scanning complex and large scenes. To reconstruct these structures correctly, normally requires a higher level of point cloud *interpretation*, often implemented in a pre-process.

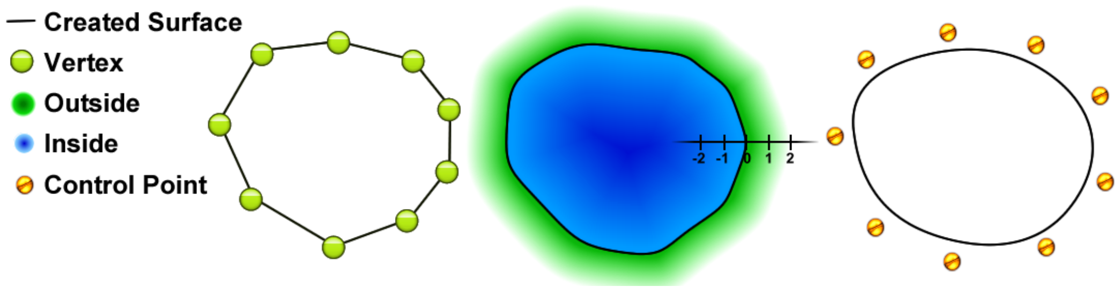


Figure 2.4: Cross-sections: A mesh (left) an implicit surface (middle), and a parametric surface (right).

Pre-processing may include the recognition of certain structures within a point cloud which a reconstruction process might not be able to handle, such as detecting sharp features [PLL12, DHOS07, JBS$^+$06, FCOS05, XMQ04].
In [HLZ$^+$09], the sampling in areas of thin surface areas is supplemented by adding additional samples. But also more complex elements can be recognized in order to be specially treated. This includes vegetation [LYO$^+$10] or elements of special interest such as stairs, windows and rooftops [SWWK08]. Model based reconstruction (see section 2.4.4) creates entire surfaces only of recognized surface elements.

2.2.7 Mutually Dependent Ambiguities

In the sections above, different point base problems were discussed. When presenting the performance of a pre-process or an algorithm for these problems, the focus is usually on one problem at a time. Some of the ambiguities of point based reconstruction, however, arise from their combined occurrence, which is common in a realistic scanning scenario.
The recognition process of sharp features becomes increasingly hard if $\mathscr{P}$ has been smoothed by noise filtering. The deletion of outliers and the reconstruction of sparsely sampled surface areas are in direct conflict. Some sample pre-processes such as noise filters, resampling, up- and down-sampling already involve a surface reconstruction method. This often makes them vulnerable to the very same problems they are meant to cure. Since many problems within $\mathscr{P}$ are mutually dependent, it is very hard to actually increase the quality of the input data without compromising its content.

2.3 Output – Surface Model

The result of a surface reconstruction process is a digital surface model $\mathscr{S}$. The quality criteria $\mathscr{S}$ needs to satisfy depend on the application case. For most practical applications $\mathscr{S}$ is an *orientable 2-manifold* with or without *boundaries*. Most fundamental for $\mathscr{S}$ is the chosen surface type, which generally defines how a surface is digitally represented. Surface properties such as its topology, orientation, and closeness depend on the operations performed to model it. In the following, these concepts and their effects on the reconstruction process are presented.

2.3.1 Surface Types

The three most common surface types in practical applications are: polygon meshes, implicit surfaces, and parametric surfaces (see Fig. 2.4).

2.3.1.1 Mesh

A polygon mesh is a piecewise linear surface, since the surface is composed of 2D polygons, which are planes that are circumscribed by a cycle of straight edges. The connection points between such edges are termed "vertices". Several polygons are connected at their edges and vertices to form a network of interconnected facets. Edges belong to two polygons inside the surface area or to one polygon at a surface boundary. The most common base polygons are triangles, leading to triangular meshes, and quads leading to quadrilateral meshes.
Triangular meshes are easy to maintain while editing and processing. Quadrilateral meshes are more intuitive in modeling and have the advantage of being more suitable for parameterization. The latter can be exploited in texturing and when fitting splines [BLP$^+$12]. If a surface has a lot of plane areas, a mesh allowing different polygons might be more efficient in terms of memory consumption.
Mesh Optimization: There are different mesh aspects for optimization. A mesh can be smoothed [Tau95] to improve the representation of curved surface areas or to eliminate rough surface areas caused by noise in a reconstruction. It can be optimized to use up as little memory as possible, while at the same time compromising as little accuracy as possible. This optimization leads to lower triangle resolutions in flat surface areas compared to curved ones. This process is often associated with the term "mesh optimization" [HDD$^+$93].
Remeshing is an optimization that aims to create a homogeneous vertex distribution over the surface area [YLL$^+$09, SLS$^+$06]. This leads to equilateral triangles, since their sides' equal lengths allow for the vertices to be placed in equally distant positions. The closeness of a triangle to be equilateral is often referred to as the *triangle quality*. The triangle quality of a mesh is important for many subsequent processes that rely on uniform vertex distributions and uniformly divided surface spaces. The different optimizations can also be combined [LTJW07].

2.3.1.2 Implicit Surface

An implicit surface is defined as a function that returns a value for any spatial position $\mathbf{p} \in \mathbb{R}^3$. It returns zero for positions on the surface, values smaller than zero for positions inside and values greater than zero for positions outside of the surface:

$$F(\mathbf{p}) = \begin{cases} < 0 & \text{for} \quad \mathbf{p} \text{ inside } \mathscr{S} \\ 0 & \text{for} \quad \mathbf{p} \in \mathscr{S} \\ > 0 & \text{for} \quad \mathbf{p} \text{ outside } \mathscr{S} \end{cases} \tag{2.1}$$

The subspace in $\mathbb{R}^3$ for which this function returns zero – also known as the *zero-level-set* of this function – is just the surface for which $\forall \mathbf{p} \in \mathbb{R}^3 : F(\mathbf{p}) = 0 \Leftrightarrow \mathbf{p} \in \mathscr{S}$ holds. The function $F(\mathbf{p})$ can be composed of a multitude of linear functions [HDD$^+$92], quadratic functions [KBH06, XMQ04, OBA$^+$03], polynomials of any degree [Lev03], varying polynomials [LCOL07], *radial base functions* (RBF) [SSZCO10, CBC$^+$01], and can even be a hyperplane in a Hilbert space [SSB05]. Some of these approaches use a weight function θ to blend locally fitted functions into one continuous surface.

2.3.1.3 Parametric Surface

A parametric surface is a piecewise smooth surface composed of *spline patches*. Splines are often cubical polynomial functions determined by a fixed number of control points. Spline patches are conceptualized to create smooth transitions when joined. In contrast to an implicit surface, spline patches have a parameterization and positions on the surface can therefore be accessed explicitly.

Common splines are *Bezier-splines* and *B-splines* [PBP02]. B-Splines have been extended to *non-uniform rational B-splines* (NURBS) [PT97] which are widely established in CAD and CAM. Building complex geometries of NURBS involves composing it by putting several non-joining pieces together. Subdivision surfaces [CC78] allow for modeling such surfaces as one coherent construct. They are based on a control mesh, which can be recursively subdivided to accomplish the desired degree of smoothness. To combine the advantages of NURBS and subdivision surfaces *T-Splines* have been proposed [SZBN03]. Parametric surfaces are typically created in a process subsequent to the reconstruction normally based on a mesh.
Segmentation: When transforming a mesh into a parametric surface, it is not replaced as a whole, but single parametric surface patches are fitted to resemble only segments of the surface. Although criteria can differ, surface segments are usually selected to be as low in curvature as possible and to have an outline of minimum length, while including as much surface area as possible [AKM$^+$06].

2.3.1.4 Continuity

The surface type of $\mathscr{S}$ determines how accurate the progression or *continuity* of $\mathscr{S}_{phy}$ can be expressed when represented with $\mathscr{S}$. Thus, the surface type determines the potential ability of a reconstruction algorithm to fulfill the surface fitting aspect of the surface reconstruction task.

2.3.1.4.1 Curvature If the progression of a surface is not linear but expresses the characteristics of a quadric or a function of higher degree, this surface area exposes curvature. Meshes are theoretically incapable of expressing curved surface areas, since they are composed of flat shapes. However, by increasing the polygon resolution in curved areas the level of inaccuracy can be minimized. With a higher resolution the model requires more memory and is less suitable for analysis, editing, and modeling processes, since more elements have to be processed. It also creates a memory consumption asymmetry between flat and curved areas.
Most base functions for implicit surfaces exceed linear functions, thus smooth areas can be represented adequately. For many use cases, however, an explicit representation is required, which makes the extraction of an isosurface necessary (see section 2.3.1.5). This transforms the implicit surfaces into a mesh exposing known problems and additionally might cause surface defects.
Parametric surfaces are also able to represent smooth surface areas and also occasionally need to be transformed into a mesh. For the parametric surface, however, the transition is well defined and can easily be adjusted. Also, the parametric surface is never actually exchanged for a mesh, but only temporarily represented as one. Therefore, parametric surfaces are extensively used when constructing and editing 3D models.

2.3.1.4.2 Sharp Features Technical structures often expose sharp features – tangential discontinuities – such as corners and creases which are crucial to represent the characteristics of an object correctly. At a tangential discontinuity the progression of a surface is not uniquely defined. Although anomalies such as infinite sharpness are not actually real world phenomena, sharp features have to be specially represented in surface modeling.
Since the transition of two polygons in a mesh is already expressed by a sharp feature, they can naturally be modeled. Some surface processing operations, however, need an explicit representation. Since all polygon transitions are expressed as sharp features, actual sharp features need to be identified first [HG01]. Implicit surfaces are normally not able to model sharp features. Here, sharp features are recognized in advance (see section 2.2.6 on recognizing surface structures) and the reconstruction is performed separately. Since parametric surfaces are already

fused together from surface patches, most models are able to express non-smooth transitions between those patches.

2.3.1.4.3 Boundaries An open surface exposes boundaries. Boundaries represent the ending of the modeling of a surface. Boundaries can be created manually when cutting off surface areas in an editing process or automatically when reconstructing areas which were deliberately excluded from a scan leading to desired holes. In a mesh, it is possible to model boundaries if the edges of a polygon are allowed to be connected to one polygon only. These edges then represent boundaries. If parametric surface patches are allowed to remain unconnected, these open ends represent boundaries. Theoretically, an implicit surface cannot represent a boundary, since its surface definition results from spatial positions being either inside or outside an object, thus excluding open objects. With some distance threshold, however, open surfaces can be represented nevertheless [HDD+92, CBC+01].

2.3.1.4.4 Self-Intersections The self-intersection of a surface, such as in the *Klein bottle*, is a progression which is normally not desired. For a common reconstruction case $\mathscr{S}$ should be a *2-manifold* and for open surfaces a *2-manifold* with boundaries. Unfortunately, all presented surface types can potentially expose self-intersections. Therefore, self-intersections have to be actively avoided in the reconstruction process.
For a mesh there are two ways to express self-intersections, the explicit way where an edge is actually connected to three or more facets, which can easily be avoided by not allowing such constructs in the first place. The second way are polygons that spatially intersect one another at some arbitrary position.

2.3.1.5 Conversion

The most common transformation is the extraction of an *isosurface* from an implicit surface. This is typically done with the *marching cubes* approach [LC87] mostly for visualization and evaluation purposes. The approach creates a uniform voxel grid around the surface. Afterwards, the surface is composed of triangles derived from cubic corners being either inside or outside the object. An extensive overview of marching cubes based polygonalization procedures is given in [NY06]. Vice versa, a mesh can be transformed into an implicit surface, for instance, by a *moving least squares* (MLS) fitting [SOS05]. Meshes are used as a basis to create a simplified control mesh for a parametric B-spline surface [LHL+12, EH96] or subdivision surface [HDD+94].

2.3.2 Topology

A sphere and a cube have different shapes, but can match one another when deformed – they are *homeomorphic* to each other. A torus, however, cannot be deformed in one of the previous objects, since it has a hole in the middle. Its transformation into a shape such as a sphere would involve cutting the surface, which is a *topological* change.
Two surfaces are homeomorphic to each other if they are topologically isomorphic, meaning that every point on one surface needs to have a unique equivalent on the other surface and vice versa, while neighbor relations for such point equivalents remain. Finding the correct homeomorphism of an object is very important in reconstruction, since a correct topology is assumed in subsequent surface processing.
In the context of mesh processing the term "topology" – conceiving the mesh as a graph – can also refer to a *discrete topology*. This discrete topology involves a finite number of positions

(vertices) and a finite number of *atomic* neighborhood relations (edges). To avoid confusion, this type of topology is always referred to as the connectivity of a mesh.

Unconnected Surfaces: Some reconstruction approaches only create $\mathscr{S}$ as one single connected surface [SLS$^+$06, DG03, Boi84]. With such an approach a point cloud $\mathscr{P}$ that contains several unconnected point subsets cannot be reconstructed correctly. This problem is often connected to the concept of having only one coherent inside space within an object (see section 2.3.4 on solid objects). To nevertheless apply these algorithms, point clouds must be separated.

2.3.3 Surface Orientation

In most computer graphics applications, surfaces are *oriented* in order to use texturing and efficient rendering techniques. To be orientable a surface needs to have a uniquely defined inside and outside at any surface position. This is also true for open surfaces with boundaries. An example for a non-orientable surface is a Möbius strip, which exposes only one side and therefore prohibits a unique definition of both inside and outside. There are three ways to establish an orientation within a reconstruction process.

Non-Oriented Surface: A reconstruction can create a non-oriented surface which is then oriented in a post-processing step. For closed surfaces (see section 2.3.4) this is trivial, since inside and outside are clearly distinguished. For open surfaces, however, an orientation propagation through the finished surface is a very complex and often ambiguous problem [DRADLN10, CLK09, HF08].

Initial Orientation Estimate: An initial orientation can be used as a basis to estimate the orientation of a newly created surface in a reconstruction process. In a region growing approach, the orientation of the first triangle determines the orientation of new triangles connected to the first [GK02, BMR$^+$99] (see section 2.4.1.2). These approaches often fail for locally unrecognizable point constellations.

An initially oriented surface can also be adapted in a deformation approach [SLS$^+$06, IJS03b, HV98] (see section 2.4.2). These techniques might fail if an adaptation process is caught in a local minimum or one surface area is represented by two different surfaces, which expose different orientations. This problem is discussed in more detail in section 5.1.1.

Normal Estimation: Many common surface reconstruction algorithms estimate the orientation in advance of the actual reconstruction process. They do not use unorganized points, but data points augmented with normals, which explicitly define the surface orientation [KBH06, OBA$^+$03, ABCO$^+$01, CBC$^+$01] (see Fig. 2.5).

If range image data is used or a point cloud is derived from a mesh or an implicit surface, normals are available. Normals can also be derived from certain technical conditions. For example, in [SSZCO10] the direction to the scanner head from each sample point is utilized.

For unorganized points normals are generally not provided and have to be estimated. Normal estimation typically consists of two steps: the independent estimation of the normal of each single sample point, and a consolidation step which unites the independent normals to deliver a consistent global surface orientation.

A normal estimation process is similar to a surface reconstruction process, since the normal estimation requires knowledge about the unknown surface $\mathscr{S}_{phy}$. Therefore, problems such as noise, outliers, and non-uniform sample densities concern this process equally. Most normal calculation concepts assume that normal directions can be calculated on a local point level. This assumption does not hold in cases where noise and insufficiently sampled structures create ambiguous point constellations. An approach that extends the calculation domain is likely to be a fully-fledged surface reconstruction approach already.

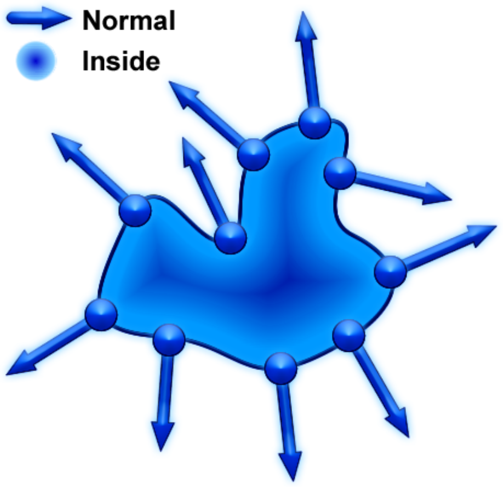


Figure 2.5: A sampling of a surface augmented with normals.

Normal Direction: A common normal estimation technique is fitting a plane to the *k* nearest neighbors to a sample under investigation [HDD+92]. The normal can be determined via *principal component analysis* (PCA). Finding the best fit to a local plane then requires finding the *eigenvector* of the lowest *eigenvalue* of the *covariance matrix*. An approach that uses filtering and thinning to improve the quality of this fit is presented in [HLZ+09]. The plane fitting problem can also be solved by *random sample consensus* (RANSAC) [RMB+08, FB81] which is robust against noise. When a *Voronoi diagram* for $\mathscr{P}$ is created, the normal directions can be associated with the connection vectors from a sample and the Voronoi vertices of its *Voronoi cell* farthest away from that sample (*poles*) [ABK98] or by using the edges of the Voronoi cells crossing through the samples [CLK09].
Normal Consolidation: To define a continuous, unique orientation, normals have to be consolidated. For densely scanned volumetric objects this can be accomplished reliably by defining the orientation based on the inner and outer poles of a vertex [ABK98]. For non-solid objects or noisy and non-uniform sampled data the orientation of the estimated normals needs to be consistently propagated through the point cloud. In [HDD+92], the orientation is propagated from one initial sample over the edges of the minimal spanning tree of the absolute dot products of neighboring sample normals. This causes the propagation paths to include the most parallel normals. In [HLZ+09], the propagation priority especially in areas of thin surfaces has been improved.

2.3.4 Solid and Non-Solid Objects

An aspect that majorly distinguishes reconstruction algorithms from one another is their orientation toward a surface or a volume. An open surface or a non-solid object has intended boundaries and therefore does not have a clearly defined inside area, which is typical when scanned environments or terrains are reconstructed. A closed surface on the other hand is called "watertight" or "solid". If such surfaces expose no self-intersections, they define a volume and can be transformed into a volume oriented representation [TWAD09].
If an object is known to be solid, this property can be exploited within the reconstruction process to create a higher quality result [SSZCO10, KBH06, HK06, SLS+06, OBA+03, CBC+01, CL96]. For such surfaces the normal estimation (see section 2.3.3) process is easier to perform. Also, the problem of differentiating between desired and undesired boundaries is non-existent, since holes can simply generally be closed. These algorithms are therefore normally suitable for hole filling tasks. Many applications that process 3D objects require watertight surfaces.

Method	Surface Type	Approximation / Interpolation	Local / Global	Open Surfaces	Needs Normals	Noise	Incomplete Data	Sharp Features	Runtime	Memory
Delaunay / Voronoi	Mesh	I	G	no	no	- -	-	+	-	- -
Region Growing	Mesh	I	L	yes	no	-	-	- -	++	+
Graph Guided Region Growing	Mesh	I	G	yes	no	-	-	+	+	-
Warping	Mesh	A	L	yes	no	+	+	+	-	++
Refinement	Mesh	A	L	yes	no	++	+	+	- -	++
Balloon Model	Mesh	A	L	no	no	+	+	+	++	+
Linear Base Functions	Implicit	A	L	yes	yes	+	+	- -	++	+
Quadric Base Functions	Implicit	A	L	yes	yes	++	+	- -	+	-
Radial Base Functions	Implicit	A	G	yes	yes	++	++	- -	- -	- -
Model Based	Implicit	A	L	yes	no	+	++	++	+	+

Table 2.1: Comparison of the presented reconstruction categories. Ratings for handling certain problems or aspects are very weak (- -), weak (-), strong (+), and very strong (++).

2.4 Function – Reconstruction Methods

As diverse as the problem cases concerning surface reconstruction are, so are the suggested concepts to solve them. It is challenging to find a sensible categorization that overlaps as little as possible.
Common categorizations are: the way the point data $\mathscr{P}$ is accessed, in local subgroups or globally all points at once; the solidity of the modeled object, being a non-solid open surface or a solid closed surface defining a volume; the used regression strategy, being either an approximation or interpolation of $\mathscr{P}$; the boundlessness toward the created topology of $\mathscr{S}$; and input data restrictions such as the requirement of estimated normals.
In the following, approaches are categorized by their working principle. A short explanation of each concept is given, followed by a discussion on what kind of advantages and disadvantages the corresponding principle has and which improvements have been suggested. In Table 2.1, a comparison of the concept dependent properties is presented, showing how these concepts basically perform without additional improvements. If some property can be specified by multiple values, the most common one is presented.

2.4.1 Interpolation

A mesh created by an interpolation method includes points of $\mathscr{P}$ as vertices in the resulting mesh. Since $\mathscr{P}$ is represented in $\mathscr{S}$, those points are considered to exactly match $\mathscr{S}_{phy}$, which is a problem if $\mathscr{P}$ includes noise and outliers.

2.4.1.1 Delaunay Triangulation and Voronoi Diagrams

A *Delaunay triangulation* is a triangulation of a set of 2D samples where a circle can be drawn through the vertices of any triangle and none of the other vertices arise inside of that circle.

Such a triangulation creates triangles as equilateral as possible while using the input samples. A Delaunay triangulation can also be accomplished in 3D space. In that case, samples are either projected onto a local tangent plane for a following 2D triangulation or a *tetrahedralization* can be created using the criteria from 2D with a sphere instead of a circle.
Note that in the context of surface reconstruction a Delaunay tetrahedralization is mostly just called Delaunay triangulation or 3D Delaunay triangulation. With a sufficient sample density, many of these methods can provide guaranties concerning the watertightness and the resulting homeomorphism of a surface. A broad survey that also includes the theoretical foundation of these methods can be found in [CG04].
Boissonnat suggests to carve out an object from a 3D Delaunay triangulation of $\mathscr{P}$ [Boi84]. The outside of the tetrahedralization represents the *convex hull* of $\mathscr{P}$. Tetrahedrons including the longest edges on the outside of the tetrahedralization are successively removed until all points of $\mathscr{P}$ are exposed. In his *α-shapes* approach, Edelsbrunner [EM92] also constructs a 3D Delaunay triangulation and erases all triangles or edges that do not fit into a sphere of a certain radius α. The approach is able to handle open surfaces, but it needs a very uniform sample distribution in order to have some reasonable setting for α. To avoid the latter problem it has been proposed to use varying α values [PLK05].
Amenta presented *the crust* [ABE98] algorithm for curve reconstruction of 2D samples and expanded it to 3D samples [ABK98]. The algorithm creates a *3D Voronoi diagram* from $\mathscr{P}$. From every *Voronoi cell* of a sample $\mathbf{p} \in \mathscr{P}$ those *Voronoi vertices* are determined that lay farthest apart from **p** and expose opposite directions from **p** to each other. These vertices – called “poles” – create the estimated normal line when connected to **p**. A 3D Delaunay triangulation of $\mathscr{P}$ together with all poles is constructed. All triangles that include only points in $\mathscr{P}$ are considered to be part of the surface (see Fig. 2.6). A final filtering process deletes triangles with normals that differ too much from the estimated normal lines.
Another improvement was the replacement of the Delaunay triangulation by a local search function which uses the space in between a double cone – called a “co-cone” – with **p** as its apex and with the orientation defined by the estimated normal line. The triangulation of **p** is performed with all points of the adjacent Voronoi cells that are inside the co-cone space [ACDL00]. The algorithm guaranties watertight meshes under the condition of densely sampled surfaces and in the absence of noise. The *tight-cocone* [DG03] was an additional fixing mechanism for holes in a co-cone surface that were caused by noise and non-uniform sample density. It was improved, leading to the *robust-cocone* which has additional advantages concerning the processing of noise [DG06].
The *power-crust* [ACK01] places spheres on every pole (see above) to create the *polar balls* to approximate the *medial axis transform* (MAT) of an object. The polar balls are then used to create a weighted Voronoi diagram – the power-diagram – which is – similar to the Voronoi diagram – a polyhedral mesh from which the surface is extracted. The surface is defined by the sum of the faces located between two cells of the power-diagram of which one belongs to an inner and the other to an outer pole. The approach is more noise resistant due to its backwards surface definition over the MAT. The approach also includes a mechanism which handles sharp features. Mederos shows an improved noise resistance technique [MAVdF05]. As these approaches define the surface over the approximated MAT, they produce an approximated surface rather than an interpolated one. Since this approach uses the Voronoi diagram, it is still added into the Voronoi category, although it is not an actual interpolation method.

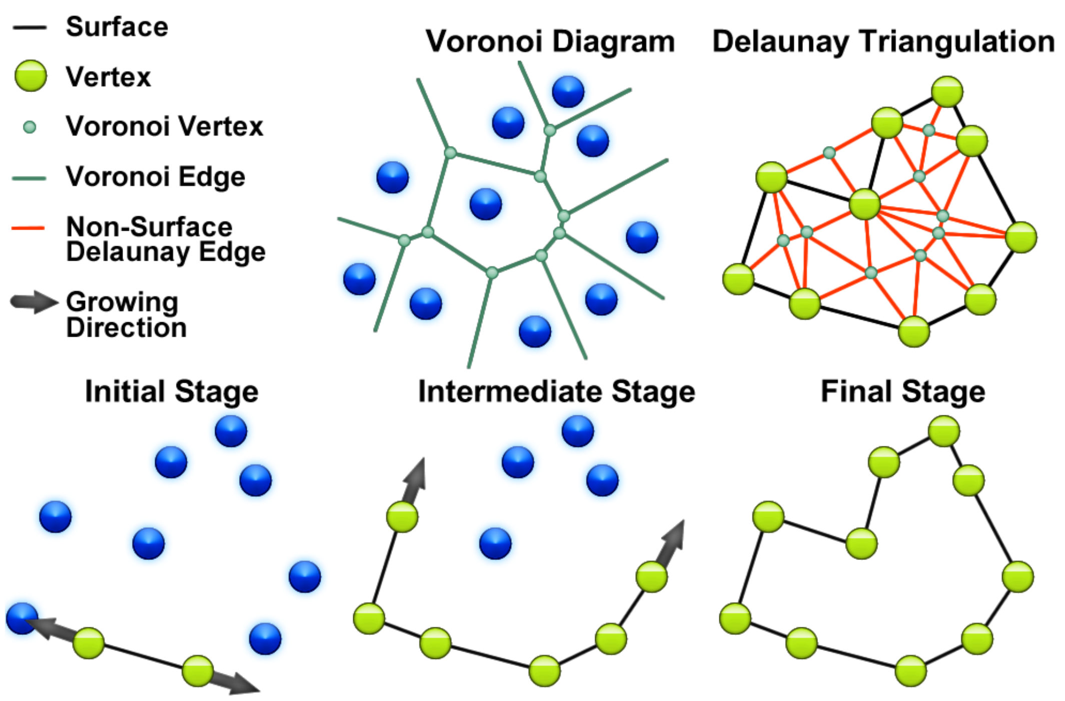


Figure 2.6: Voronoi diagram and Delaunay triangulation of the initial samples and its Voronoi vertices (top). Different stages of a region growing approach (bottom).

2.4.1.2 Region Growing

Region growing approaches are surface oriented searching strategies that try to locally interpret $\mathscr{P}$ to find $\mathscr{S}_{phy}$. Since the search strategy is based on a surface and not on a volume representation, these approaches are capable of reconstructing open surfaces. They start with a simple boundary mesh such as a single triangle that is fitted to a certain local surface area in $\mathscr{P}$. This initial surface is incrementally expanded from its boundaries (see Fig. 2.6). Since only the boundary of a surface has to be represented as data structure such approaches can be designed to be very time and memory efficient. However, merging of boundaries may create ambiguous situations and because of the local working principle, these approaches are problematic when dealing with sharp features, and with noisy and non-uniformly sampled data.
Boissonnat presented an approach [Boi84] where a list of outer triangle edges are expanded by new triangles. When a new triangle is added to one of the edges in the list, the process aims to select the yet unconnected sample which creates the triangle with the smallest angle to the surface it is connected to. Thus, if a new triangle is connected to an edge of the list, the two resulting triangles connected to the selected edge are as parallel as possible. The new edges from the new triangle are added to the list. An extension of this approach presents a more sophisticated method for adding triangles and a mechanism for dealing with non-uniform sample densities. The idea of varying the number of nearest points that are investigated is presented by Huang [HM02]. An essentially optimized version of the algorithm that solves local ambiguities is presented by Gopi [GK02].
Bernardini uses the ρ-*ball* to determine the surface [BMR+99]. If the surface is densely sampled and the size ρ of the ball is correctly chosen, the ball is "rolled" over the edges of an initial triangle and creates new triangles if the ball is "caught" from the point behind that edge. Every new triangle creates new edges. The algorithm stops if all edges have been tested without creating a new triangle.

2.4.1.3 Graph-Guided Region Growing

The following approaches introduce a surface oriented searching strategy, but utilize a global rather than a local perspective on $\mathscr{P}$ by using an additional global graph for reconstruction.
Kuo presents a region growing approach that uses a 3D Delaunay triangulation for its search base [KY05]. Cohen-Steiner shows an approach that chooses points and edges from a previously built 3D Delaunay triangulation which avoids singularities in the triangle construction [CSD04]. It includes a mechanism for handling unconnected point subsets and sharp features. Mencl uses a *Euclidian minimal spanning tree* (EMST) of the points in $\mathscr{P}$ which he extends with additional edges as a base framework for the construction of triangles [Men01]. Since the approach is surface rather than volume oriented, it can handle open surfaces, and the EMST based framework is insensitive to non-uniform sample densities. The approach also includes a mechanism for handling sharp features.
Chang [CLK09] uses a *medial scaffold* (MS) which is a one dimensional unique simplification of the *medial axis* (MA) of an object. The calculation of the MS is comparable with the creation of a Voronoi diagram. The approach creates a queue of *shock curves* which are the edges of Voronoi cells next to sample points. The approach considers the points to be the result of several surface deletion operations called “gap transforms”. These gap transforms need to be reversed in order to reconstruct the surface. The shock curves that run through three points can directly be associated with a triangle. Shock curves are ordered in a queue to create triangles in flat and smooth regions first. The evaluation of the order is based on the MS. The triangles are considered according to the order of the queue. The approach also presents an error recovery function for ambiguous situations. The algorithm stops if all shock curves have been considered. The algorithm can handle open meshes, noise, sharp features and can additionally create meshes that are not 2-manifolds and non-orientable structures. Created open meshes are non-oriented.

2.4.2 Deformation

Most deformation approaches are based on meshes which are “deformed” for creating an approximation of $\mathscr{P}$. Therefore, these approaches are conceptually able to deal with noise and non-uniform sample densities, due to their independence of exact sample locations in $\mathscr{P}$ (see Fig. 2.7). Since these approaches often have no mechanism to change the topology of their initial surface estimate, arbitrary topologies can often not be reconstructed. If a surface is falsely fitted to $\mathscr{P}$, a false state might be further optimized, but can generally not be left anymore – a local minimum.

2.4.2.1 Warping

Warping algorithms start with an initial surface estimation which then is improved by deforming it toward the samples in $\mathscr{P}$. The adaptations only involve geometric properties of the initial surface estimation and exclude the mesh connectivity. These approaches strongly depend on their initial positioning of the first surface estimate. The demanded resolution for the surface needs to be known in advance which is not easy to accomplish in the presence of noise and complex shaped geometries.
A very early work of this kind is [TV91] where a range image is approximated by deformating a grid of the same size to match it. Many more advanced approaches of this type are based on the neural network concept of the *self-organizing map* (SOM) by Kohonen [Koh82]. In these approaches [BF02, Yu99, HV98, BH93], an initial mesh with a fixed connectivity – mostly a regular grid – is iteratively deformed to match $\mathscr{P}$. Single samples are accessed randomly and the closest vertex to a selected sample and its neighbors are moved toward the sample. This basic

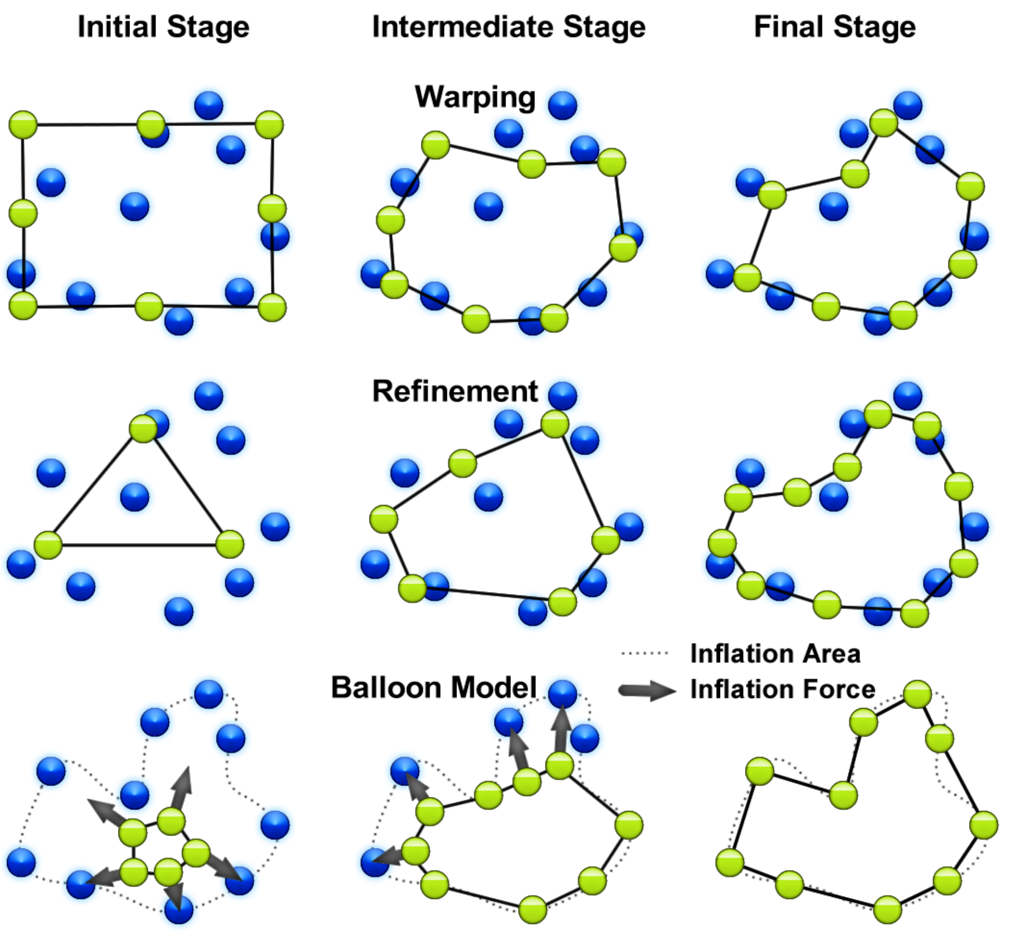


Figure 2.7: The initial (left), an intermediate (middle), and the final (right) stage of different deformation strategies, first warping (first row) then refinement (second row) and finally a balloon model (third row).

step is iteratively repeated until some error or time condition is matched (for more detail on this approach see section 3.1.2).

Since the surface is the result of multiple surface adaptations instead of a representation of exact samples such approaches are very robust when dealing with noise, non-uniform sample densities, and outliers. Since only single samples of $\mathscr{P}$ are accessed, the number of points $|\mathscr{P}|$ which can be processed is virtually unlimited.

Another class of warping algorithms use physics based models [AS96]. Here, *masses* are assigned to every vertex of a quadrilateral mesh and the mesh edges are interpreted as *springs*. Every vertex is connected with another spring to its closest point $p \in \mathscr{P}$. The springs drag the mesh toward $\mathscr{P}$. This is performed iteratively. The initial mesh is generated by adding $\mathscr{P}$ into a regular voxel grid and using the corners of the outer squares as vertices. This allows for reconstructing surfaces of arbitrary genus. The problem with the initial mesh generation is that a uniform sample density is essential for this approach and that it is only practical in the case of solid objects.

In [EBV05], $\mathscr{P}$ is also added into a regular voxel grid, but the empty voxels are also kept, building a bounding box. The algorithm then determines the biggest areas of aligned voxels on each of the six sides of the bounding box. At the initial stage all voxels at every side are aligned. Empty voxels of these aligned voxels, that are not surrounded by voxels which contain points, are continuously removed. The process is repeated with the newly created aligned voxel areas until no voxel can be removed anymore. Intrusions of the surface are avoided through a backtracking mechanism. Generally speaking the process shrinks a discrete surface onto the object surface

and can therefore also be seen as warping algorithm. After that the discrete surface is obtained and an implicit surface is created based on it.
Zhao's method [ZOF01] also includes shrinking a voxel grid to the volume of the actual object. The voxel grid is then used to build an initial surface that is applied as a flexible membrane to $\mathscr{P}$ and fitted by using partial differential equations. Again, the result is an implicit surface.

2.4.2.2 Refinement

A refinement approach can basically be seen as a warping approach (see previous section) that is enhanced by a mechanism which adapts the surface resolution if an area in $\mathscr{P}$ cannot be represented accurately. Such algorithms usually start with a low resolution surface. The final mesh is the result of a vast number of gradual refinements toward the sample distribution, thus the result depends very little on the mesh initialization. The process shapes – as in warping – and refines the mesh to create $\mathscr{S}$. Here, any mesh stage within the refinement process represents a valid solution $\mathscr{S}$ for the reconstruction of $\mathscr{S}_{phy}$.
[VT92] presents a refinement process working on a springs and masses model (see section above) that deforms a sphere-shape. The mesh is initialized as an icosahedron. If the spring model reaches a *force equilibrium*, the distance between $\mathscr{P}$ and the surface is calculated. If the distance of a triangle exceeds a certain threshold, a subdivision is triggered. This process is repeated until there is no need for further subdivisions.
An extension of the SOM (see section above) is the *growing cell structures* (GCS) approach introduced by Fritzke [Fri93]. When the process adapts mesh vertices, it additionally tracks an approximation error. The process then adds new vertices in areas of high approximation errors. Usually the approach is initialized with a simple tetrahedron. A GCS based surface reconstruction approach [VHK99, IJS03a, IJS03b, AB12c] is very powerful when creating soundly orientated surfaces since it builds newly created surfaces upon former versions of that surface. This gives a surface a certain "inertia" when being modified which avoids local failures caused by ambiguous point constellations (for more detail on this approach see chapter 3).
Since the current mesh can be taken as the result $\mathscr{S}$ at any given time, the process can be stopped and resumed when demanded, creating different resolutions of $\mathscr{S}$. In [VHK99], the use of GCS for surface reconstruction is introduced. The process has been further improved by optimizing the produced mesh quality [IJS03a]. The resulting topology of the process is limited to be homeomorphic to its initial mesh. Cutting and coalescing of the mesh during the iterative growing process are demanded to match homeomorphisms to arbitrary topologies. In [IJS03b], the triangle size is suggested as an indicator for a cutting operation and in [AB10a] cutting the mesh is triggered by high vertex valences. A mechanism to specifically handle sharp features was suggested in [AB10a]. A disadvantage of the concept is that every vertex in the mesh is visited and refined over and over again, making it inefficient in runtime.
Growing neural gas (GNG) [Fri95] is another iterative refinement concept that works quite similarly to GCS, but it builds its refinement on an arbitrary graph instead of an oriented mesh when it is used for surface reconstruction [HF08, DRADLN10]. The mesh needs to be derived from that graph at the end of the process, which can be very challenging. Additionally, the surface is less stable in comparison to mesh based methods, since former surface stages are represented less distinctly with the arbitrary graph (for more detail on this approach see section 3.3.1).
Hornung [HK06] suggests a voxel based algorithm which additionally uses a recursive refinement concept. The process initially calculates a low resolution mesh based surface by a voxelization of the object's *visual hull*. Then all points $\mathscr{P}$ are added into those voxels that cut that surface. The voxelisation is then spatially increased to a voxel region which is termed "crust". An unsigned distance function is defined by propagating the occupation of the voxels through the

crust by building averages of the surrounding voxels. The crust is extended to a graph by adding octahedral subgraphs to any of its voxels. The edges of the graph are weighted according to the distance function and by determining edges which intersect the current surface. Then a *min-cut* [BK04] of this graph is calculated to determine the voxel faces that are used to build a new surface.
With the new surface the process can be repeated until a predetermined minimal voxel resolution is reached. The process is very powerful in handling noise and outliers and, since it is bound to volumetric objects, it is very efficient in closing holes. Similar to the iterative refinement process from above, the process is very suitable when local point ambiguities have to be solved, since new surfaces are based on former surface stages. But this stability may also produce problems if a thin surface region has been recognized incorrectly, since the process also adds inertia to a falsely recognized surface.

2.4.2.3 Balloon models

Balloon models construct a volumetric object surface by the "inflation" of a balloon inside of $\mathscr{P}$. The process finishes if the balloon cannot be further expanded. These models are limited to solid objects and are very capable of dealing with incomplete data. They are also very efficient compared to other deformation approaches since the adaptation process is limited only to fronts of the balloon which still expand.
When a mesh is used as surface base [CM95, MBL$^+$91], the process is initialized with a low resolution sphere-shaped mesh, for example an icosahedron. The initial mesh is placed inside the object. In order to simulate surface tension and to keep the surface smooth, springs are attached at the mesh edges. The inflation force depends on the samples lying ahead of a vertex in their normal direction. The normals can be estimated from the surrounding triangles of a vertex. If vertices reach their corresponding samples, they are considered *anchored* and cannot be moved anymore. If triangles exceed a certain size through the dragging process, they are subdivided. The process ends when all vertices are anchored. An extension of that model has been presented in [SLS$^+$06], where the positioning of the vertices is additionally improved by adapting them to a MLS fit (see section 2.4.3.2). The approach also allows for reconstructing objects of different topologies by adding a function that can merge fronts of the balloon.

2.4.3 Distance Function

$\mathscr{P}$ can be transformed into a discrete distance function by estimating oriented normals $\mathscr{P}_{ori}$ (see section 2.3.3). A normal and its origin point define an oriented plane. If point $\mathbf{p} \in \mathbb{R}^3$ lies above or beneath the origin point of a plane and relatively close to it, the signed distance to this plane can be assumed to be the distance of $\mathbf{p}$ to $\mathscr{S}_{phy}$. This defines a signed distance function $F(\mathbf{p})$ with a limited domain $\mathbf{p} \in \mathbb{D}_F$ (see the discrete distance function in Fig. 2.8). If $F(\mathbf{p})$ can be correctly completed $\mathbb{D}_F = \mathbb{R}^3$ (see Fig. 2.8), its *zero-level-set* would be the surface under investigation:

$$\forall \mathbf{p} \in \mathbb{R}^3 \wedge F(\mathbf{p}) = 0 \Leftrightarrow \mathbf{p} \in \mathscr{S}_{phy} \tag{2.2}$$

The main concept of the following reconstruction approaches is to complete an incomplete distance function. The prior estimation of normals is necessary to define the incomplete distance function which makes the previous normal estimation process a vital step in these approaches. As the signed distance indicates an inside or outside position, approaches of this kind are volume oriented. Since the measuring of $\mathscr{P}$ is known to be imperfect, the resulting surface $\mathscr{S}$ usually approximates $\mathscr{P}$. $\mathscr{S}$ can by constructed as a global function or can be composed of many locally

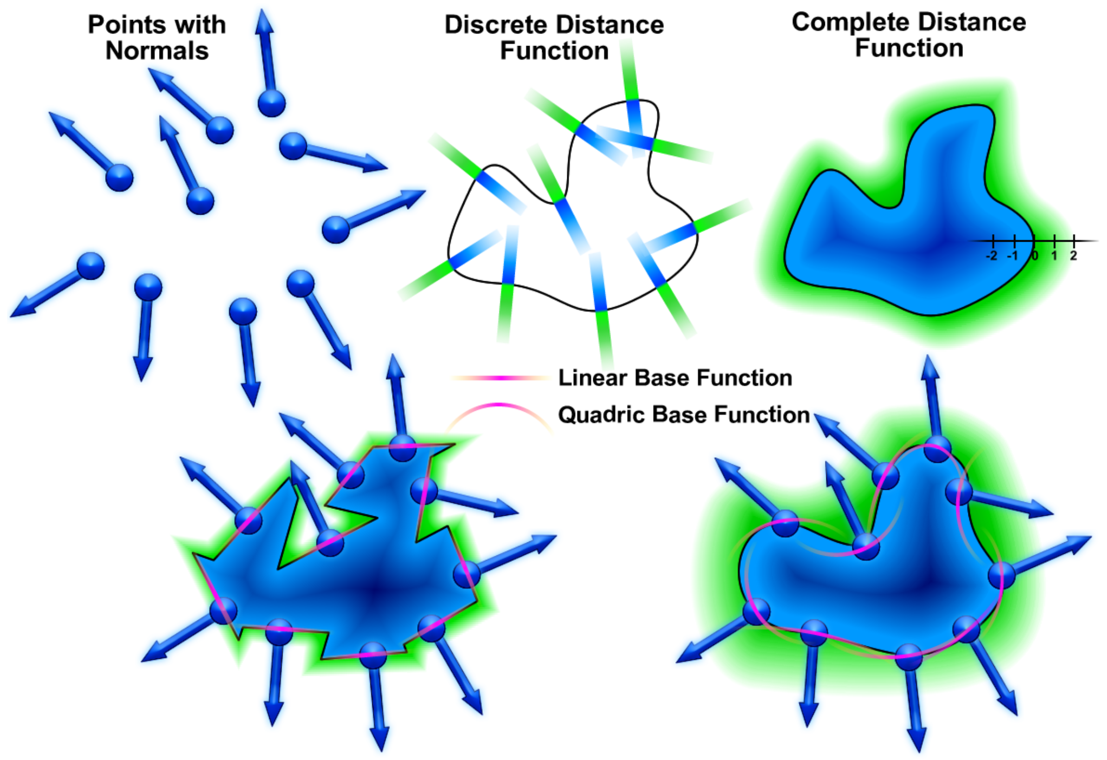


Figure 2.8: The principle of a reconstruction approach based on a distance function: The point cloud augmented with normals (top left), the discrete distance function based on it (top middle), and the completed distance function which is the result of the process (top right). The distance function can be composed of linear base functions (bottom left) or of quadric base functions (bottom right).

fitted functions which are blended together by a weight function θ. Approaches of this type are usually very powerful when processing data including noise, outliers, non-uniform sample densities, or holes. On the downside they often smooth out sharp features and small details, and in locally ambiguous point constellations normals might be orientated incorrectly which leads to models of incorrectly complex topologies.

2.4.3.1 Linear Function Base

Hoppe [HDD$^+$92] completed the distance function with piecewise linear functions. The distance to $\mathscr{S}$ for a point $\mathbf{p}$ is the distance to the plane that is defined by the normal of the closest sample to $\mathbf{p}$ in $\mathscr{P}_{ori}$. For closed surfaces this definition is sufficient to define $\mathscr{S}$. However, existing surface boundaries might be stretched to infinity. For that case the approach includes a distance threshold that depends on the assumed sample density and noise level. Note that Hoppe's approach also includes one of the most common normal estimation approaches (see section 2.3.3). The advantage of this approach is the simple calculation of $F(\mathbf{p})$, which makes the approach extremely runtime efficient.

2.4.3.2 Moving Least Squares

The *moving least squares* (MLS) approach, introduced by Backus and Gilbert [BG67], can be used for surface reconstruction in different ways [LS81]. Levin showed the infinite differentiability of a MLS surface [Lev98]. Here, generally polynomials are fitted to have the *weighted-least-square* distance to the surrounding points of a local center point. The farther away a point is from that center point, the less it "weighs" for the local polynomial fit. This is expressed by a non-negative weight function θ that, depending on the distance $d \in \mathbb{R}$, returns a

weight $w \in \mathbb{R}$. θ also blends the polynomials together smoothly. Many different weight functions have been suggested. The most prominent ones are the *Gaussian* and the *Wendland* function.
If θ decreases to zero slowly, the surface becomes very smooth, but many points have to be considered in the fitting process, making it expensive in runtime. If θ decreases fast, $\mathscr{S}$ gets closer to an interpolation. Since all points in $\mathscr{P}_{ori}$ are used as centers, the weighted-least-square fit "moves" over the surface. If all polynomials have been fitted, the surface is defined as the weighted sum of them.
A surface definition based on this concept was presented and further specified by Alexa and Levin [Lev03, ABCO$^+$01]. Levin presented a general framework for a mesh independent surface representation in arbitrary dimensional space based on the MLS concept. He suggested using a double least-square approach and a Gaussian function for θ. In a first step, local reference domains are defined. In 3D those domains are planes, and their orientation can be taken from $\mathscr{P}_{ori}$, but for the origin (center point) of these planes the first least-square fit must be accomplished. The sample to which a normal belongs is moved alongside the normal line until the weighted-least-square distance to its surrounding points is minimal. This guarantees that – in the case of noise and outliers – the center point is close to the assumed surface to have a surface centered weight in the next step.
Then a polynomial is fitted in this local coordinate system to have the weighted-least-square distance to the surrounding points. Afterwards, the center points are projected onto the polynomial and define a new set of points. This new set of points represents $\mathscr{S}$ by performing the same calculation again without the last projection step. Usually $\mathscr{S}$ can be accurately represented with a fraction of the initial samples in $\mathscr{P}$.
Alexa implemented such an approach by using quadric polynomials. The approach can handle noisy data sets and also incomplete data to a certain degree. It reduces the sample density by the removal of redundant samples and adds points at surface locations of low sample density.
MLS surface is used in many additinal computer graphics applications, such as point-rendering [ABCO$^+$01, AK04, KB04] and model animation [MKN$^+$04]. To model sharp features they have to be recognized initially [FCOS05] or the polynomials have to vary depending on the locally approximated data [LCOL07].

2.4.3.3 Multi-Level Partition of Unity Implicits

Ohtakes approach [OBA$^+$03] locally fits quadric functions into $\mathscr{P}_{ori}$ which are then blended together with a weight function. The core asset of this approach is domain decomposition. An octree-based subdivision is applied which subdivides spherical regions that expose a high normal variance. The different spatial expansion of the local functions is accounted for in the weight function. Sharp features are recognized in a clustering process. Since every sphere has to include a certain minimum of points, incomplete data can also be handled. Because of the spatial decomposition scheme the process is very fast and suitable for huge point clouds.

2.4.3.4 Poisson Surface Reconstruction

In Kazhdan's approach, the surface is represented as an *indicator function* χ which indicates – for a position $\mathbf{p} \in \mathbb{R}^3$ – whether it is inside or outside the surface [KBH06]. $\mathscr{P}_{ori}$ can be interpreted as a sampling of the gradient $\nabla\chi$ of that indicator function χ and therefore $\mathscr{P}_{ori}$ can define a continuous vector field $\overrightarrow{\mathbf{V}}$ of this gradient. Then χ can be determined as the scalar function that best matches $\overrightarrow{\mathbf{V}}$, which is an optimization problem that can be formalized as a *Poisson equation*:

$$\Delta\tilde{\chi} = \nabla \cdot \overrightarrow{\mathbf{V}} \tag{2.3}$$

$\mathscr{P}_{ori}$ is added into an octree where every node contains a *tri-variate B-spline*. All splines together define the function-space of the Poisson equation. The approach finds a global solution, since $F(\mathbf{p})$ is represented by connected spline patches, but is local in the way that patches are locally fitted, which makes it runtime efficient. The process is very robust against noise and non-uniform sample densities, but cannot explicitly represent sharp features.

2.4.3.5 Radial Base Function

To model $\mathscr{S}$ as a single global function that is continuous and differentiable is desirable for object simplicity and further processing. One way of achieving this is to compose an object of *radial base functions* (RBF). $F(\mathbf{p})$ is then defined as the sum of the outputs of all RBFs that are centered at the data-points $\mathscr{P}$ or a subset of these points.
Muraki [Mur91] defines the surface as an implicit function composed of Gaussian primitives [Bli82] and terms it the “Blobby model”. The surface fitting is formulated through an energy function. The model starts with one primitive and then iteratively splits those primitives that reduce the energy the most until the energy function reaches below a certain threshold. The problem with these approaches is the enormous computational effort for fitting the parameter settings of the base functions.
Carr [CBC$^+$01] suggests the use of *poly-harmonic* base functions. First, he defines a signed distance function by introducing off-surface points by displacing points in both directions of their associated normal in $\mathscr{P}_{ori}$. The RBFs only have to approximately fit this distance function, also known as the *fast multi-pole method*, which significantly lowers the computation effort. Additionally, the number of RBF centers is minimized by only adding new centers if the desired error margin cannot be reached. This technique is the most robust one when using incomplete data sets and it is also extremely robust when dealing with noise, which has been even further enhanced in [CBM$^+$03]. Shalom presents an improvement of the definition of the distance function that uses visibility cones that improve the hole filling ability of the approach [SSZCO10].

2.4.3.6 Support Vector Machines

Support vector machines (SVM) are an algorithmic concept originating from the area of *machine learning*. SVMs can be used for *classification* or *regression analysis*. Their basic concept for classification is to separate point groups by projecting them into a higher – or even infinite – dimensional space where they can be separated via a *hyperplane*.
In the reconstruction algorithm of Steinke [SSB05], the point groups are created by displacing every sample in $\mathscr{P}_{ori}$ along and against the normal direction, creating an inside and an outside point group. These points are then projected into a *Hilbert space* where a hyperplane is calculated with the so called “kernel-trick” to separate them. The hyperplane represents the implicit surface. Since the hyperplane performs a linear separation of inside and outside space, the resulting implicit function is very efficient. Due to its machine learning background the SVM concept is very powerful when handling noise and outliers. The authors noted that the concept should be able to basically map any demanded function type that a surface could theoretically be composed of.

2.4.4 Model Based

Model based reconstruction approaches recognize and fit entire surface components into $\mathscr{P}$. $\mathscr{S}$ is then composed from these surface components (see Fig. 2.9). This means that sharp features can directly be recognized as such. Incomplete surface areas can be fixed if their surrounding is

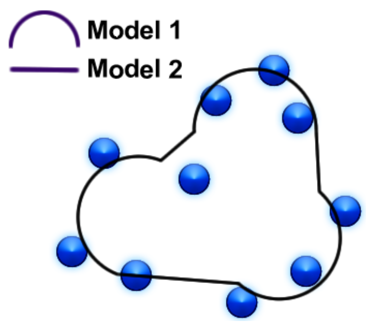


Figure 2.9: A surface composed of predetermined model elements, a semicircle and a line segment.

set into a bigger context of components that might have been recognized already. Since complete surface components are fitted, the resulting surface is robust concerning noise and outliers. These approaches are often used for creating models for CAD applications, since composing the model of primitives can achieve very compact surface representations. The problem of these approaches is to define the recognizable surface components while limiting the surface progression flexibility as little as possible. Since the surface components are predetermined specific surface details are often lost.
Buildings and indoor areas expose many plane areas which, if they can be modeled as such, lead to very space-efficient scenes [Rot03, BZ00]. In Gal's reconstruction approach [GSH$^+$07], an implicit surface is defined by a composition of a-priori defined components that are least-square fitted into $\mathscr{P}$. In Schnabel's work [SDK09], a set of primitive shapes are recognized in a point cloud to reconstruct high quality models from incomplete data sets. In Pauly's work [PMW$^+$08], repetition patterns in the point cloud can be recognized and are used to complete even areas that are missing entirely in the data.

2.5 Conclusion

This chapter presented a detailed problem analysis of the unorganized points surface reconstruction task. The problem was first defined and then reduced to the essential problem of finding 2D surface indications in a point cloud of 3D samples.
Challenges in the input data, such as noise, non-uniform sample densities, and outliers are often created in the scanning process itself. Since the presented solutions are often measured in different metrics, it is hard to compare them. Also, many problems are addressed separately, but rarely in combination, like they actually occur in real world data. Since some of the ambiguities in the point data arise from mutually dependent aspects, presented results often do not apply to real world data. Artificially created noise is another specific problem since in real world data noise levels often vary in intensity and type. Calculating the average distance of $\mathscr{S}$ to $\mathscr{P}$ as an accuracy measurement is only reasonable if $\mathscr{P}$ does not include noise.
The resulting surface model $\mathscr{S}$ can be represented in different surface types, such as a polygon mesh, an implicit surface, and a parametric surface, which determine the continuity properties of the model. Surface properties such as topology, orientation, and watertightness depend very much on the reconstruction process applied. To precisely determine the capabilities of an approach can be very challenging, since most involve many parameters. It is hard to tell how much "fine tuning" went into a particular presented reconstruction result.
An overview of common surface reconstruction methods was presented. The results of the SVM based reconstruction approach were quite impressive, since the process creates a global solution and could basically map any demanded function type. Model based approaches with

their abilities in hole filling and regaining information of completely missing surface regions appeared very promising. It is the only approach that naturally integrates sharp feature handling.

Chapter 3

Growing Cell Structures

In this chapter, the different development stages of the *growing cell structures* (GCS) algorithm toward a surface reconstruction approach are presented. The versatility of the approach is demonstrated by showing novel additional uses of the algorithm concept. The GCS algorithm is additionally distinguished from the GNG algorithm and the effects of different parameter settings are tested for both algorithms.

3.1 Evolution of Growing Cell Structures

GCS as a surface reconstruction approach is presented in a successive algorithm buildup including prior algorithm stages and algorithm extensions. All algorithm stages are presented in a fixed pattern. First, a short outline of an algorithm is given. This is followed by an introduction in which problem aspects are discussed that led to the novel algorithm. Then the approach to solve the problem is presented, first by an illustration highlighting its important features, then in a schematic presentation of its instructions (see Alg. 1 for an example). The algorithm features and instructions are then explained in detail.
This detailed algorithm presentation allows certain algorithm properties to be specifically distinguished and localized. Understanding the origin of such properties enables the required changes for the upcoming algorithm improvements in the following chapters to be identified.

3.1.1 *k*-Means Clustering

The *k*-means clustering approach [Mac67] (see Fig. 3.1 and Alg. 2) is a clustering algorithm based on *hard competitive learning*. It places a predefined number of unconnected *reference vectors* into a cloud of input points. The positioned reference vectors are used for clustering and vector quantization.
Although the term “vertex” is typically used to describe constellations of interconnected vectors, in this section, the not interconnected reference vectors are referred to as vertices to be consistent with the upcoming sections.

3.1.1.1 Introduction

When clustering measured input data, processes need to deal with noise and outliers and large amounts of input data. These are also common problems in surface reconstruction. In order to represent the input data $\mathscr{P}$ robustly in the presence of noisy or misplaced samples, a process needs to generalize the data in $\mathscr{P}$. To deal with large amounts of input data, operations that load the input data as a whole need to be avoided.

3.1.1.2 Approach

When the algorithm is initialized, it needs an initial number of vertices. The *Voronoi regions* of these vertices represent clusters after the algorithm finishes (see *(c)* in Fig. 3.1). The initial positioning of the vertices is set by randomly drawn positions from $\mathscr{P}$ (see *(a)* in Fig. 3.1 and line 1 and 2 in Alg. 2).

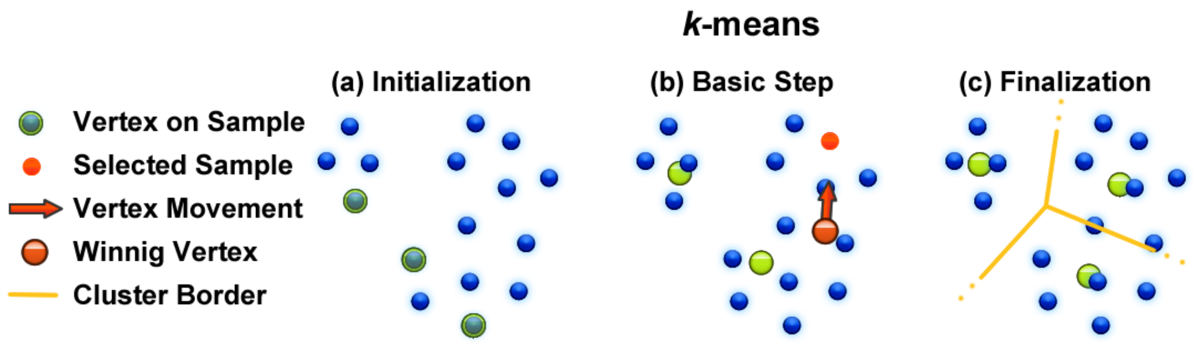
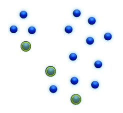
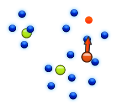


Figure 3.1: *k*-means clustering: *(a)* the initial placement of the vertices; *(b)* the basic adaptation process toward the sample distribution; *(c)* an illustration of the final point cloud clustering.

Algorithm 2 *k*-Means Clustering



```
   Initialization
1: Select an initial number of vertices
2: Initialize the positions of those vertices by random positions drawn from $\mathscr{P}$
3: repeat
     Basic Step
4:   Select random sample $\mathbf{p}_x$ of $\mathscr{P}$
5:   Find the winning vertex $\mathbf{v}_x$ that exposes the smallest Euclidian distance to $\mathbf{p}_x$
6:   Move $\mathbf{v}_x$ as much toward $\mathbf{p}_x$ as determined by the learning rate: $\Delta\mathbf{v}_x = lr(\mathbf{p}_x - \mathbf{v}_x)$
7:   Decrease the learning rate: $\Delta lr = -f(t)$
8:   Increment the iteration counter: $\Delta t = 1$
9: until No further learning takes place: $lr = 0$
```



Then the main loop of the algorithm is started, which iteratively repeats the *basic step* (see *(b)* in Fig. 3.1). The basic step starts by only loading one random sample $\mathbf{p}_x$ from $\mathscr{P}$ (see line 4 in Alg. 2). At any given time during the process only one point of $\mathscr{P}$ is loaded.

Find Closest Vertex: The vertex $\mathbf{v}_x$ that exposes the smallest Euclidian distance to $\mathbf{p}_x$ is found (see line 5 in Alg. 2). $\mathbf{v}_x$ is also referred to as the “winner”, since it won the competition for the signal $\mathbf{p}_x$. The search process for the winner is repeated many times in the algorithm and therefore needs to be executed as efficiently as possible. To avoid calculating the distance of a sample to all existent vertices, an additional geometric data structure, such as an *octree*, is needed. An octree divides space by starting with one cube – including all vertices – which is then subdivided in eight sub-cubes. This subdivision is recursively repeated. When searching for a closest vertex, only a logarithmically small fragment of search space needs to be visited.

Adapt Vertex Position: The algorithm aims to place the vertices to be a generalization of the data in $\mathscr{P}$. To achieve that, vertex positions are adapted to represent all samples for which they

won. To represent a sample $\mathbf{p}_x$ in a vertex $\mathbf{v}_x$, the vertex is moved toward the sample position. Since previous sample positions already represented in $\mathbf{v}_x$ should not be lost, its position is only changed by a fraction. The size of this fraction is determined by the *learning rate lr*. The vector from $\mathbf{v}_x$ to $\mathbf{p}_x$ defines the movement from $\mathbf{v}_x$ toward $\mathbf{p}_x$, which is scaled by the learning rate and added to position $\mathbf{v}_x$ (see line 6 in Alg. 2). On the one hand the learning rate determines how strong new samples are "learned" by moving $\mathbf{v}_x$, on the other hand it also determines how many positions of older samples are preserved or *memorized*. After a vertex has been adapted many times to represent the positions of its surrounding samples, it is positioned in the "means" of these samples.
The learning rate in the k-means clustering approach is constantly decreased by a function $f(t)$ (see line 7 in Alg. 2) until no further learning takes place (see line 9 in Alg. 2). t is represented as a simple loop counter (see line 8 in Alg. 2) $f(t)$ can be implemented in different ways, such as a constantly, linearly, or exponentially progressing function.

3.1.2 Self-Organizing Map

The *self-organizing map* (SOM) [Koh82] (see Fig. 3.2 and Alg. 3) is a *soft competitive learning* approach. Where as in hard competitive learning only one vertex is adapted (*winner-take-all*) within the SOM process additionally neighboring vertices are adapted (*winner-take-most*). Neighbor relations are established by connecting vertices through edges. Mostly vertices and edges are arranged as a rectangular shaped 2D grid. With the vertices being part of a rectangular surface, the adaptation process can recognize underlying topologies in $\mathscr{P}$. This can be used for data visualization, pattern recognition, dimensionality reduction, and regression analysis.

3.1.2.1 Introduction

When recognized, the underlying topology in $\mathscr{P}$ can be used to visualize, process, and identify additional information about the subspace the samples originate from. A vertex adaptation process guided by an underlying topology can also add stability to the vertex distribution. This creates a smoother vertex distribution, provided that the applied topology matches the one underlying $\mathscr{P}$. To establish a topology, it first needs to be created and secondly the adaptation process needs to take it into account when adapting vertex positions.

3.1.2.2 Approach

The final topology of the SOM is created immediately at initialization. The initially chosen vertices are typically connected like a rectangular 2D grid (see line 2 in Alg. 3). This initially chosen topology remains unchanged throughout the process. The 2D grid is regular and consists of triangles or quads, making the surface type of the model $\mathscr{S}$ a mesh $\mathscr{M}$.

Neighbor Position Adaption: When the vertices in $\mathscr{M}$ are initialized, their positioning is randomized creating a *chaotic* situation in $\mathscr{M}$ (see *(a)* in Fig. 3.2). To finally establish a consistently spread surface (see *(c)* in Fig. 3.2), the movements of vertices need to depend on their connectivity. This dependence is expressed in the neighbor adaptation process (see *(b)* in Fig. 3.2 and line 8 in Alg. 3). When a winning vertex $\mathbf{v}_x$ is moved, all its adjacent – connected with an edge to $\mathbf{v}_x$ – neighbors $\mathscr{N}_x$ are moved as well. This decreases the created surface tension caused by moving $\mathbf{v}_x$ (see line 7 in Alg. 3). This additional movement implicitly adds elasticity when adapting the surface. The neighbor movement is done in the same way as for $\mathbf{v}_x$ but with a smaller learning rate, resulting in a smaller movement.

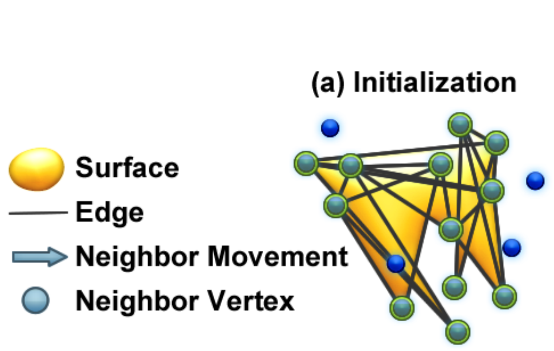


Figure 3.2: Self-organizing map: *(a)* the initialization of the 2D grid at randomly chosen sample positions; *(b)* the basic adaptation process including the adaptation of neighboring vertices; *(c)* a finalized grid positioning.

Algorithm 3 Self-Organizing Map (based on k-means clustering Alg. 2)

```
    Initialization
 1: Select an initial number of vertices
 2: Connect these vertices to a rectangular 2D grid
 3: Initialize the positions of those vertices by random positions drawn from $\mathscr{P}$
 4: repeat
        Basic Step
 5:     Select random sample $\mathbf{p}_x$ of $\mathscr{P}$
 6:     Find the winning vertex $\mathbf{v}_x$ that exposes the smallest Euclidian distance to $\mathbf{p}_x$
 7:     Move $\mathbf{v}_x$ as much toward $\mathbf{p}_x$ as determined by the learning rate: $\Delta\mathbf{v}_x = lr(\mathbf{p}_x - \mathbf{v}_x)$
 8:     Move all neighbors $\mathscr{N}_x$ of $\mathbf{v}_x$ as much toward $\mathbf{p}_x$ as determined by the neighbor learning rate:
        $\forall \mathbf{v}_i \in \mathscr{N}_x \; \Delta\mathbf{v}_i = lrn(\mathbf{p}_x - \mathbf{v}_i)$
 9:     Decrease the learning rate: $\Delta lr = -f(t)$
10:     Increment the iteration counter: $\Delta t = 1$
11: until No further learning takes place: $lr = 0$
```

In the numerous use cases of the SOM, different neighbor movements and neighbor relations above the first neighborhood degree have been suggested to the process. This is due to different application cases and the – unlike GCS – static connectivity for $\mathscr{M}$ in case of the SOM.

3.1.3 Growing Cell Structures

The *growing cell structures* (GCS) algorithm [Fri93] (see Fig. 3.4 and Alg. 4) is like the SOM a soft competitive learning approach. While the mesh connectivity in the SOM process remains unchanged, it is constantly changed in the GCS approach. Using vertex-local signal counters, the process keeps track of how often a vertex has been winning during the basic step. In areas where many signals have been counted the mesh is refined by subdivisions, while in areas of low counts the mesh is simplified by removing subdivisions. The mesh becomes more sophisticated during the process and can therefore be initialized with a very simple structure only, as illustrated in Fig. 3.3.
The GCS approach is inspired by "growing" organic tissue and the connected vertices are seen as a "structure" of "cells".
The original GCS algorithm has mainly been presented for samples in a 2D space. The GCS algorithm presented in this section includes some changes to deal with samples in a 3D space and

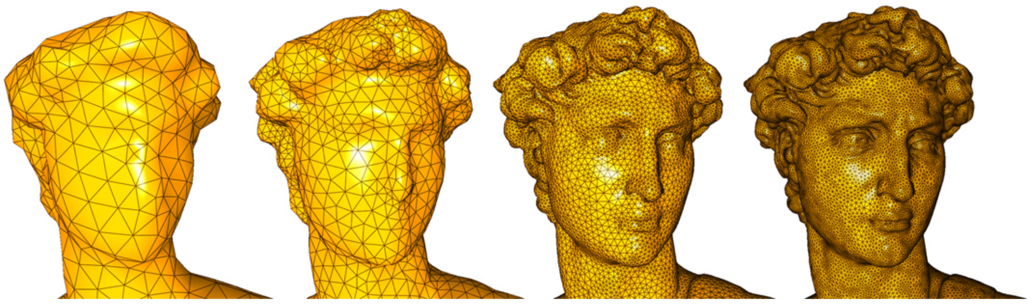


Figure 3.3: Different successive surface stages in the surface adaptation of the GCS algorithm.

to make it more suitable for surface reconstruction. These developments were mostly presented in [IJS03a].

3.1.3.1 Introduction

In SOMs, all final vertices are already present at the beginning of the process. The initial positioning of these vertices is then rearranged to create a result. This result is highly dependent on its initialization, since the rearrangement is built upon it. The result, however, is not only dependent on the initial positioning of the vertices, but also on the chosen number of vertices. If their number is insufficient to approximate the shape of the subspace of interest, a proper result is unachievable. A static initial connectivity additionally inhibits the adaptation process from adjusting the mesh resolution to the required spatial distribution.
To remove these weaknesses, the initialization of vertex positions, their number, and their specific placement in the mesh need to depend on the given sample distribution $\mathscr{P}$.

3.1.3.2 Approach

Initialization: The GCS algorithm starts with a very simple mesh $\mathscr{M}$, such as a tetrahedron (see *(a)* in Fig. 3.4 and line 1 in Alg. 4). $\mathscr{M}$ is placed in the means of $\mathscr{P}$ and scaled accordingly, meaning its vertices are more or less immersed in $\mathscr{P}$ (see line 2 in Alg. 4).

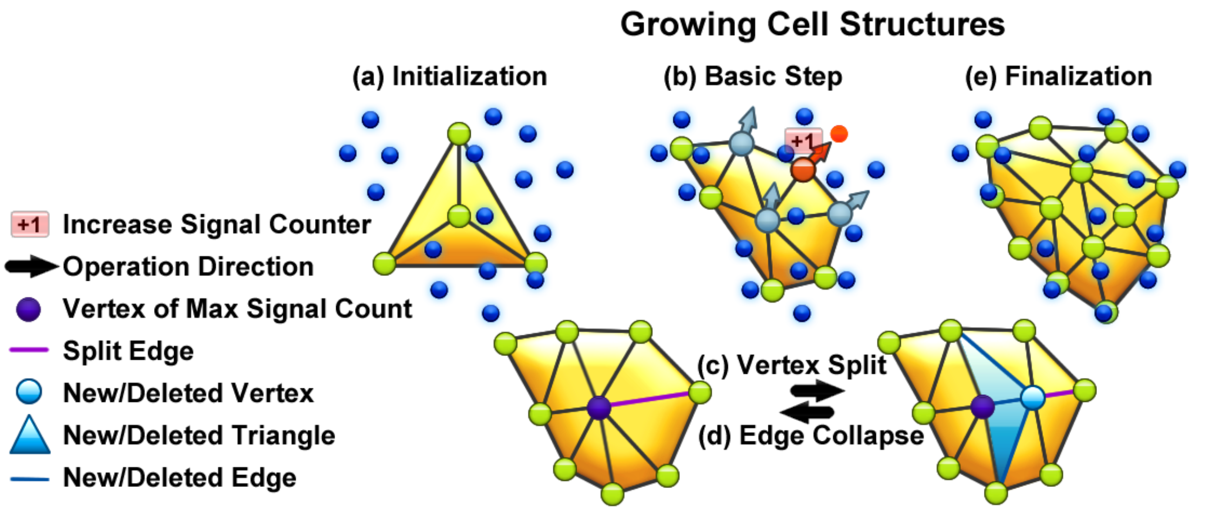
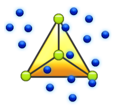


Figure 3.4: Growing cell structures: *(a)* the process is initialized with a tetrahedron; *(b)* during basic adaptation, the process additionally increases signal counters; high signal counts lead to *(c)* subdivisions via vertex split operations and low signal counts to *(d)* edge collapse operations; *(e)* a finalized mesh adaptation. Note that in *(b)* the neighbor movements do not point to the selected sample, since Laplacian smoothing is applied.

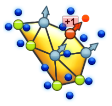
Algorithm 4 Growing Cell Structures (based on SOM Alg. 3)

Initialization



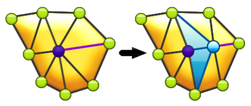
```
 1: Select an initial simple mesh $\mathscr{M}$ (In 3D, typically a tetrahedron)
 2: Place its middle point on the average of $\mathscr{P}$ and scale it according to $\mathscr{P}$
 3: repeat
 4:   repeat
 5:     repeat
```

Basic Step



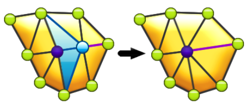
```
 6:         Select random sample $\mathbf{p}_x$ of $\mathscr{P}$
 7:         Find the winning vertex $\mathbf{v}_x$ that exposes the smallest Euclidian distance to $\mathbf{p}_x$
 8:         Move $\mathbf{v}_x$ as much toward $\mathbf{p}_x$ as determined by the learning rate: $\Delta\mathbf{v}_x = lr(\mathbf{p}_x - \mathbf{v}_x)$
            ~~Move all neighbors $\mathscr{N}_x$ of $\mathbf{v}_x$ as much toward $\mathbf{p}_x$ as determined by the neighbor learning rate: $\forall\mathbf{v}_i \in \mathscr{N}_x\ \Delta\mathbf{v}_i = lrn(\mathbf{p}_x - \mathbf{v}_i)$~~
 9:         Perform Laplacian smoothing on all neighbors $\mathscr{N}_x$ of $\mathbf{v}_x$ as much as determined by the neighbor learning rate $lrn$
            ~~Decrease the learning rate: $\Delta lr = -f(t)$~~
10:         Increase signal counter $sc_x$ of vertex $\mathbf{v}_x$ by one: $\Delta sc_x = 1$
11:         Decrease signal counters of all other vertices by a fraction: $\forall sc_i(i \in \mathscr{M})\ \Delta sc_i = -\beta \cdot sc_i$
12:         Increment the iteration counter: $\Delta t = 1$
13:     until The basic step has been performed $c_{add}$ times: $t$ mod $c_{add} = 0$
```

Vertex Split



```
14:       Select vertex $\mathbf{v}_x$ with the highest signal counter value: $\forall sc_i(i \in \mathscr{M}) <= sc_x$
15:       Select the longest edge of $\mathbf{v}_x$ and perform a vertex split operation
16:       Split $sc_x$ between $\mathbf{v}_x$ and the new vertex $\mathbf{v}_{new}$: $sc_{new} = sc_x/2 \wedge sc_x = sc_x/2$
17:   until The basic step has been performed $c_{del}$ times: $t$ mod $c_{del} = 0$
```

Edge collapse



```
18:   Select vertex $\mathbf{v}_x$ with the lowest signal counter value $\forall sc_i(i \in \mathscr{M}) >= sc_x$
19:   if Signal counter value $sc_x$ is too low: $sc_x < min_{sc}$ then
20:     Perform an edge collapse operation on the edge leading to valences closest to six.
21:   end if
22: until A certain number of vertices is reached: $|\mathscr{M}| >= n_{final}$
      ~~No further learning takes place $lr = 0$~~
```

Basic Step: The learning rate lr determines how far a vertex moves toward a sample. The SOM starts with a chaotic mesh, to leave this unordered state a large learning rate is used in the beginning. In later algorithm stages, vertex positions are assumed to represent surrounding samples and movements are decreased to preserve this knowledge.

The GCS algorithm uses a constant learning rate, since vertices are always considered to be roughly at a correct position and only need marginal adaptations. Decreasing vertex movements in later algorithm stages is implicitly reached by successively increasing the number of vertices. The subset of samples being closest to a vertex constantly decreases, making farther movements unlikely over time.

Neighbor Position Adaption: In [IJS03a], the neighbor movement (see *(b)* in Fig. 3.4 and line 8 in Alg. 3) has been replaced by Laplacian smoothing [Tau95] (see line 9 in Alg. 4). This increases the triangle quality, *i.e.*, their similarity to equilateral triangles (see Fig. 3.5). This

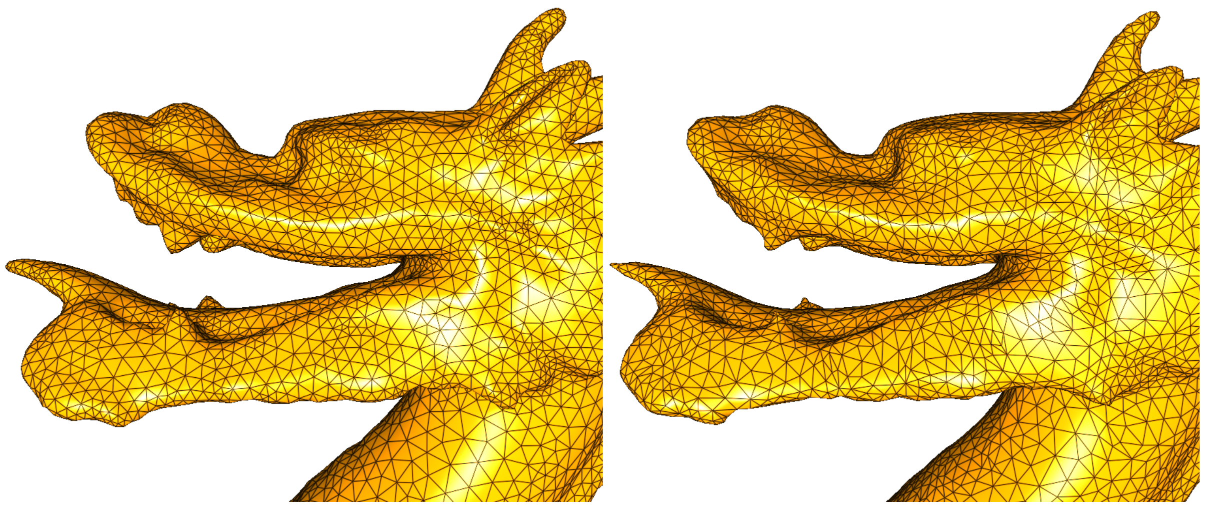


Figure 3.5: The GCS process with the Laplacian smoothing (left) and the standard neighbor movement (right). The former shows more equilateral triangles.

still fulfills the purpose of implementing surface elasticity and also the neighbor learning rate still determines how far this movement is executed. However, instead of a movement toward a sample, it now represents the movement that smoothes the surface.

Signal Counters: To refine $\mathscr{M}$, the GCS approach constantly needs to determine the best position for a subdivision. This position is determined by the highest sample to vertex ratio. Every vertex carries a signal counter, which counts the times a vertex wins (see *(b)* in Fig. 3.4 and line 10 in Alg. 4). Counted signals lose significance over time since vertices move and their subset of closest samples changes. Therefore, signal counters are gradually decreased. For every winning vertex, all other signal counters are decreased (see line 11 in Alg. 4). For meshes of small size a constant decreasing factor β is sensible, as suggested in [IJS03a, Fri93].

When reconstructing huge meshes, small decreasing rates are too slow in the beginning and large rates lead to signal counters converging to zero in later phases of the algorithm. In [IJS03b], a dynamic β value has been introduced, sensitive to the current mesh size:

$$\beta = 1 - \sqrt[(\gamma \cdot |\mathscr{M}|)]{min_{sc}} \tag{3.1}$$

After performing the basic step as many times as vertices exist in the current mesh $|\mathscr{M}|$, every vertex is supposed to have won once. The last winning vertex should "survive" all decrements while other vertices were winning, before reaching the deletion threshold min_{sc}. However, this assumes that the vertex distribution would perfectly resemble the sample distribution. The parameter γ determines how many times a vertex is allowed to be missed if $|\mathscr{M}|$ samples have been processed.

Vertex Split: After a constant number of iterations a vertex split operation is performed (see *(c)* in Fig. 3.4 and line 13 in Alg. 4). The process also selects the vertex with the highest signal counter in $\mathscr{M}$ as well as the longest edges emerging from that vertex. The edge and its connected triangles are split and a new vertex is placed in the middle of the edge. As suggested in [IJS03a], additional edges of the split vertex are connected to the new vertex to get both vertices as close as possible to a valence of six. The signal counter value is equally spread between the two vertices.

Edge Collapse: After a constant number of iterations the vertex with the lowest signal counter is selected and its signal counter is tested (see line 19 in Alg. 4). If the signal counter is below the deletion threshold min_{sc}, it is removed by an edge collapse operation (see *(d)* in Fig. 3.4 and line 17 in Alg. 4). The process selects the edge which, if collapsed, leaves behind vertices with valences as close as possible to six (see [IJS03a]).

Tumble-Tree: The basic step involves decreasing all signal counters. The vertex split operation involves finding the highest and the edge collapse operation involves finding the lowest of all signal counters. To avoid visiting the signal counters of all vertices for those operations, a specific binary search tree termed "tumble-tree" [AB10b] is used. It reduces these operations to logarithmic runtime complexity by using a *lazy evaluation* concept. This lazy evaluation lies in the decreasing factor, which is only applied to the top node of the tree. Only when actually visiting signal counter values in the tree, decreasing factors are applied to the visited signal counters and their neighboring nodes. With every search path decreasing factors "tumble" down the tree. The basic tumble-tree is implemented as a *red-black* tree and since the introduced multiplications are only performed in proportion to nodes visited in the basic *red-black* tree scheme in any case. The complexity properties of the initial tree remain unchanged.
***k*-d Tree:** The most often executed function in the GCS approach is the rearrangement of vertices in the spatial subdivision data structure. Due to the winner-take-most concept, not only the winning, but also its neighboring vertices are moved and need to be updated in the spatial subdivision data structure. Thus, optimization is most valuable here.
To increase the efficiency of this operation a *k*-d tree is used, which is adapted to the given task. A *k*-d tree divides space by a series of planes standing orthogonal to the dimensional axes of a *Cartesian coordinate system*. While for an octree the placement of the spatial fragmentation is predetermined by the initial cube, the *k*-d tree uses an initialization process to determine the best placement for the spatial separation planes.
The classic *k*-d tree repeatedly alternates through the different dimensional axes and places the planes to split the input samples as equally as possible. Thus, the input data is more evenly distributed throughout the spatial fragmentation, making fewer fragments necessary and fewer fragments need to be visited in the search process. This spatial separation is therefore more efficient than using an octree.
For the GCS approach, however, the classic *k*-d tree would be less efficient than an octree. The classic *k*-d tree is based on the notion that the points searched for are on exactly the same positions as the input data $\mathscr{P}$. This is not the case for the GCS approach. The points searched for are the vertices, which only approximately resemble the samples in $\mathscr{P}$. If these vertices move, as they do during the process, they might pass through a separation plane. If this occurs, a vertex needs to be repositioned in the *k*-d tree. If separation planes lie close to each other, even minor positional changes lead to vertex repositioning, making the data structure inefficient.
When placing the planes for the GCS vertices, the aim to efficiently separate space to minimize the number of spatial fragments still remains. Additionally, the distance between the input samples and the separating planes must be maximized. To achieve this, for every new separation plane the axis is chosen where the sample to sample-median distances are at the maximum. If, *e.g.*, a long object is fragmented, space might be repeatedly separated on the same axes, until fragments are separated into cube-like bounding volumes.
Finalization: The GCS algorithm can use an accuracy measurement to stop the refinement process, such as the average distance of the vertices to the samples. However, this is not the *sample-to-surface* distance and its significance might be very low if sample densities vary throughout $\mathscr{P}$ (see chapter 6 for further discussion on this topic). For surface reconstruction most implementations use the number of vertices in the mesh as an abort criterion (see *(e)* in Fig. 3.4 and line 22 in Alg. 4).

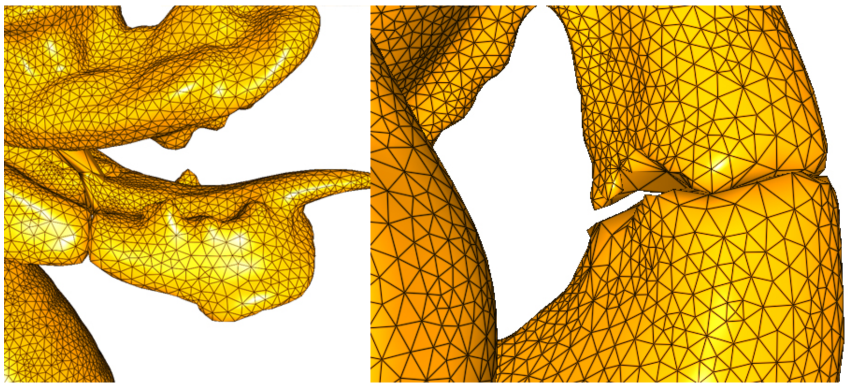


Figure 3.6: At the mouth of the Dragon (left) a surface area was formed by drawing it from a topologically incorrect surface area, creating a “pocket” in an early mesh stage. This stage cannot be left anymore and is further optimized creating a local minimum. The tail of the Dragon (right) cannot be approximated correctly, since the process was initialized with a tetrahedral (sphere-shaped), which has a different topology from the Dragon (torus-shaped).

3.1.4 Smart Growing Cells

The *smart growing cells* (SGC) approach [AB12c] (see Fig. 3.7 and Alg. 5) is a conceptual extension of GCS. Where in GCS all vertices (*cells*) behave identically, a *smart growing cell* exposes individual behavior to handle certain aspects of interest during the refinement process. If customized for surface reconstruction, different behavior at surface boundaries, sharp features and curved surface regions can be implemented. The SGC approach also allows for introducing behavior to cut and to coalesce surface regions.

3.1.4.1 Introduction

When using the suggested signal counter operation, GCS creates a *likelihood distribution*. Here, vertices are evenly distributed over the supposed surface. Solely following this aim, however, misses some important reconstruction aspects.
At boundaries, vertices need to be placed at the end of a boundary instead of in the means of their surrounding samples. Sharp features need to be explicitly modeled, since they are otherwise smoothed out. In areas of curved surfaces, a higher triangle resolution than in plane areas is required to reach the same approximation quality. When optimizing those triangle resolutions, the process is more memory efficient and more importantly gets closer to its target $\mathscr{S}_{phy}$ at an earlier algorithm stage.
An attempt to deal with this problem is presented in [JIS03] where the change in normal directions of vertices is tracked and used for adding additional vertices in such areas. A disadvantage of the suggested method is the requirement of an additional vertex adding process and normal movement counters, creating an additional process within the actual GCS algorithm.
The vertex split and edge collapse operations both preserve the pre-existing surface topology. The inability of GCS to create surfaces of arbitrary topology is their most significant constraint for surface reconstruction (see Fig. 3.6). To choose the needed topology in advance, assuming it is known or could be calculated, would not solve the problem. A more complex initial topology would make it even more likely for the process to get stuck in a *local minimum*. Here, a state of falsely fitted surfaces is further optimized, but cannot be resolved (see Fig. 3.6).
In [IJS03b], the cutting of a surface is triggered by the size of its triangles and the coalescing by a certain threshold of the *Hausdorff distance* of two boundaries facing each other. The triangle sizes that reliably indicate topologically incorrect surfaces take a long time to appear. In the case of non-uniform sample densities, where triangles naturally are of different sizes, the process is

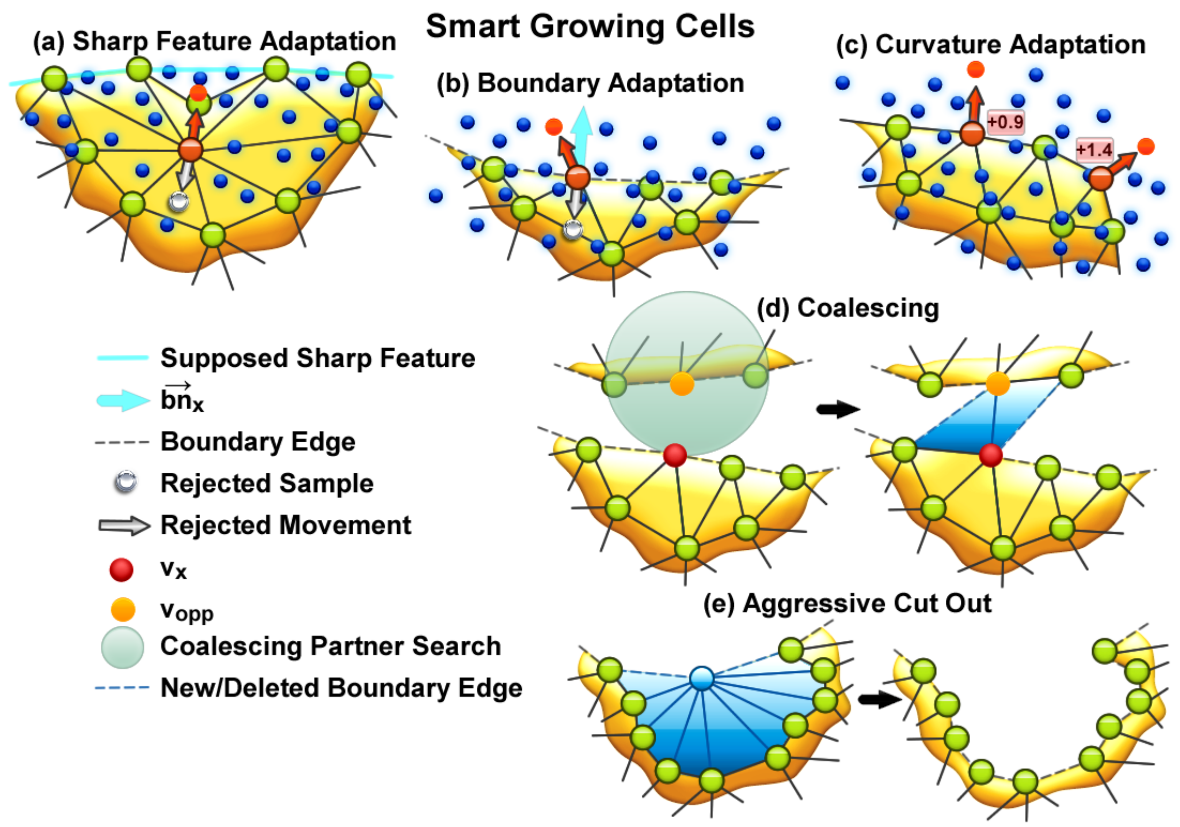


Figure 3.7: Smart growing cells: *(a)* a supposed sharp feature has been identified and only movements toward it are accepted; *(b)* a movement of a boundary vertex in surface direction is rejected while a movement away from it is accepted; *(c)* when the signal counter of a vertex is increased the surface curvature determines the increment; *(d)* the coalescing process searches for a connection partner $\mathbf{v}_{opp}$ to create new triangles; *(e)* a deletion vertex is tested for representing a misplaced surface area and cut-out from the surface entirely.

impractical. Excluding those sample distributions would take away an important advantage of the GCS approach. Additionally, the cutting and coalescing processes in the suggested form do not actually integrate into the GCS algorithm. Since especially the coalescing process is costly in runtime, it is only seldom performed in an independent loop. A solution that would be an integral component of the refinement process would be more desirable.

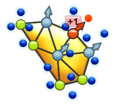
3.1.4.2 Approach

Discontinuity Adaption: The standard GCS approach assumes $\mathscr{S}_{phy}$ to be positionally continuous, *i.e.*, any point on $\mathscr{S}_{phy}$ has a uniquely defined neighbor in any direction (2-manifold without boundaries), and to be tangentially continuous, *i.e.*, any point on $\mathscr{S}_{phy}$ has a uniquely defined surface progression. To use the same movement operation for all vertices is reasonable under this latter assumption. However, if $\mathscr{S}_{phy}$ exposes discontinuities, differentiated vertex adaptations are needed.

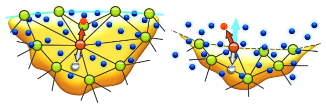
Sharp Feature: The standard GCS movement moves vertices in the means of their surrounding samples, this, however, would smooth-out sharp features. Therefore, a different movement is applied. First, a winning vertex needs to be identified to expose a sharp feature (see line 7 in Alg. 5).
A vertex is first tested for exposing an above-average curvature (see equation 3.2). If so, the curvature values around the vertex are search for two values of high curvature, while elsewise

Algorithm 5 Smart Growing Cells (based on GCS Alg. 4)

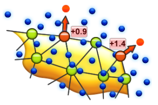
```
1:  Initialization
2:  repeat
3:      repeat
4:          repeat
                Basic Step
```



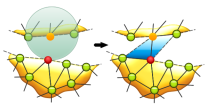
```
5:              Select random sample $\mathbf{p}_x$ of $\mathscr{P}$
6:              Find the winning vertex $\mathbf{v}_x$ that exposes the smallest Euclidian distance to $\mathbf{p}_x$
                Discontinuity Adaptation
```



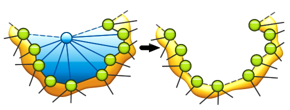
```
7:              if ($\mathbf{v}_x$ neither exposes a sharp feature OR is a boundary vertex) OR
                ($\mathbf{v}_x$ exposes sharp feature AND movement is toward sharp feature) OR
                ($\mathbf{v}_x$ is boundary vertex AND boundary is moved forward: $(\mathbf{p}_x - \mathbf{v}_x)\cdot\mathbf{bn}_x > 0$) then
8:                  Move $\mathbf{v}_x$ as much toward $\mathbf{p}_x$ as determined by the learning rate: $\Delta\mathbf{v}_x = lr(\mathbf{p}_x - \mathbf{v}_x)$
9:              end if
10:             Exclude vertices of sharp features and boundaries from Laplacian smoothing
                Curvature Adaptation
```



```
11:             repeat
12:                 Perform Laplacian smoothing on all neighbors $\mathscr{N}_x$ of $\mathbf{v}_x$ as much as determined by
                    the neighbor learning rate $lrn$
13:             until Iterations $i$ proportional to roughness: $i <= (curv_x - \overline{curv_{\mathscr{M}}})/\sigma_{curv}$
14:             Increase signal counter $sc_x$ according to its curvature: $\Delta sc_x = curv_x/\overline{curv_{\mathscr{M}}}$
15:             Decrease signal counters of all other vertices by a fraction: $\forall sc_i(i \in \mathscr{M})\ \Delta sc_i = -\beta \cdot sc_i$
                Coalescing
```



```
16:             if $\mathbf{v}_x$ is boundary vertex AND correctly oriented partner $\mathbf{v}_{opp}$ was found then
17:                 Perform coalescing operation with $\mathbf{v}_x$ and $\mathbf{v}_{opp}$
18:             end if
19:             Increment the iteration counter: $\Delta t = 1$
20:         until The basic step has been performed $c_{add}$ times: $t$ mod $c_{add} = 0$
21:         Vertex Split
22:     until The basic step has been performed $c_{del}$ times: $t$ mod $c_{del} = 0$
        Remove
23:     Select vertex $\mathbf{v}_x$ with the lowest signal counter value $\forall sc_i(i \in \mathscr{M}) >= sc_x$
24:     if $\mathbf{v}_x$ meets criteria for misplaced vertex then
            Aggressive Cut Out
```



```
25:         Cut $\mathbf{v}_x$ out including its surrounding faces and edges
26:         Clean $\mathscr{M}$ from undesired structures
27:     else
28:         Edge Collapse
29:     end if
30: until A certain number of vertices is reached: $|\mathscr{M}| >= n_{final}$
```

exposing average curvature. If all this is the case, a vertex is considered being part of a sharp feature. Then the normal of the feature is estimated and inverted if the feature is directed inwards. The movement toward the sample is compared with the feature normal. Only when pointing to a certain degree in the direction of the normal it is performed and otherwise rejected (see *(a)* in Fig. 3.7). Thus, vertices move to the farthest point of a feature instead of the means of the samples representing it. Since Laplacian smoothing also produces smooth surfaces, such vertices are excluded from the smoothing process (see line 10 in Alg. 5).
Boundary: Similar to sharp features, boundary vertices are also constrained in their movement. They are recognized by including an edge which is connected to one triangle only (see line 7 in Alg. 5). They are only allowed to move in the direction of the boundary normal $\mathbf{bn}_x$ (see *(b)* in Fig. 3.7). The boundary normal $\mathbf{bn}_x$ points in the direction where the surface at $\mathbf{v}_x$ would proceed if it was not interrupted. It is parallel to the surface and orthogonal to the boundary tangent when conceiving the boundary as a contour. This constraint makes vertices move to the end of a surface boundary instead of moving to the means of the samples in that region. Additionally, it causes open surface regions, created by the surface cutting process (see below), to close faster. Since boundary vertices have solely neighbors on one side, they would only be pulled in one direction during the smoothing process. Therefore, they are also excluded from the Laplacian smoothing (see line 10 in Alg. 5).

Curvature: Curved areas need more triangles to be accurately approximated. To set the triangle resolution in relation to the approximation accuracy, the surface curvature needs to affect the subdivision process.
The criterion for placing new vertices is the number of signals for which a vertex has won. To integrate the surface curvature into this process, signals received by curved vertices are valued higher when counted. The curvature $curv_x$ of a vertex $\mathbf{v}_x$ is calculated as one minus the average scalar product of $\mathbf{n}_x$ and the normals of its surrounding vertices $\mathcal{N}_x$:

$$curv_x = 1 - \frac{1}{|\mathcal{N}_x|} \sum_{\forall \mathbf{n}_i \ (i \in \mathcal{N}_x)} \mathbf{n}_i \cdot \mathbf{n}_x \tag{3.2}$$

This curvature $curv_x$ is divided by the average curvature $\overline{curv_{\mathcal{M}}}$ and added to the signal counter (see line 14 in Alg. 5). Thus, signal counters of average vertices still get increments of one, curved ones get higher increments and flat ones lower ones (see *(c)* in Fig. 3.7). This leads to more vertex split operations in areas of high surface curvature.
Since more vertices in the same region lead to fewer sample hits, the Laplacian smoothing is applied fewer times, reducing the mesh quality in areas of high curvature. With different operations per vertex the smoothing can be made dependent on the relative curvature of a vertex, so that in areas of high curvature more smoothing is performed. The global curvature is set in relation to the local curvature of $\mathbf{v}_x$. The number of smoothing processes is set to be the number of times $curv_x$ exceeds the standard deviation σ_{curv} of the average curvature value $\overline{curv_{\mathcal{M}}}$ (see line 13 in Alg. 5).
Coalescing: While the cutting process (see below) enables the process to cut the mesh, the coalescing enables it to connect surface regions. If a boundary vertex $\mathbf{v}_x$ is winning, a potential partner $\mathbf{v}_{opp}$ is searched for within a search radius relative to the length of the surrounding edges of $\mathbf{v}_x$. The surface region of $\mathbf{v}_{opp}$ needs to have an orientation like the one of $\mathbf{v}_x$. Thus, when connected, the new surface has a continuous orientation. If $\mathbf{v}_{opp}$ meets this criterion, a minimum of two triangles is built to connect the vertices. The triangles are chosen to expose the best possible triangle quality (see *(d)* in Fig. 3.7 and line 17 in Alg. 5).
Remove: The deletion process in the SGC process is more differentiated. If a surface region is overrepresented by too many vertices, these are removed by an edge collapse operation similar

to the GCS approach (see line 28 in Alg. 5). An additional effect of the curvature adaptation process is the deletion of vertices which are geometrically redundant. Flat surface regions do not need many vertices in order to be represented geometrically correct. With the curvature adaptation, vertices in those regions have low signal counts, due to their low curvature.
Aggressive Cut Out: Before a vertex is, however, processed in the edge collapse operation it is tested for being "misplaced" (see line 24 in Alg. 5). A misplaced vertex $\mathbf{v}_x$ is recognized by its valence $|\mathcal{N}_x|$. This indicator works since high valences are a result of the misrepresentation of the underlying sample distribution. Although vertices cannot directly be moved into sample free spaces, they might be drawn there as neighbors, due to falsely recognized topology. Such vertices are repeatedly removed, but also quickly replaced by newly drawn vertices. Since no subdivisions take place there, valences tend to grow in these regions.
If a misplaced vertex is recognized, it is removed including its surrounding edges and triangles (see *(e)* in Fig. 3.7 and line 25 in Alg. 5). All vertices involved in the first cutting process need to be tested for certain unwanted structures. Such structures include unconnected vertices, edges without triangles connected to them and vertices connected to a multitude of boundaries. The most important structures, however, are vertices with high valence, which are supposedly also misplaced and therefore are removed as well. This process is called *aggressive cut out* (ACO), since regions of misplaced vertices are not successively cut in the course of many iterations, but in one "aggressive" attempt.

3.2 Additional Usage

With marginal changes the presented approaches can be used in many different ways in computer graphics applications.

3.2.1 Resampling

The input data $\mathcal{P}$ of a reconstruction process often includes noise and outliers. Many reconstruction algorithms, however, cannot process such data, making initial noise and outlier removal necessary. The described k-means clustering approach can be used to create a *resampling*. The re-sampled $\mathcal{P}$ is represented by the resulting vertex distribution. Since the vertices represent the means of their surrounding samples, a generalization of the data in $\mathcal{P}$ is created. This generalization avoids noisy or misplaced samples from having too much of an impact on the resulting resampling. The worse the distortions in $\mathcal{P}$ are, the more generalization is required. More generalization means more samples are represented by fewer vertices, leading to an additional down-sampling. This concept has successfully been used in many distortion filters [LCOLTE07, HLZ$^+$09].

3.2.2 Unwrapping

The SOM approach typically uses a rectangle as it is easy to be parameterized by width and height. Since the rectangle topology remains unchanged throughout the process, this parameterization can always be performed. High dimensional input data can be efficiently projected into the 2D space of the rectangle. In pattern recognition and dimensionality reduction, coherences are easier to recognize and more efficient to calculate in a space of lower dimension.
The processing strategy of the SOM very closely resamples the *unwrapping* problem in computer graphics. Here, a unique projection of the surface of a 3D model $\mathcal{S}$ into a 2D rectangular shaped space is required. The process shows resemblance to "unwrapping" the packaging paper (2D

space) of a package (3D model). After this projection is found, the 2D space can be used for texturing the model.
The SOM basically *wraps* a surface around a model, which simply is an inverse unwrapping process. Therefore, its model creation inherently includes a surface unwrapping. For an existing model, an unwrapping is created by sampling the surface of that model first, and then, by using these samples to create a *wrapping* of this model with the SOM. For simple models the SOM can be used as an efficient unwrapping algorithm. However, for complex shaped models it is unlikely to find a proper unwrapping due to self-intersections and misplaced surface regions.

3.2.3 Remeshing

Many of the GCS changes presented in [IJS03a], such as the Laplacian smoothing and valence optimization in the vertex split and edge collapse operations lead to higher triangle quality. This makes the algorithm suitable to be used as a *remeshing* algorithm. A sampling of a given input mesh $\mathscr{M}_{input}$ can be used as $\mathscr{P}$. Instead of a tetrahedron $\mathscr{M}_{input}$ is used as the initial mesh. c_{add} and c_{del} are both set to the same value, *e.g.* 100, so the mesh would neither grow nor shrink. The algorithm then runs until every vertex is likely to have been moved a certain number of times, *e.g.* t is equal to five times the number of vertices in $\mathscr{M}_{input}$ (see Fig. 3.8).

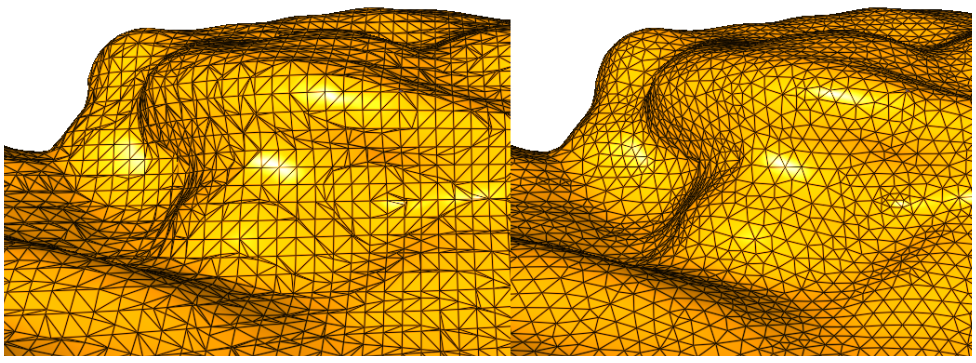


Figure 3.8: A mesh with low triangle quality created with a *marching cubes* algorithm from a MLS surface (left); Mesh after treating it with the GCS remeshing algorithm (right).

3.2.4 Level of Detail

If the GCS approach is used for reconstruction, $\mathscr{M}$ is steadily growing. In the previous described remeshing algorithm, the size of $\mathscr{M}$ stagnates. The GCS algorithm can also be entirely reversed, making $\mathscr{M}$ shrink steadily. The remeshing algorithm from above can be used, but c_{add} is changed from 100 to 500 and the abort criterion is modified to stop if the algorithm reaches or gets below a certain number of vertices. With these simple changes the GCS algorithm is transformed into a *level of detail* (LOD) algorithm. Creating differently detailed versions of the same model can, for instance, be used for *normal mapping*. Here, a low resolution version of a model is used for rendering and the geometric detail of its high resolution version is carried in a *normal map* (see Fig. 3.9).

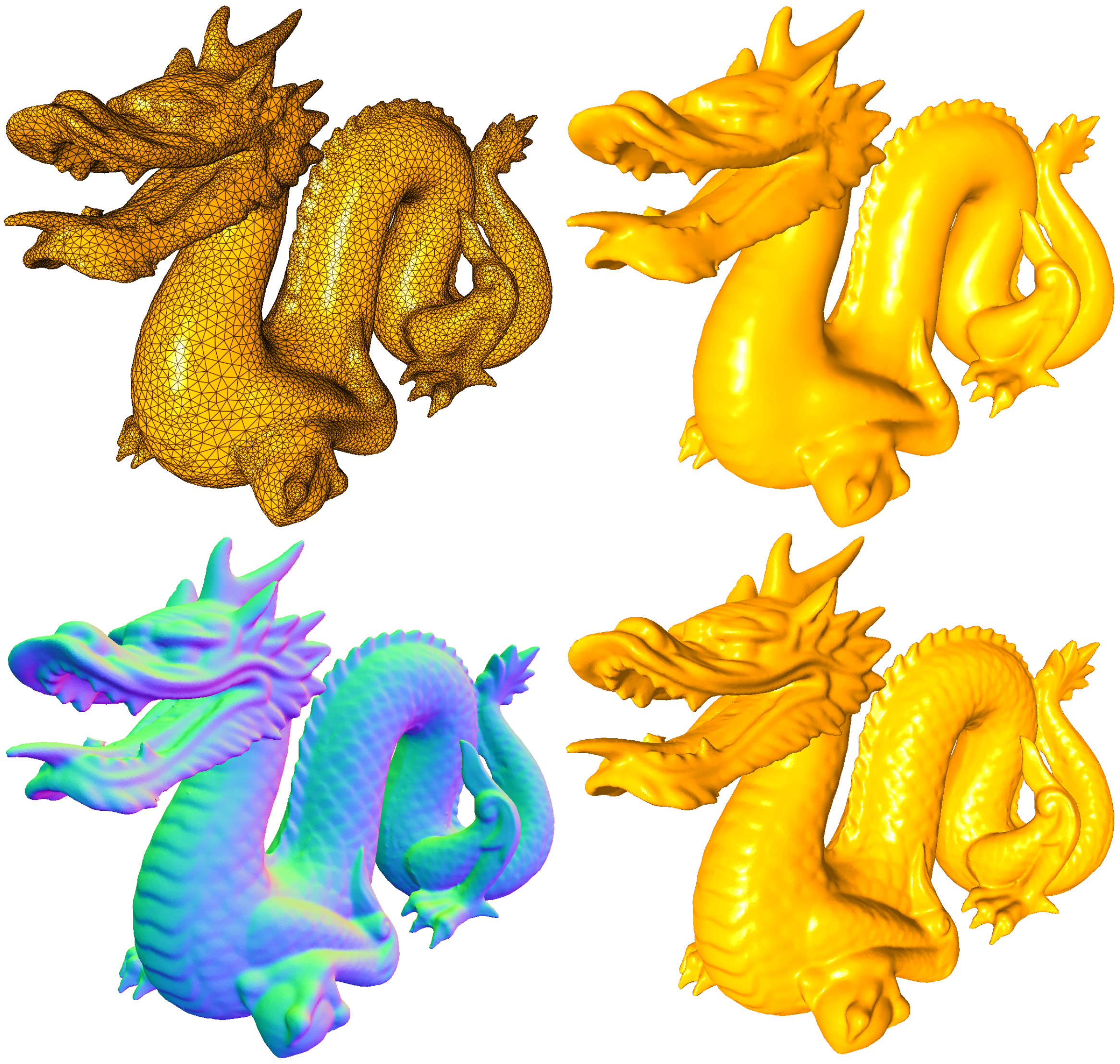


Figure 3.9: Visualization of the *normal mapping* process with a mesh (40K triangles) produced with the GCS LOD algorithm, modeling the Stanford Dragon (871K triangles): With (top left) and without (top right) wire frame; Normal map created from full resolution model shown as color texture (bottom left) and used as normal map in Phong shading (bottom right).

3.3 Growing Cell Structures vs. Growing Neural Gas

In the following section, the GCS approach is compared with the *growing neural gas* (GNG) approach. Both algorithms are applied in clustering, dimensionality reduction, data visualization, and approximation tasks. The comparison focuses on their usability as surface reconstruction processes. First, the GNG algorithm is presented. Then the similarities and differences of both algorithms are investigated, in input data, output model, topological properties, and parallelization potential.

3.3.1 Growing Neural Gas

The GNG algorithm was, as well as the GCS approach, introduced by Bernd Frizke [Fri95] and represents another competitive learning strategy. It is also based on the vertex distribution principle of the *k*-means clustering approach. The construction of connectivity, however, is different. GCS refines its mesh connectivity by subdivisions, but has a static topology and

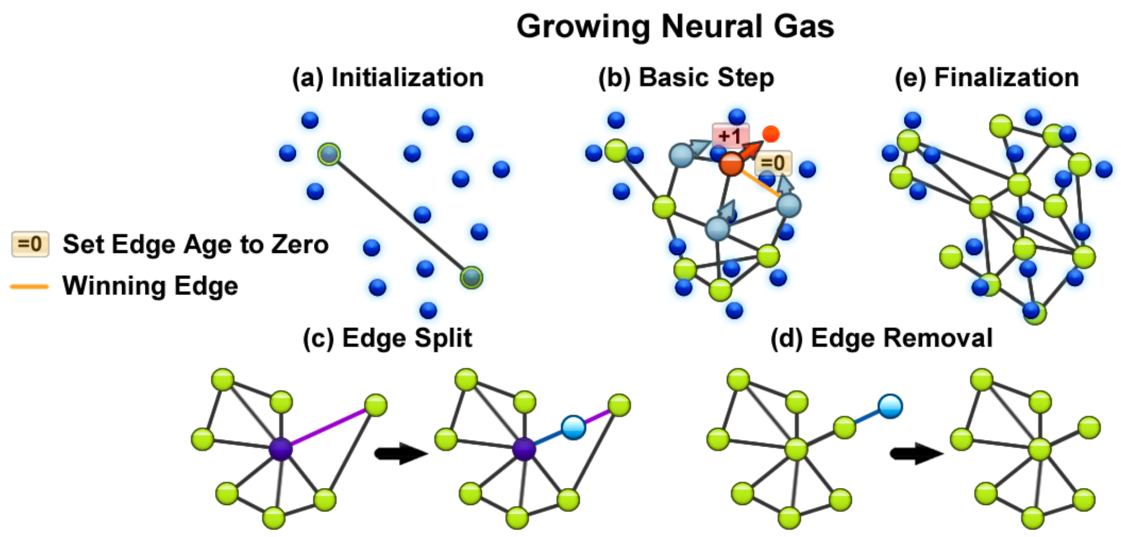
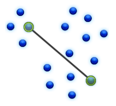


Figure 3.10: Growing neural gas: *(a)* two vertices connected by an edge are initialized on randomly chosen positions from $\mathscr{P}$; *(b)* a winning vertex and its neighbors are moved toward a sample and the age of the edge between the closest vertices to that sample is set to zero; an edge is *(c)* split by adding a new vertex in its middle and an edge is *(d)* removed and additionally a vertex is removed, since it is not longer connected; *(e)* a finalized graph. Note that the graph can include arbitrary connections and therefore does not represent a surface.

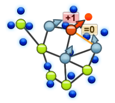
represents an extension of the SOM concept. GNG is an extension of Martinetz's *neural gas* (NG) [MS91, MS94]. Here, connections between two vertices are created if they are closest to a given sample. The topological structures built through these connections are arbitrary. Thus, the process can create line segments, triangles, and tetrahedra. Although the created graph first needs to be interpreted to recognize an actual surface, NG approach itself can already be used for surface reconstruction [MHH08].
Just as for the SOM the number of used vertices is fixed throughout the process for NG. This creates the same initialization problem as for the SOM. The sufficient number of vertices needs to be known in advance and the result depends strongly on the initial placement of the vertices. Fritzke solved the problem with a growing capability of the GNG approach. GNG, like GCS, uses vertex-local signal counters to determine areas for subdivision. The deletion process, however, is determined by the "aging" of edges and therefore is independent of signal counts. Since connectivity is created similarly to NG, the result is an arbitrary graph. GNG has also been used for surface reconstruction purposes [HF08, DRADLN10, MHH08].
Initialization: GNG is initialized with one edge and two vertices (see *(a)* in Fig. 3.10 and line 1 in Alg. 6).
Basic Step: The basic step is nearly identical to the GCS approach (see *(b)* in Fig. 3.10). However, since the Laplacian smoothing involves triangle normals, which GNG cannot provide, the process uses the standard neighbor movement (see line 10 in Alg. 6). In addition to the closest vertex $\mathbf{v}_x$ of a sample $\mathbf{p}_x$, the second closest vertex $\mathbf{v}_y$ is searched for (see line 8 in Alg. 6). If an edge $\mathbf{e}_{xy}$ between these vertices does not exist, it is created. In either case, the *age* of $\mathbf{e}_{xy}$ is set to zero. Then the age of all edges surrounding $\mathbf{v}_x$ excluding $\mathbf{e}_{xy}$ is increased (see line 15 in Alg. 6).

Edge Split and Removal: Similar to GCS, a new vertex is added after some number of iterations. The structures in the GNG graph are arbitrary, thus operations involving higher order structures, such as triangles in the classic vertex split operation, cannot be performed. GNG simply splits an edge in two and adds a new vertex in the middle (see *(c)* in Fig. 3.10). For the GNG process

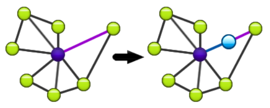
Algorithm 6 Growing Neural Gas (based on GCS Alg. 4)



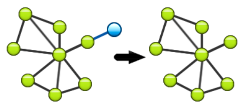
```
    Initialization
1:  Add two vertices to $\mathscr{M}$, which are connected by an edge.
2:  Initialize the positions of those vertices by random positions drawn from $\mathscr{P}$
3:  repeat
4:     repeat
5:        repeat
```



```
              Basic Step
6:            Select random sample $\mathbf{p}_x$ of $\mathscr{P}$
7:            Find the winning vertex $\mathbf{v}_x$ that exposes the smallest Euclidian distance to $\mathbf{p}_x$
8:            Find the second closest vertex $\mathbf{v}_y$ to $\mathbf{p}_x$
9:            Move $\mathbf{v}_x$ as much toward $\mathbf{p}_x$ as determined by the learning rate: $\Delta\mathbf{v}_x = lr(\mathbf{p}_x - \mathbf{v}_x)$
10:           Move all neighbors $\mathscr{N}_x$ of $\mathbf{v}_x$ as much toward $\mathbf{p}_x$ as determined by the neighbor learning
              rate: $\forall \mathbf{v}_i \in \mathscr{N}_x\ \Delta\mathbf{v}_i = lrn(\mathbf{p}_x - \mathbf{v}_i)$
11:           if $\mathbf{v}_x$ and $\mathbf{v}_y$ are not connected by an edge then
12:              Connect $\mathbf{v}_x$ and $\mathbf{v}_y$ by an edge $\mathbf{e}_{xy}$
13:           end if
14:           Set the age $a_{xy}$ of $\mathbf{e}_{xy}$ to zero
15:           Increase the age of all edges surrounding $\mathbf{v}_x$ but $\mathbf{e}_{xy}$: $\forall a_{xi}(i \in \mathscr{N}_x \setminus \mathbf{v}_y)\ \Delta a_{xi} = 1$
16:           Increase signal counter $sc_x$ of vertex $\mathbf{v}_x$ by one: $\Delta sc_x = 1$
17:           Decrease signal counters of all other vertices by a fraction: $\forall sc_i(i \in \mathscr{M})\ \Delta sc_i = -\beta \cdot sc_i$
18:           Increment the iteration counter: $\Delta t = 1$
19:       until The basic step has been performed $c_{add}$ times: $t$ mod $c_{add} = 0$
```



```
          Edge Split
20:       Select vertex $\mathbf{v}_x$ with the highest signal counter value: $\forall sc_i(i \in \mathscr{M}) <= sc_x$
21:       Select the longest edge $\mathbf{e}_{xy}$ of $\mathbf{v}_x$ and add a new vertex $\mathbf{v}_{new}$ in its middle.
22:       Split $sc_x$ between $\mathbf{v}_x$ and the new vertex $\mathbf{v}_{new}$: $sc_{new} = sc_x/2 \wedge sc_x = sc_x/2$
23:    until The basic step has been performed $c_{del}$ times: $t$ mod $c_{del} = 0$
```



```
       Edge Removal
24:    Select the edge $\mathbf{e}_{xy}$ with the highest age $a_{xy}$: $\forall a_{ij}(i,j \in \mathscr{M}) >= a_{xy}$
25:    if Age $a_{xy}$ of $\mathbf{e}_{xy}$ is too high: $a_{xy} > max_a$ then
26:       Delete $\mathbf{e}_{xy}$ and $\mathbf{v}_x$ or $\mathbf{v}_y$ in case they are not not connected to any other vertex anymore.
27:    end if
28: until A certain number of vertices is reached: $|\mathscr{M}| >= n_{final}$
```

removing connections and subdividing the graph are operations determined by different algorithm aspects. In order to be remove an edge, it has to exceed a certain maximum age. If such an edge is identified in the edge removal operation (see line 25 in Alg. 6), the edge is removed. Unconnected vertices are removed as well (see *(d)* in Fig. 3.10).
Finalization: The result of the GNG proces – in contrast to GCS – is not a surface (see *(e)* in Fig. 3.10). For surface reconstruction purposes, the resulting graph therefore needs to be interpreted and processed to be transformed into a surface. This can be a complex task.

3.3.2 Similarities

Both GCS and GNG use a competitive learning approach and both inherit all advantages from the k-means clustering approach. This includes the ability to process virtually an infinite number of samples, the possibility of online changes of the sample points and a strong noise resistance (see Fig. 3.11).
Their base loop is fairly simple and easy to understand, making them maintainable and easy to alter for individual needs. Both processes can be stopped or continued at any given time during the iterations and always expose a current approximation result. $\mathscr{S}$ is represented as a mesh or a graph which is eventually transformed into a mesh. Neither approach has been presented using an implicit surface.

3.3.3 Input – Unorganized Points

Like in every surface reconstruction algorithm, both GCS and GNG are supposed to recognize $\mathscr{S}_{phy}$ with as few samples as possible.
The basis for the existence of structures in $\mathscr{S}$, such as vertices, edges, and triangles, is differently positioned in these algorithms. In GCS, a signal counter determines whether a vertex is deleted or not. In GNG, the deletion is determined by the age of an edge. In order for both structures to “survive”, they need to be closest to a sample point. Since a mesh typically includes fewer vertices than edges, a GNG graph simply needs more samples in order to create an equivalent approximation (see Fig. 3.12).
For GCS any signal counter is associated with one vertex. In order for every vertex to constantly receive signals, the number of vertices should not exceed the number of samples $|\mathscr{P}|$.
Given that every vertex in a GNG graph has the ideal valence of six, then the total number of edges is three times the number of vertices, since every edge is shared by two vertices. In order for these edges to constantly receive signals, the number of samples needed is three times as high as the number of vertices.

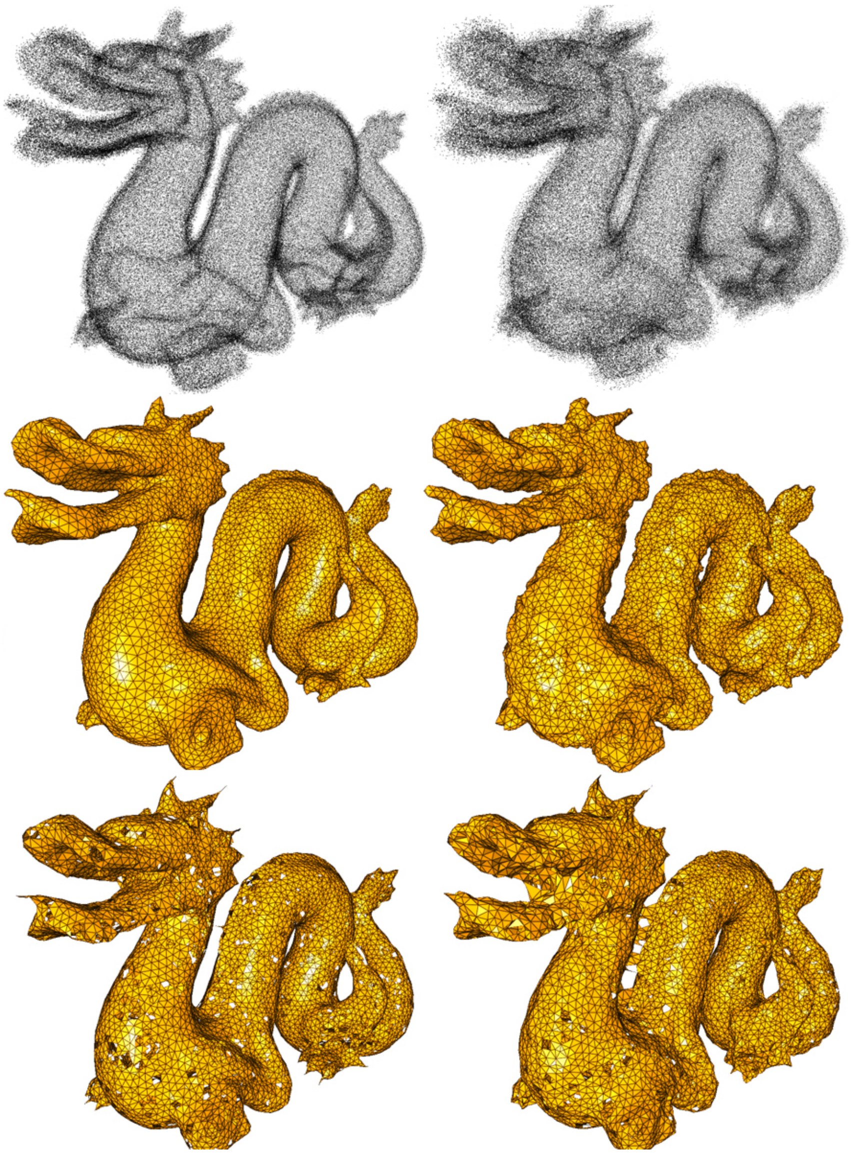


Figure 3.11: Both GCS and GNG have a natural ability to deal with noisy data. Point clouds (top row), GCS (middle row), GNG (bottom row), 2% Gaussian noise (left column) and 4% Gaussian noise (right column).

However, in practical tests, the estimate of needing three times more samples proved to be too low. The true value, most likely due to actual point distributions not being perfect, seems to be between ten to twenty times. This problem cannot, as it might seem, be solved by increasing the maximum life-time max_a of the edges, since this would strongly compromise the adaptation quality of the process (see Fig. 3.13).
The edge age can be implemented in different ways. Fritzke suggested adding a counter to each edge. These counters are increased if connected vertices are winning. This is also the case in the implementation presented in Alg. 6. This solution requires no additional data structures.
However, detached and misplaced parts of the graph might never be removed, since their vertices are not winning anymore. To avoid this, the edges can be stored in a ordered, doubly linked list. Here, the winning edge is simply put to the front to refresh its age. At the end of the list are all old edges. Besides the extra memory for the list, both implementations perform equally well and notable differences could not be determined in the performed tests.

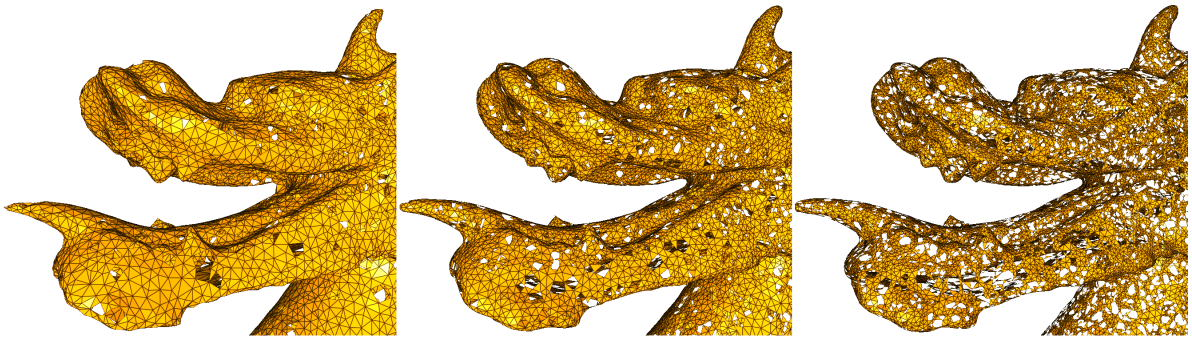


Figure 3.12: If the number of samples in the GNG process is not sufficient to match the distribution of connections in the graph, the structure starts to dissolve. From left to right Dragon model reconstructed with GNG with 60K, 120K, and 240K triangles.

3.3.4 Output – Surface Model

For GCS and GNG $\mathscr{S}$ is a graph. GCS is only allowed to include *simplices* of the previously chosen dimension, triangles in the case of surface reconstruction. For GNG $\mathscr{S}$ can contain any kind of simplex that does not exceed the dimension of the input space, for instance, tetrahedra in the case of samples in a 3D space.
Given that the dimension of the desired output graph is unknown or needs to be variable, then GNG is one of the few algorithms to provide this flexibility. However, in surface reconstruction this is not the case. Here, $\mathscr{S}$ finally needs to resemble an actual 2D surface.
GNG cannot guarantee for $\mathscr{S}$ to consist only of triangles. Thus, a post-process needs to delete or transform all structures in $\mathscr{S}$ which differ from triangles. Additionally, edges with more than two triangles attached to them need to be fixed, since they infringe the criteria for a 2-manifold (see section 2.3.1.4.4 on self-intersection). Also, a hole filling mechanism is needed to fill polygons with more than three edges. In Fig. 3.14, an overview of these structures is given.
In the presented GNG algorithm, such a hole-filling method is not implemented. First, to give an impression of the actual result of the GNG algorithm, and secondly, since such a method could be implemented in many different ways. This would lead to very different results for the final mesh, creating an undesired independence of the actually created graph of the GNG approach.
In the presented figures, $\mathscr{S}$ is shown as a graph of non-oriented triangles, not bound to be a 2-manifold. Neither GCS nor GNG in their initial papers [Fri93, Fri95] define an oriented surface.

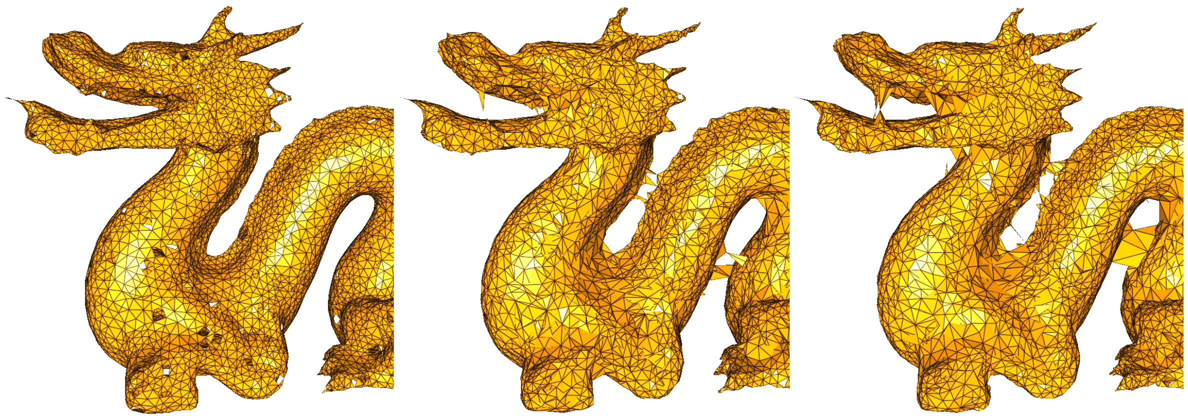


Figure 3.13: Reconstruction of the Stanford Dragon with 20K triangles: With the maximum age of $max_a = 88$ (left), $max_a = 176$ (middle) and $max_a = 352$ (right). Although the two latter reconstructions expose no holes, their adaptation process is compromised proven by many incorrectly left connections.

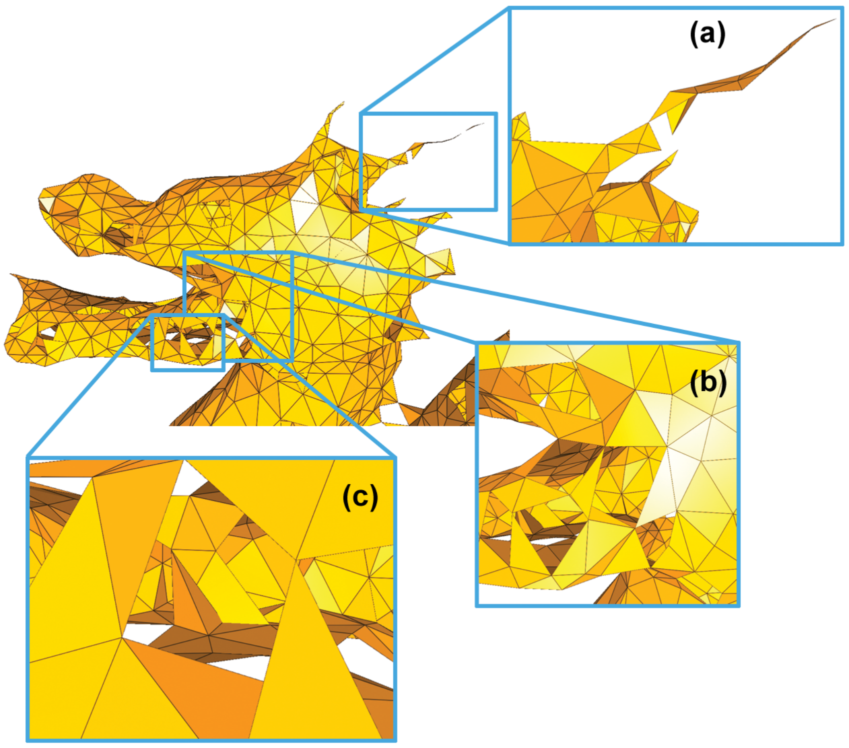


Figure 3.14: Different possible structures in a GNG graph: *(a)* a none-manifold edge with three triangles attached; *(b)* many holes in the surface; *(c)* a tetrahedron, which rather represents a volume than a surface.

In computer graphics applications, however, $\mathscr{S}$ is supposed to be oriented (see section 2.3.3 on orientation).
Although this is not part of the basic GCS definition all reconstruction algorithms presented so far used a mesh that already includes oriented triangles. For GCS a new surface is refined from a pre-existing surface, creating sound oriented surfaces even for locally ambiguous constellations of points.
Since the connection building process in GNG is not restricted, $\mathscr{S}$ cannot be constrained to consist of oriented triangles throughout the process. Therefore, refinements in $\mathscr{S}$ are not based on a pre-existing surface. If the process is finished and the necessary cleaning mechanism has been performed, the orientation of the determined triangles needs to be propagated through $\mathscr{S}$. As demonstrated in Holdstein's implementation [HF08], this can lead to inconsistencies in the surface orientation, especially in noisy areas.
Since connectivity changes are unrestricted, GNG is naturally able to adapt any topology in $\mathscr{S}_{phy}$. The standard GCS algorithm can only create topologies equal to its initial surface structure. This limitation, however, has been removed, as discussed in section 3.1.4.

3.3.5 Parallelization

On first impression both algorithms offer great potential for parallelization. If two random samples at two very different positions in $\mathscr{S}$ are selected, their processing is entirely independent. The difference in GCS and GNG lies in the graphs they use. In case of GNG, it is a simpler data structure, which only represents vertices and the connections between them. A CPU based parallelization only needs to *lock* the connection and disconnection process, since this avoids vertices from being connected twice or incorrect deletions of connected vertices. Operations such

as moving winning or neighboring vertices, however, can still be performed by other *threads* even during graph manipulations.
When a thread iterates through the neighbors of a vertex, a neighbor might appear or disappear, but the data structure is still functioning correctly. The connections a vertex has to other vertices are implemented as a simple, singly linked list. If such a connection is created or deleted, this can be performed by changing the content of one single pointer. The operation is therefore *atomic*, meaning it cannot be interrupted by an operation of another thread.
Additionally, the k-d tree needs to be modified for parallel use and instead of the tumble-tree the signal counters have to be stored in a *hash-map*, which is easy to parallelize.
The GCS approach in the presented implementation uses a more complex data structure – the *half-edge* data structure. If this data structure is altered, it cannot be accessed at the same time. Here, changes of the mesh cannot be designed to be atomic without additional locking techniques. When a surface area is processed by a thread, it always needs to be locked before any operation can be performed.
None of the created experimental implementations of parallel GCS algorithms achieved any increase in runtime. This was mostly due to the locking process to shield working areas from being accessed by other threads. Most operations performed in these working areas are drastically faster performable than the locking process. Note that these considerations already assume that changes to the surface can be performed in the basic algorithm step.

3.3.6 Modifications

Both methods [DRADLN10, MHH08] introduced a mechanism to decrease the rate in which vertices are added. In [MHH08], with increasing size of the network $\mathcal{S}$ the number of basic steps before adding a vertex is increased. In [DRADLN10], after the demanded size for $\mathcal{S}$ is reached another learning phase is introduced, where only the basic step is performed. The reason for these operations is that the basic vertex adding step creates holes because it creates vertices that have only two connections.
A different approach to tackle this problem is presented in [dRAdLN07], where the basic adaptation step and the adding step are modified. Three instead of two closest points to a random sample are searched for. This is supposed to promote the creation of triangles rather than arbitrary dimensional structures. The adding of vertices is performed by a vertex split operation as in the GCS method, creating a *hybrid* approach. To enable this operation, $\mathcal{S}$ has to be some form of a triangular mesh. The problem with this concept is that the connections that are made in the basic step are still arbitrary. Therefore, the approach still creates cross connections and overlapping triangles, which have to be cleaned up in a post-process.

3.4 Results

k-Means Clustering: The presented approach is stochastic and the same input data can produce different results, since $\mathcal{P}$ is accessed in a random series. The input data $\mathcal{P}$ is only accessed one sample at a time and never has to be loaded or processed entirely. The approach can therefore process an infinite number of procedurally generated data points from an analytically unknown function or as many measured samples as a modern hard drive can carry. However, the desired number of vertices and the spatial subdivision data structure – in which these vertices are stored and organized – need to be held in main memory in order for the process to run efficiently.
The process is very robust when dealing with noise and outliers and can even be used as sample distortion filter.

With a spatial subdivision data structure, such as an octree, the search for the winner – given an average vertex distribution – has the runtime complexity of $O(\log\ n)$. Leading to a process runtime complexity of $O(n\ \log\ n)$, given the number of loops relates to number of vertices n. This can be assumed, since any single vertex needs a more or less constant number of updates until it represents its surrounding samples appropriately.
Within the adaptation process vertices might by moved into sample free regions. Before they move out of these regions, other vertices might move toward their surrounding samples. Thus, if these surrounding samples are randomly chosen, these vertices are winners instead of the vertices in the sample free regions. Such vertices – shielded from being winners – are called "dead units". They do not contribute meaningful information to the processes result. Dead units are the result of the *winner-take-all* concept of the hard competitive learning paradigm.
Self-Organizing Map: Due to the *winner-take-most* concept in the SOM approach, the problem of *dead units* does not occur, since vertices in isolated sample free regions are still moved as neighboring vertices.
As the SOM produces a mesh $\mathscr{M}$ with a topology, it can and has been used for surface reconstruction [BF02, Yu99, HV98]. Its strengths are the inherited robustness toward noise and outliers from the k-means clustering approach and its ability to deal with non-uniform sample densities and holes. The SOM drags the initially existent surface area toward samples, instead of using some *sample-to-sample* distance assumption to recognize the surface area. Within that process the number of samples dragging a surface area into place does not matter, therefore non-uniform sample densities do not cause problems. When the surface area around a hole is pulled into place, holes are likely to be covered, since the surface area is simply dragged over it.
The SOM uses a static mesh connectivity for parameterization purposes. This static mesh connectivity limits its usability in surface reconstruction. It also leads to a static surface topology, the one of a bounded plane. As it cannot be changed, the initial topology of $\mathscr{S}$ needs to coincide with the topology of $\mathscr{S}_{phy}$ to achieve a fitting reconstruction. With the static connectivity the process can additionally not adapt the resolution of $\mathscr{M}$. Thus, if the surface in its spatial distribution does not coincide with the one of a rectangle, irregular and distorted vertex distributions are created. Given that the initial surface coincides in topology and spatial distribution with $\mathscr{S}_{phy}$, the process can still end up in a local minimum. Here, a state of falsely fitted surface is further optimized to represent $\mathscr{S}_{phy}$, but the topologically false surface construction cannot be resolved anymore.
The outcome of the process very much depends on its initialization, making repeated executions necessary. Also, the appropriate mesh resolution to represent $\mathscr{S}_{phy}$ needs to be known in advance. In the presence of noise and complex shaped geometry, this is hardly possible, again making repeated executions necessary.
Growing Cell Structures: The most basic surface reconstruction problem is deriving implications about an unknown surface $\mathscr{S}_{phy}$ from $\mathscr{P}$, which is only accessible by 3D search queries. Therein lies an important distinction between a SOM and GCS. While the SOM initializes all vertices at the beginning and then successively corrects their placement (warping strategy), the GCS algorithm starts with a simple mesh and initializes new vertices to refine a current surface estimate (refinement strategy). While the result of the SOM is created at the end of the process, the current surface estimate of the GCS algorithms always represents a result for the approximation of $\mathscr{S}_{phy}$. The process is very flexible, since it can be stopped and resumed at any time whether it is sufficiently accurate or not. Note that a balloon surface reconstruction process (see section 2.4.2.3) also initializes new vertices within the process, but fills up a space confined by $\mathscr{P}$. An actual reconstruction result again is only created at the end of the process when that space is fully filled.
Whenever the *information* (samples in $\mathscr{P}$) about the unknown surface $\mathscr{S}_{phy}$ is accessed, it is set in relation to a pre-existing surface estimate $\mathscr{S}$. This makes the GCS approach very powerful

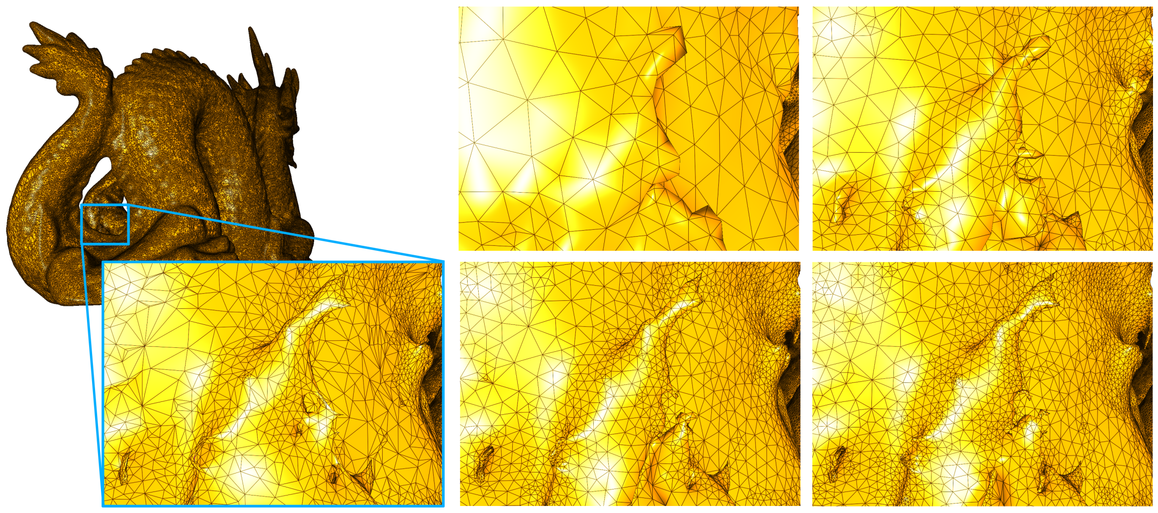


Figure 3.15: The strength of a refinement surface reconstruction approach. A challenging region at the Dragon's leg (left) and different stages of the GCS reconstruction process (right).

when avoiding local failures caused by ambiguous point constellations. In GCS, a newly created surface is built upon former surface stages, giving the surface a certain inertia when changed. In Fig. 3.15, this strength is demonstrated.
While $\mathscr{M}$ can be changed in resolution to specifically fit the spatial distribution of a surface, unlike the SOM, $\mathscr{M}$ cannot be changed in its topology. The topology of $\mathscr{M}$ is defined by the initial surface structure. In case of a tetrahedron, it is the topology of a sphere. Even if the initial topology coincides with the one of $\mathscr{S}_{phy}$ the process can still end up in a local minimum. It is, however, less likely to happen as for the SOM, since vertices are initially placed more sensibly. Changing the topology of $\mathscr{M}$ within the process would require operations, which can open the surface and seal it again.
As the presented algorithm adds vertices dependent on a local signal count the adding process automatically adapts to non-uniform sample densities (see Fig. 3.16).

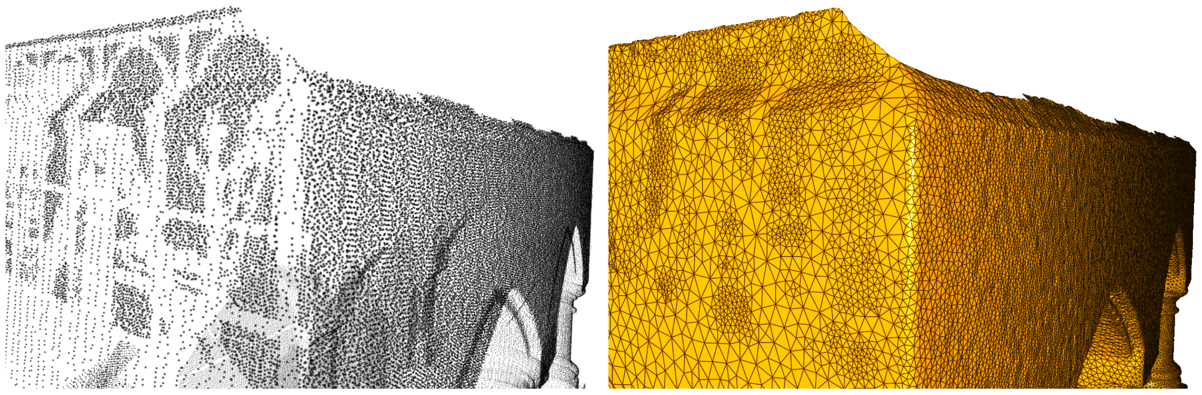


Figure 3.16: Non-uniform sample densities (left) only lead to different sized triangles (right) in a GCS reconstruction.

Although the GCS algorithm is built upon two previous algorithm stages it still involves only a simple set of instructions. The algorithm solves complex problems in reconstruction, not by a variety of specific instructions for any situation, but as a result of emergence. Here, units themselves just follow simple rules, but they create complex behavior as a result of their interaction. This simplicity makes such an algorithm easy to maintain. Also, the rules applied can be extended or changed easily. In Fig. 3.17, the reconstruction of a scanned tree is shown. The algorithm produces a suitable result even in this complex case.

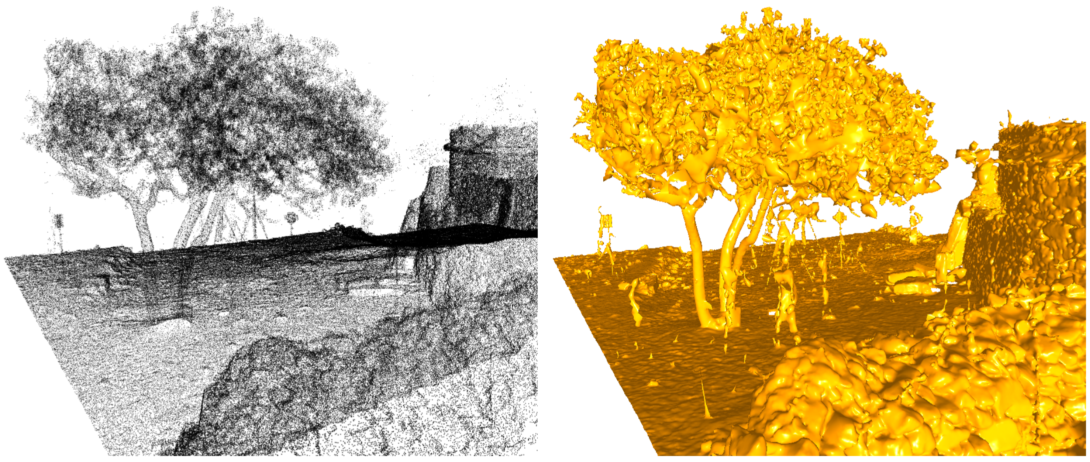


Figure 3.17: Without any explicit operations added to the GCS algorithm a tree in a scanned environment is suitably reconstructed (right), although the point cloud (left) appears to be chaotic.

For any creation of a vertex – belonging to the final n vertices – a constant number of basic steps has to be performed. Since this number is constant, the overall number of basic steps performed has a linear relation to n. Using the k-d tree, the search for the winning vertex – given an average vertex distribution – has the runtime complexity of $O(\log\ n)$. When using the tumble-tree, the additional signal counter operations can be performed with a runtime complexity of $O(\log\ n)$ as well. This creates an overall runtime complexity of $O(n\ \log\ n)$.

Growing Cell Structures vs. Growing Neural Gas: The data structures used for the GCS process can equally be used for the GNG approach, also leading to an average case runtime complexity of $O(n\ \log\ n)$. The GNG edge aging process has constant complexity $O(c)$ in both suggested implementations, given the edges have a *pointer* to their list position in the second implementation. However, the GNG algorithm has a slightly smaller constant, since the simple graph structures used by the algorithm make some of its operations less costly in runtime.

The presented comparison of GCS and GNG focused on aspects and tests concerning their working principles. In the following, their surface reconstruction abilities with different parameter settings are presented (see Table 3.1). For comparability purposes the Stanford Dragon is reconstructed with a fixed mesh resolution of 60K triangles. The model has non-uniform sample densities, some detailed areas such as the horns, and a relatively challenging overall shape.

The applied GCS approach already uses a mechanism to change its otherwise static topology (see section 3.1.4). To avoid holes from having a negative effect on the distance results of the GNG process, distance values are measured differently here. The sample-to-surface distance is measured as the average distance of all samples, which are closest to a triangle centroid. Thus, the considered samples are likely to lie above and close to existing triangles.

The benchmark versions (see Fig. 3.18) use the following parameter settings:

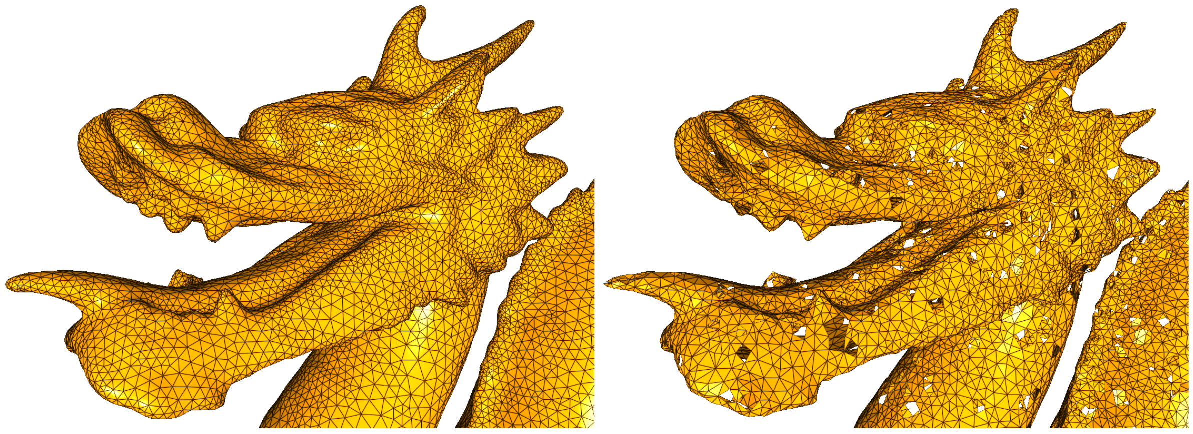


Figure 3.18: The Stanford Dragon reconstructed with GCS (left) and GNG (right) with the benchmark settings.

Parameter Settings:

Symbol	GCS	GNG	Meaning
lr	0.1	0.08	Learning rate
lrn	0.08	0.02	Neighbor learning rate
c_{add}	100	100	Basic step count before a vertex is added
c_{del}	500	500	Basic step count before a vertex is removed
γ	7	/	Allowed misses before deletion theshold
min_{sc}	0.3	/	Deletion theshold
max_a	/	88	Maximum edge age before deletion

In the performed tests, the GCS approach is less runtime efficient due to its more sophisticated underlying data structure. The Laplacian smoothing operation requires more runtime, but has very positive effects on the triangle quality. It also improves sample-to-surfaces distances. This operation cannot be used for GNG, since the operation uses triangle normals. The task specific k-d tree led to 30% gain in runtime efficiency in comparison to an octree. This proves the efficiency of the created data structure. It additionally shows the importance of the nearest neighbor search and the rearrangement of vertices concerning the overall runtime.
When lowering max_a in the GNG process, triangle quality can be gained in exchange for more deletions and thereby more holes in the surface. When increasing c_{add}, runtime can be exchanged for an overall better approximation in triangle quality and sample-to-surface distances. A reasonable tradeoff appeared at a setting for c_{add} of 200.
Smart Growing Cells: The SGC approach represents a very general solution for a variety of problems. Since it only involves *local* units – the vertices – rather than the *global* processing of significant amounts of mesh data, it is efficient and smoothly integrates into the *local* refinement approach.
The ACO process uses a local property – the vertex valence – to identify surfaces to cut. Also, the presented coalescing process only locally connects vertices. In [IJL$^+$04] in contrast, separate processes are introduced to cut and coalesce surfaces. These separate processes make the approach less flexible. The global triangle size criterion creates a demand for generally equally sized triangles, causing problems with non-uniform sample densities.
A disadvantage of locally performed cutting and coalescing operations is the inability to guarantee, for instance, solid objects. This would demand a comprehensive perspective on the mesh situation.

Parameter & Setting	time	dist	equilaterality	valence$[5;6;7]$
GCS				
Benchmark	0:0:17	3.57	78.5%	90.0%
Different c_{add}				
200	0:0:35	3.41	78.7%	90.6%
400	0:1:16	3.28	78.9%	91.3%
2000	0:6:55	3.16	79.0%	92.4%
octree instead of k-d tree	0:0:22	3.57	78.5%	90.0%
Standard Neighbor Movement (instead of Laplacian smoothing)	0:0:13	4.99	61.0%	86.1%
Different # of smoothing rounds				
5	0:0:40	3.67	78.6%	90.5%
20	0:3:54	3.69	79.0%	90.7%
GNG				
Benchmark	0:0:7	7.67	63.6%	67.9%
Different c_{add}				
200	0:0:12	7.41	67.5%	80.2%
400	0:0:25	7.28	70.5%	87.3%
2000	0:1:53	7.15	75.0%	94.4%
Different # of Threads				
2	0:0:4	7.72	63.0%	67.2%
4	0:0:3	7.70	62.6%	66.7%
Different max_a				
44	0:0:7	7.25	68.5%	79.7%
176	0:0:6	11.30	45.9%	31.8%

Table 3.1: Reconstruction of Stanford Dragon (# samples 438K; # triangles 60K) with different parameter settings for the GCS and GNG algorithm.

The curvature dependent placement of vertices makes the model $\mathscr{M}$ more memory efficient and leads to a faster adaptation of $\mathscr{S}$ toward $\mathscr{S}_{phy}$. Also, the process is less vulnerable to holes or non-uniform sample densities, since surface geometry additionally influences the placement of vertices. By integrating the curvature dependent subdivisions into the signal counter process, the algorithm does not need any additional data structures. In [JIS03], additional normal movement counters and data structures to organize them are added.
With the SGC concept special behaviors such as constraint movements for vertices at sharp features or boundaries can be created. Combined with the algorithm abilities of handling open surfaces, noise, outliers, non-uniform sample densities, and ambiguous sample constellations, the algorithm is very suitable for the creation of virtual environments, as presented in [AB11]. The creation of virtual environments is a very demanding reconstruction task. Special precautions, which are usually taken for certain materials or complex surface shapes, are impractical when scanning large scenes.

3.5 Conclusion

k-Means Clustering: The approach is efficient, robust against noise and outliers, and capable of processing point clouds of any size, which are produced in a realistic scanning scenario. For surface reconstruction purposes the algorithm would additionally need to determine the underlying topology which $\mathscr{P}$ originates from.

The occurrence of dead units and the overall quality of the produced clustering highly depends on the initialization of the process. Thus, to increase the likelihood of a high quality clustering, the process needs to be executed multiple times.

Self-Organizing Map: With the introduction of topology to the process, the SOM can be used as a reconstruction algorithm. Since the surface is “dragged” into the samples, incomplete data such as non-uniform sample densities and holes can be compensated. Its worst limitation is its static mesh connectivity.

Growing Cell Structures: In this section, the reconstruction strategy which all following algorithm developments are based on – iterative surface refinement – has been presented. In the refinement concept, every stage of $\mathscr{M}$ is an actual estimate of $\mathscr{S}_{phy}$ creating the flexibility to use the algorithm as a remeshing or LOD approach. This also establishes robustness when dealing with ambiguous point constellations.

While k-means clustering and the SOM change their learning rate throughout the process, the GCS algorithm uses a constant learning rate. This makes it suitable to be used as an online process, where samples are added while the process is running. Since the learning rate is unchanged, these new samples have the same chance to be learned as old ones.

Since the GCS approach can arbitrarily change the local resolution of $\mathscr{M}$ the guaranteed parameterization of the SOM is lost. The most significant limitation of the process, however, is its inability to leave the initial mesh topology. Vertex split and edge collapse change the meshes connectivity, but not its topology. In order to do so, operations to cut and reconnect the surface would be required.

The presented approach uses signal counters creating a *likelihood distribution*. It aims to create equal likelihoods for all vertices to be winners. A *distance minimization* on the other hand would aim to minimize the distances between vertices and surrounding samples. This topic is further discussed in chapter 6 where different approximation techniques for GCS are presented.

Growing Cell Structures vs. Growing Neural Gas: Both GCS and GNG are potentially suitable for surface reconstruction. They are resistant to noise and non-uniform sample densities and they are virtually independent of sample set sizes. Both algorithms can be implemented with a relatively low average case runtime complexity of $O(n \log n)$. The GNG approach is better suited for parallelization and can adapt its graph to models of arbitrary topologies. But GNG needs inherently more points to reconstruct an approximation of the same resolution in comparison to the GCS approach. Also, GNG needs a cleaning phase that can only be done in a post-process.

The progressively evolving surface represented as an actual triangulated mesh in the GCS approach is its major advantage over the GNG approach. The presented Laplacian smoothing, for instance, accesses the surface triangles to calculate surface normals, and most of the upcoming developments of the algorithm access information provided by the exiting surface. The upcoming redesign of the GCS concept in chapter 6 even extends the availability of surface related information.

To achieve this potential with the GNG approach makes an explicit surface model necessary. This is possible by creating a hybrid approach [dRAdLN07]. However, it is questionable if such an approach actually still qualifies as a GNG approach, since the freedom to build arbitrary simplices is discarded. This, however, conflicts with the basic idea of a *neural gas*. An alternative would be to build an additional implicit surface model on top of the GNG graph. In [Fri96], a RBF model has been suggested, but no reconstructions approach has been presented so far.

Smart Growing Cells: The enhancement of the concept behind the ACO process enable a generally more flexible algorithm implementation. Here, every created structure – not only the ones created in the deletion process – could be identified and then edited or removed. This development is discussed in detail in chapter 4.

Although *locally* the GCS approach creates soundly oriented surfaces, it can *globally* fail to create a consistent surface orientation. This problem is strongly connected to the *local* working principle of a refinement strategy. The problem itself and its solution are discussed in chapter 5. The presented mechanisms to deal with sharp features, boundary vertices, curvature and topology can be set into a broader context. These mechanisms only compensate shortcomings caused by a vertex focused model in the GCS approach. Here, the algorithm focuses on creating a distribution of vertices. In chapter 6, a new learning scheme using a surface focused model is presented, which eliminates these shortcomings at the conceptual level.

Chapter 4

The Filter Chain Concept

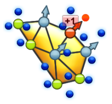
The filter chain concept (see Fig. 4.1 and Alg. 7) introduces a generalized method to individualize and precisely model the GCS algorithm behavior. The ACO process involves finding undesired structures in the mesh after the cutting process. The filter chain concept is inspired by this process and extends it to arbitrary structures that should be removed or edited. When different chains of filters are applied at different places of the algorithm, the modifications to the surface can be controlled very precisely.

4.1 Introduction

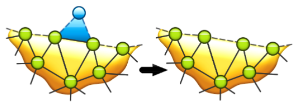
Reconstruction from unorganized points involves many different problem cases, which can even be ambiguous, *i.e.*, a sample constellation could be interpreted in different ways. Outliers in a broader sense, such as *ghost geometry* (see section 2.2.5), are an example for such a constellation. These undesired structures are identified by their very low sample resolution. But a low resolution might also be caused by a material, which is hard to scan. Very thin and tiny structures (see section 2.2.6), hardly captured by a scanner, such as cables, are another example. Such samples might also be caused simply by outliers or strong noise.
For such cases, a reconstruction algorithm has to be editable to account for individual preferences. Subsequent mesh processing approaches might also have certain requirements on the meshes they process.
A problem specific for iterative refinement approaches are structures, which are likely to lead to undesired surface constructs in later stages of the refinement process. These should also be identified and removed or edited.
To account for these necessities in the editability of the algorithm behavior, a concept is needed that allows for precise changes on the one hand, but is general enough to allow for easy adding and removing of such changes on the other hand.

Algorithm 7 Filter Chain Concept (based on SGC Alg. 5)

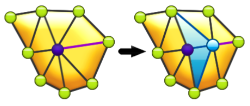
```
 1: Initialization
 2: repeat
 3:     repeat
 4:         repeat
                Basic Step
```



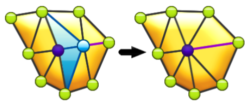
```
 5:             Select random sample $\mathbf{p}_x$ of $\mathscr{P}$
 6:             Find the winning vertex $\mathbf{v}_x$ that exposes the smallest Euclidian distance to $\mathbf{p}_x$
 7:             Discontinuity Adaption
 8:             Curvature Adaption
 9:             Decrease signal counters of all other vertices by a fraction:
                $\forall sc_i(i \in \mathscr{M})\ \Delta sc_i = -\beta \cdot sc_i$
10:             Coalescing
                Filter Chain
```



```
11:             repeat
12:                 Select a vertex $\mathbf{v}_x$ from the vertices to filter $\mathscr{V}_{filt}$
13:                 Expose $\mathbf{v}_x$ to all filters in $\mathbf{F}\triangleright_{filt}$
14:             until $\mathscr{V}_{filt}$ contains no more vertices: $|\mathscr{V}_{filt}| = 0$
15:             Increment the iteration counter: $\Delta t = 1$
16:         until The basic step has been performed $c_{add}$ times: $t$ mod $c_{add} = 0$
            Vertex Split
```



```
17:         Select vertex $\mathbf{v}_x$ with the highest signal counter value: $\forall sc_i(i \in \mathscr{M}) <= sc_x$
18:         Select the longest edge of $\mathbf{v}_x$ and perform a vertex split operation
19:         Split $sc_x$ between $\mathbf{v}_x$ and the new vertex $\mathbf{v}_{new}$: $sc_{new} = sc_x/2 \wedge sc_x = sc_x/2$
20:         Filter Chain
21:     until The basic step has been performed $c_{del}$ times: $t$ mod $c_{del} = 0$
        Edge collapse
```



```
22:     Select vertex $\mathbf{v}_x$ with the lowest signal counter value $\forall sc_i(i \in \mathscr{M}) >= sc_x$
23:     if Signal counter value $sc_x$ is too low: $sc_x < min_{sc}$ then
24:         Perform an edge collapse operation on the edge leading to valences closest to six.
25:     end if
26:     Filter Chain
27: until A certain number of vertices is reached: $|\mathscr{M}| >= n_{final}$
28: Add all vertices in $\mathscr{M}$ into $\mathscr{V}_{filt}$
29: Filter Chain
```

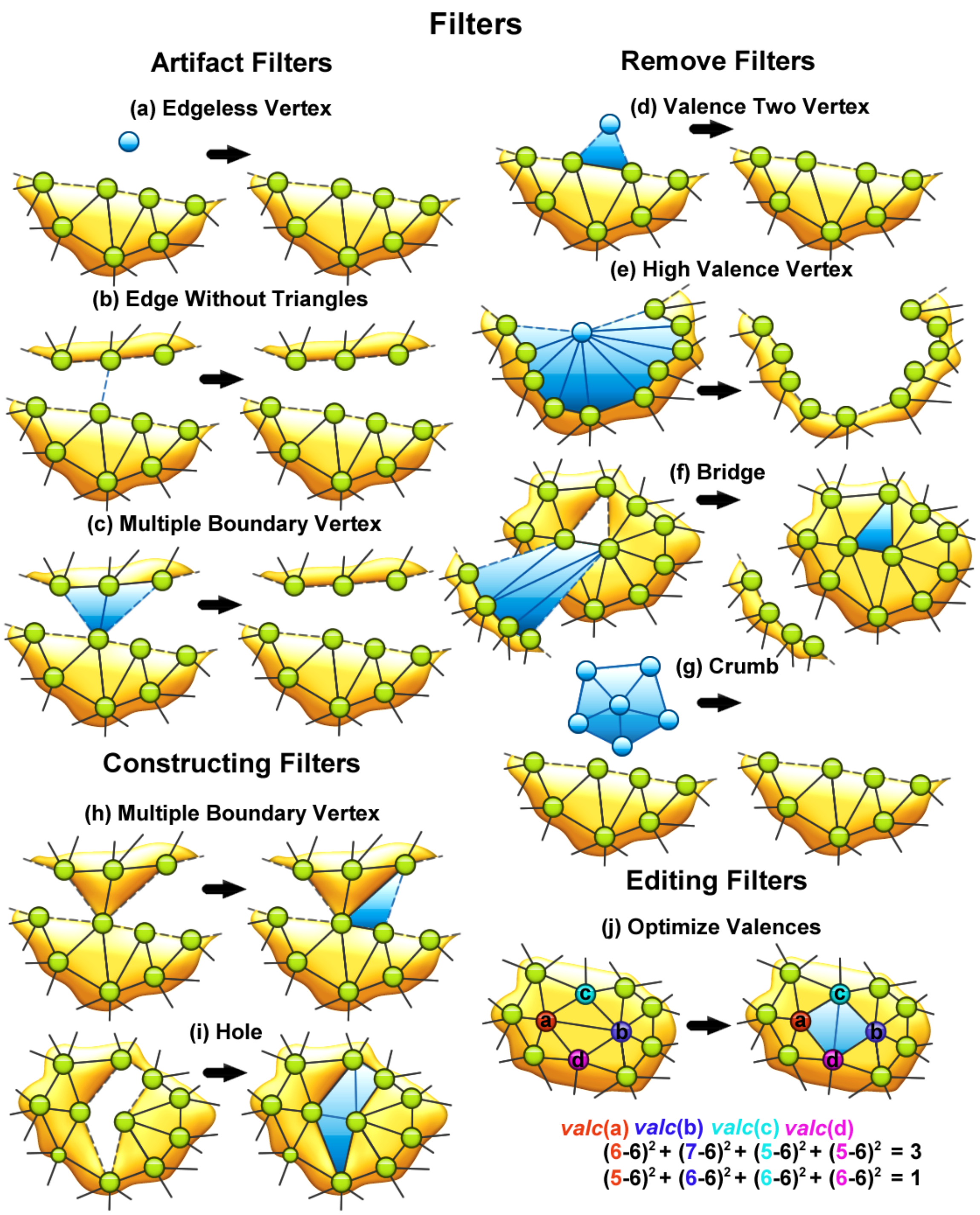


Figure 4.1: A broad selection of artifact, removal, construction, and editing filters. The illustrations show mesh structures before (left) and after (right) applying a certain filter.

4.2 Approach

The aim of the filter chain concept is to gain more control over the results of every single editing step of the mesh. When vertices are changed in the basic step, the vertex split or the edge collapse operation adds them into a set of vertices $\mathscr{V}_{filt}$. These vertices are then one by one taken out of that set (see line 12 in Alg. 7) and exposed to a singly linked list of filters $\mathbf{F}\triangleright_{filt}$ (see line 13 in Alg. 7). $\mathbf{F}\triangleright_{filt}$ may contain different filters depending on where in the algorithm it is used. If a vertex of $\mathscr{V}_{filt}$ is filtered and it passes all filters in $\mathbf{F}\triangleright_{filt}$, it is deleted from $\mathscr{V}_{filt}$. If one of the

filters in $\mathbf{F}\triangleright_{filt}$ is activated, vertices might be edited and therefore added to $\mathcal{V}_{filt}$ or deleted and therefore removed from $\mathcal{V}_{filt}$. If all vertices in $\mathcal{V}_{filt}$ have passed the filters in $\mathbf{F}\triangleright_{filt}$, the process ends (see line 14 in Alg. 7).
The key in the design of the algorithm behavior lies in the choice of filters added to the different filter lists. Although the order of the filters in $\mathbf{F}\triangleright_{filt}$ makes a difference, it is not considered an actual algorithm behavior design choice. Generally, simple filters, *e.g.*, a filter looking for edges without faces connected to them, come first and more complex ones, *e.g.*, a filter looking for tiny surface segments, are placed later in the list. This order leads to editing more general problems which might render the execution of more complex filter operations futile first. In the following, a selection of important filters is presented.

4.2.1 Artifact Filters

The following filters are typically used as and after cutting processes, where parts of the surface such as vertices, edges, and faces are removed. The structures they search for and remove are undesired in most mesh processing algorithms. These structures mostly were searched for in the initial ACO process as well.
Edgeless Vertex: If a vertex has no edges and is therefore not connected to any triangle, it is in fact an artifact in a triangle based surface representation. It is also worthless for the GCS process and is therefore removed (see *(a)* in Fig. 4.1). Note that in order to represent point clouds with a mesh file format, these unconnected points (vertices) are accepted in many implementations.
Edge Without Triangles: An edge which has no triangles connected alongside it, is also an artifact. Such a structure can only occur in a mesh data structure explicitly representing edges, such as a *half-edge* data structure. In triangle based file formats, however, they cannot be represented. They are also not needed in the GCS process and hence they are removed (see *(b)* in Fig. 4.1).
Multiple Boundary Vertex: A vertex can be connected to multiple boundaries and then have more than two boundary edges (see *(c)* in Fig. 4.1). For most mesh processing applications such vertices are not valid. This mesh structure can be fixed by removing all triangles and edges of a vertex, except those belonging to the biggest “wedge” (see *(c)* in Fig. 4.1).

4.2.2 Removal Filters

While artifact filters remove structures that are invalid in a mesh, the following filters remove structures that are generally valid, but have proven to either lead to bad results in the course of the refinement process or are generally undesirable in a resulting mesh.
Valence Two Vertex: A boundary vertex can have a valence of two (see *(d)* in Fig. 4.1). Such a vertex is very unlikely to support a correct reconstruction process. It is more likely to cause twists and malformed mesh regions if involved in a coalescing process, and is therefore deleted.
High Valence Vertex: A vertex of a high valence (see section 3.1.4.2 on the ACO process) is a good indicator for topologically misplaced surfaces. It can be cut out of the mesh (see *(e)* in Fig. 4.1). This filter, as in the ACO process, might cause a series of deletions, since the remaining vertices are once again added to the filter process.
Bridge: When close surfaces are differentiated by the refinement process, former falsely built connections between those surfaces are constantly diminished, until they are completely deleted. If only a hole is left from a former connection, it is sealed by the coalescing process. However, if a connection remains between two surfaces, the remaining hole cannot be sealed. In such a case, all sealing operations can be performed, until reaching a hole size of only three vertices (one missing triangle), which has the additional connection to another surface (see *(f)* in Fig. 4.1).

This connection in form of a one edge broad “triangle fan” needs to be deleted in order to close the remaining hole. If the hole is seen as a “gate”, this fan is its “bridge”. A bridge is very likely to be part of a misplaced surface and is therefore deleted. This speeds up the differentiation process between close surface regions.
Bottleneck: The bottleneck filter is a generalization of the bridge filter. Every bridge is also a bottleneck and thereby the filter is a more aggressive version, tackling the same problem. While the bridge filter focuses on a one edge broad connection to a triangle sized hole, the bottleneck filter focuses on every pathway in the mesh of triangle size (see Fig. 4.2). Three vertices are the minimum perimeter for a pathway in a triangular mesh – being a *2-manifold* – and therefore always resemble a “bottleneck”. Whenever such a pathway is found the number of triangles on both sides is counted and the ones that are of the lower count are deleted. The filter is efficient if a reconstruction is known to contain many close surfaces but no thin structures, which this filter might hinder in their construction.

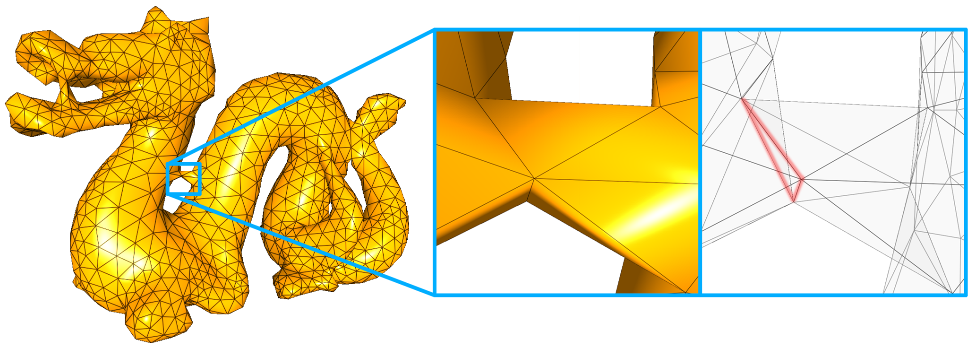


Figure 4.2: An early stage in the reconstruction of the Dragon model. Neck and back of the Dragon are falsely connected. The connection is already minimized to a bottleneck, which is a pathway in the mesh of three vertices perimeter.

Triangle Size: The triangle size filter locates and deletes triangles of above-average size. The filter needs a setting of how far the average triangle size is allowed to be exceeded by a triangle, before it is considered to be “too big”. In cases of distorted samples, where huge triangles are caused by outliers or constructed due to ghost geometries, it is very helpful to clean up the mesh. Since the filter cannot be used in combination with a hole closing filter (see below) it is best used after the main loop (see line 29 in Alg. 7).
Crumb: A *crumb* is a mesh structure of a certain size of vertices unconnected to the rest of the surface. This size can reach from one to a few thousand vertices. If this structure is identified, it is deleted (see *(g)* in Fig. 4.1). Crumbs are often caused by noise, outliers, and complex shaped surface regions, which are not yet correctly recognized. As they are less stabilized during the refinement process by surrounding surfaces, crumbs can cause twisted and degenerated surface regions. The crumb filter is effective against these effects, but can also be used after the main loop to delete ghost geometry.

4.2.3 Editing and Constructing Filters

All filters presented so far solved the occurrence of undesired mesh structures by their removal. This limits the process to finding undesired structures and implies that after their deletion the GCS process creates desired structures. But filters can also be used to edit and construct mesh structures.
Multiple Boundary Vertex: Multiple boundary vertices, presented above in section 4.2.1 on artifacts, can also be fixed constructively. In this solution, triangles are added until a vertex is only connected to one boundary instead of multiple (see *(h)* in Fig. 4.1).

Hole: Holes of sizes between three to five vertices in perimeter are rarely desired. The GCS process is thereby speeded up by sealing them immediately, instead of waiting for them to be closed by the coalescing process. The GCS algorithm therefore should always include a hole closing filter (see *(i)* in Fig. 4.1). If the result of the GCS process is supposed to be a solid object and the point cloud is known to include multiple undesired holes, the perimeter can be set to higher values. Values up to twenty are sensible. Above that, however, the simple closing mechanism by "zigzagging" triangles might be insufficient and could create degenerated mesh structures.

Optimize Valences: The valence optimization filter searches for potential edge swap operations which optimize vertex valences. In order to perform an edge swap operation, an edge has to be connected to two triangles. An edge connects two vertices **a** and **b** (see *(j)* in Fig. 4.1). With these two vertices and the two opposite to the edge, two triangles are defined $(\mathbf{a},\mathbf{b},\mathbf{c})$ and $(\mathbf{a},\mathbf{d},\mathbf{b})$. If the edge between **a** and **b** is swapped, those triangles are transformed into $(\mathbf{a},\mathbf{d},\mathbf{c})$ and $(\mathbf{d},\mathbf{b},\mathbf{c})$. An edge is considered a swapping candidate (see *(j)* in Fig. 4.1) if the sum of the square distances to valence six of its triangle vertices can be lowered by an edge swap operation:

$$(valc(\mathbf{a})-6)^2+(valc(\mathbf{b})-6)^2+(valc(\mathbf{c})-6)^2+(valc(\mathbf{d})-6)^2 \tag{4.1}$$

Square distances are used to express the undesirability of extremely high or low valences. If all edges for a vertex are tested, the swapping candidate exposing the largest old to new valence difference (see equation 4.1) is chosen to perform an edge swap operation (see *(j)* in Fig. 4.1). If no candidate exists, none is chosen.

This optimization can easily lead to circular or alternating swapping operations, causing infinite filter loops. To avoid this, the filter needs to either not add modified vertices to $\mathscr{V}_{filt}$ or it needs to have a memory of performed edge swap operations to exclude those from consideration.

4.3 Additional Usage

Most of the presented filters remove or transform undesired mesh structures. This is not only reasonable within the GCS process, but also to clean arbitrary meshes form artifacts or distortions. The filter chain concept can therefore be used as a general mesh cleaning approach. An example of such a filtering process is shown in its use after the main loop (see line 28 in Alg. 7).

4.4 Results

With the filter concept the GCS algorithm can easily be edited to account for very different reconstruction scenarios. This can be achieved just by adding and removing filters from the different filter chains. In the following, some standard filter configurations are presented as well as some individual configurations to solve specific problem cases.

When cutting processes are used, all artifact filters should be enabled to guarantee a valid mesh. Valence two vertices and bridges have proven to slowdown the correct adaptation of $\mathscr{S}_{phy}$ and should therefore always be removed. The high valence filter should be added to the filter chain in the deletion section (see line 26 in Alg. 7) to cut out misplaced surface regions.

The valence optimization filter has proven to generally increase the mesh quality (see Fig. 4.3) and can be added to any filter chain. The optimization of the valences leads to higher triangle quality and also allows for improved sample-to-surface distances.

A hole filter, sealing all holes up to perimeters of five, is also a general improvement of the adaptation speed of the algorithm. If the finished mesh is expected to be a solid object, a higher

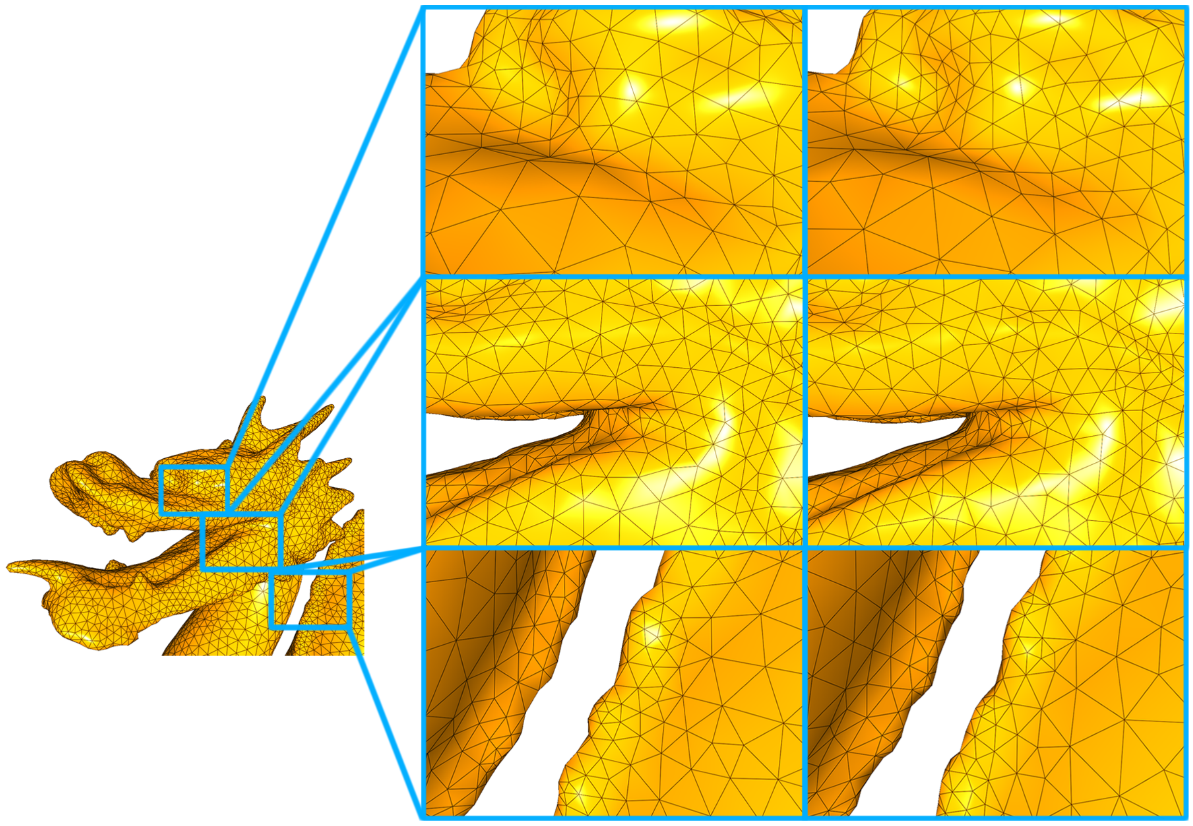


Figure 4.3: A reconstruction of the Stanford Dragon with 30K triangles with (left) and without (right) the valence optimization filter. Visually, the difference of better valences is hardly recognizable. When measured, the triangle quality (equilaterality) with the filter (81.19%) is clearly higher than without it (78.05%). Additionally, the sample-to-surface distance (dist) with the filter (5.20) is also lower than without it (5.27).

perimeter can be used. Since the hole filter is still a local operation, no perimeter setting can guarantee the resulting mesh to be a solid object.
The bottleneck, the triangle size, and the crumb filter are more specific filters, which can be used to deal with certain situations. If surfaces are close to each another, the bottleneck filter speeds up the differentiation process of those surfaces. It can, however, also inhibit the construction of thin structures.
Using editing and constructing filters can give rise to mutually dependent filters. This can lead to infinite filtering loops, where one filter repeatedly constructs the deletion-target of another filter. Such a dependency can be created, *e.g*, if the triangle size filter is used in combination with the constructive multiple boundary vertex filter or the hole filter. The latter filters create triangles, while the triangle size filter deletes them. Thus, it is reasonable to place it in the final filter chain to delete oversized triangles (see Fig. 4.4).
The crumb filter can also be used as a post-processing filter, as shown in Fig. 4.4. If, however, many crumb structures tend to disturb the refinement process the filter can also be used within the algorithm.

4.5 Conclusion

The filter chain concept allows for easy editability of the algorithm behavior. New behavior can easily be added and removed. The expressiveness of the concept is demonstrated by replacing the ACO process by a collection of filters. Additional modifications of the algorithm behavior have

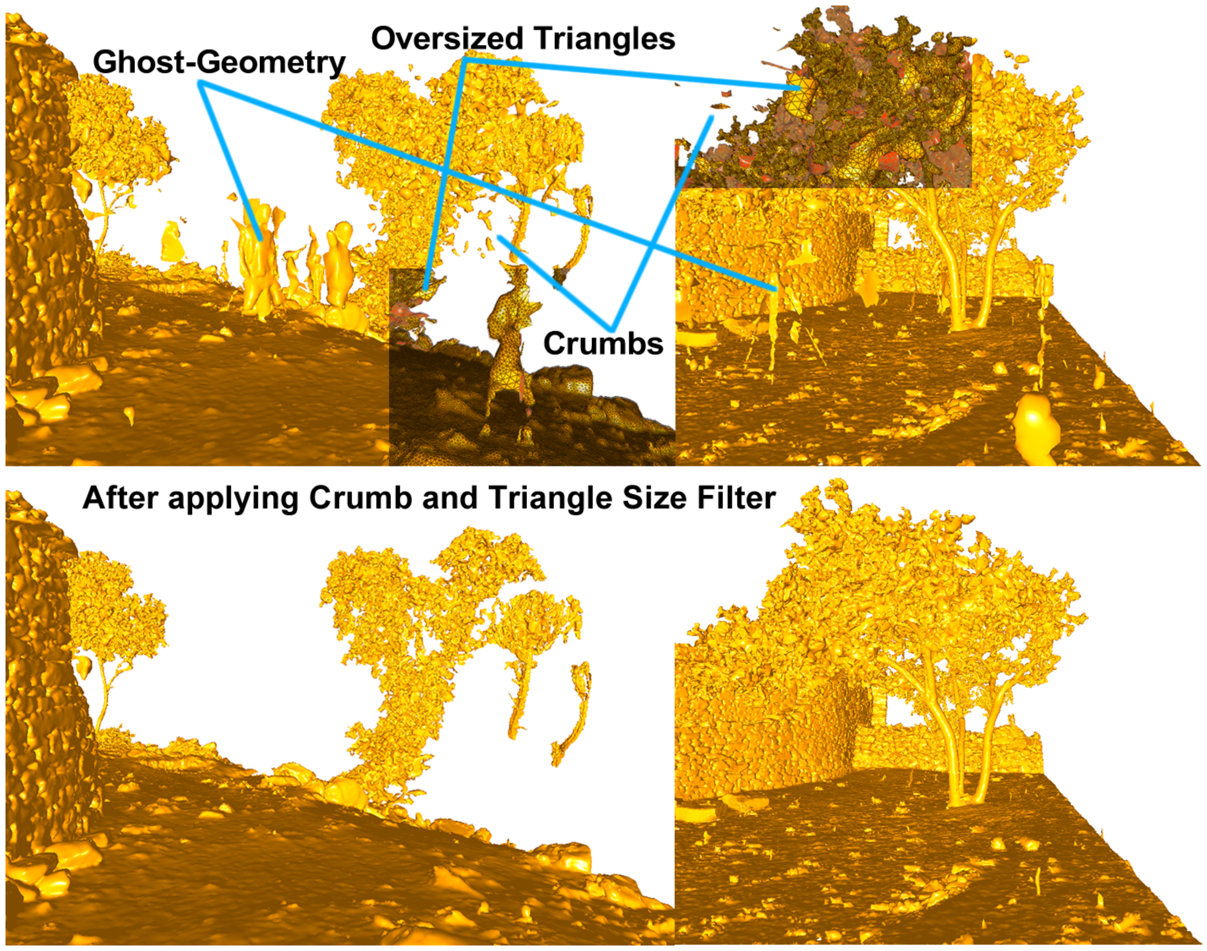


Figure 4.4: Reconstruction of an environment around a farm building. The two pictures at the top show the uncleaned mesh. It exposes many crumbs, due to disturbances in the scan. People and scanning equipment represent ghost geometries. Oversized triangles are exposed in ghost geometries due to outliers. The two bottom pictures show the same mesh after applying a triangle size (size limit: three times average triangle size) and a crumb (# vertices: 300) filter.

been experimentally demonstrated. Since cleaning filters can be added anywhere in the process, operations cutting the mesh can also be implemented wherever needed without additional effort. This adds flexibility when introducing new algorithm operations.

Filter combinations mutually invoking each other are a conceptual weakness. This problem can generally be addressed by executing certain filters always in separate chains or by preventing filters from executing the same operation twice.

The presented implementation uses a singly linked list to represent a filter chain. However, a filter hierarchy where a tree of filters is passed or the results of filters invoke adding additional filters might also be an interesting direction for the filter chain concept.

Filters are basically local mesh constellation modification operators like all SGC operations are. Thus, they are conceptually suitable to also express and replace other former SGC operations, such as Laplacian smoothing or the coalescing operation. Modeling the entire algorithm functionality as filters would elevate the local rule based SGC concept to a new level. Expressed as filters, rules could automatically be exchanged and tested. Possibly the rule set could even be adapted to the given input data, exploiting the editability of the filter chain concept to create automatically adjusting refinement processes.

Chapter 5

Solving Twisted Surface

For complex geometries and especially for non-solid objects, the GCS approach may fail to create a consistently orientated surface. The *twist solving process* [AB12b] (see Alg. 11) identifies and corrects such inconsistently oriented surface regions. To properly integrate into the GCS process, the solution needs to be efficient in runtime and memory consumption. The approach involves a novel data structure – the *edge-front* – which enables an efficient and at the same time compact solution.
To demonstrate the potential within the *edge-front* based processing scheme, it is applied to approximate *geodesic* distances.

5.1 Introduction

The problem of inconsistently oriented surface regions involves the emergence of such regions, their identification, and their correction. The problem affects the surface on a *global* level, while an iterative refinement approach uses a *local* optimization strategy. This creates an additional integration problem for a viable solution.
The edge-front based data structure used to solve this problem can also be applied to geodesic calculations.

5.1.1 Emergence of Twisted Surface

A refinement strategy uses a current surface estimate $\mathscr{S}$ of the surface under investigation $\mathscr{S}_{phy}$. $\mathscr{S}$ has an orientation and is constantly optimized in course of the refinement process. Iterative refinement approaches are very capable of creating sound orientated surfaces even when confronted with locally ambiguous point constellations. Other reconstruction approaches often fail in these cases, since they try to derive the surface progression directly from the samples. The GCS approach evolves new surfaces from former surface stages and thereby stabilizes the surface development process. $\mathscr{S}$ is always an attempt to approximate all of the data in $\mathscr{P}$ at any given algorithm stage.
Thus, in early stages some complex shaped geometry might be represented by a very crude and simplistic shape. The transition from that simple to a more sophisticated shape is performed as a series of local adaptations. Without any global overview or supervision, these local adaptations might produce results that are inconsistent on a global level. This can lead to surfaces ending up in a local minimum, which can only be left if cutting operations are introduced. It can also lead to inconsistencies in the orientation of surfaces.
A twisted surface region is created if one continuous surface has evolved from two different surface origins with two different orientations. Since these two surfaces have different orienta-

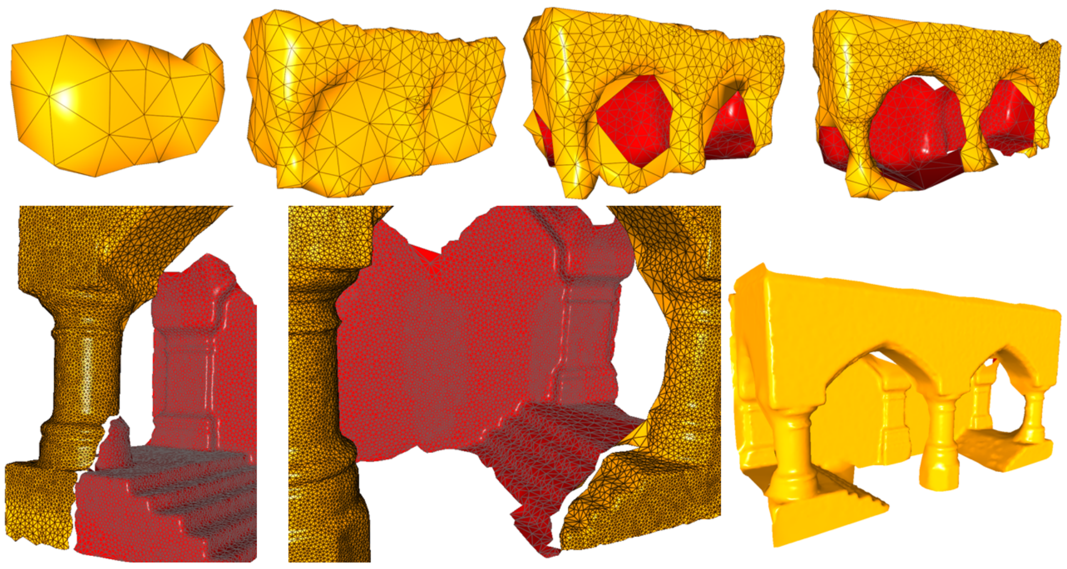


Figure 5.1: A series of reconstruction stages in the GCS approach creating an inconsistently oriented surface (top row). When further optimized, the differently oriented surfaces remain with a gap in between (bottom left and middle). The correct reconstruction of the Vault model (bottom right). The twist occurs, since the back and front of the Vault have the same orientation, but since the initial GCS estimate is box-shaped, it exposes different orientations for front and back.

tions, they cannot be coalesced, leaving a gap where the two surfaces meet. Typical sources of twists are: long surface structures, such as pipes, where surface areas are drawn inside out; small surface segments (see section 4.2.2 on crumbs) evolving without the stabilization from a larger surface area; and generally failed attempts in recognizing a correct surface orientation in an early algorithm stage. Independent of their origin twisted surface regions remain in $\mathscr{S}$ and are further optimized, but cannot be resolved by local refinement operations (see Fig. 5.1).

Twist Example: If a s-shaped plane is reconstructed (see *(a)* in Fig. 5.2), it might be recognized as being box-shaped at an early algorithm stage (see *(b)* in Fig. 5.2). This initial estimate assigns the same orientation to the ends of the plane, but a s-shaped plane exposes different orientations at its ends. When further refined, the actual shape of the surface becomes more evident (see *(c)* in Fig. 5.2). However, the different orientations remain and collide at some point (see *(d)* in Fig. 5.2). Due to different orientations, the surface regions cannot be joined by coalescing and remain twisted toward one another, while the surface is further refined and locally improves its approximation quality.

5.1.2 Solving a Global Problem on a Local Level

The surface refinement in the GCS approach is realized by *local* mesh optimization operations. Resolving twisted surface regions is a *global* problem, which possibly involves modifying up to half of the entire mesh surface. This additionally poses the challenge of finding a solution that sensibly integrates into the *locally* operating GCS scheme.

One simple solution to the orientation problem would be to create a first surface estimate with a different algorithm. An algorithm which involves a global orientation concept is, for instance, Hoppe's approach [HDD$^+$92], which uses globally oriented linear base functions (see section 2.4.3.1). This reconstruction result can then be used instead of the initial tetrahedron.

When using this result in the GCS algorithm, however, possible reconstruction mistakes from that first approach are incorporated. Even worse, instead of having one coherent algorithm the approach would become a patchwork algorithm. This would make it complex in maintenance and finding a correct parameter setting would become increasingly difficult. It would be especially

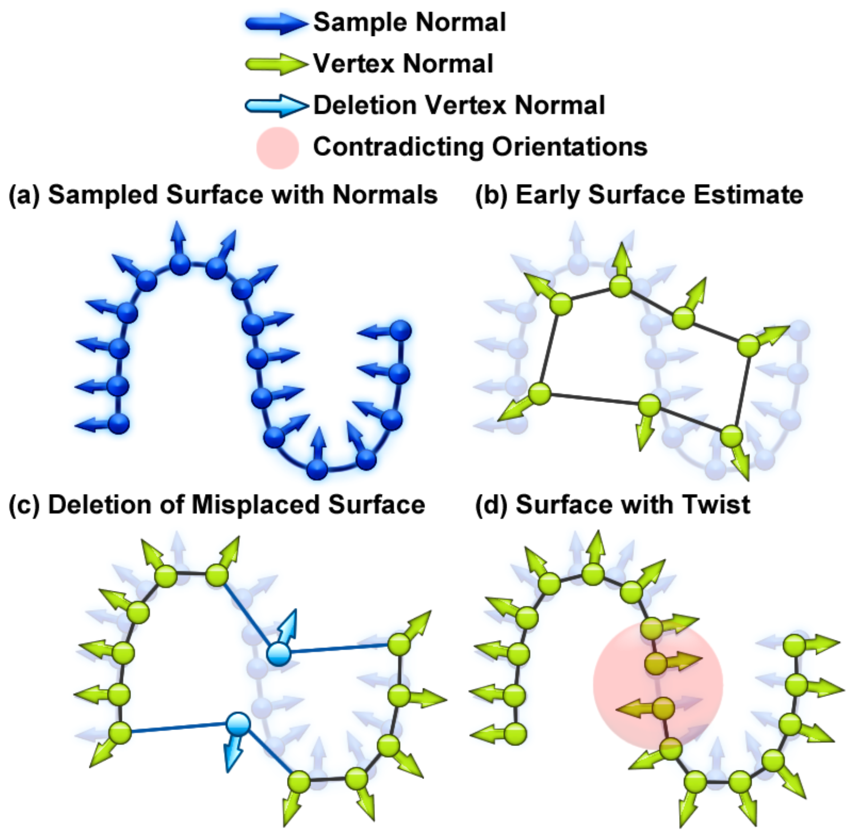

Sample Normal
Vertex Normal
Deletion Vertex Normal
Contradicting Orientations
(a) Sampled Surface with Normals
(b) Early Surface Estimate
(c) Deletion of Misplaced Surface
(d) Surface with Twist

Figure 5.2: Emergence of twisted surface: *(a)* a sampled s-shaped surface with normals; *(b)* an early reconstruction stage of those samples, which is box-shaped; *(c)* topologically misplaced surface is recognized and deleted. The false initial orientation estimate of the box-shape, however, remains; *(d)* the surface is further optimized, but the differently oriented surfaces cannot be connected and the surface exposes a twist.

challenging to choose the transition point between the two algorithms. Setting such a point implies that all aspects concerning orientation would not be critical from this point on.
In [ILL$^+$04], the GCS approach is used to create numerous low-resolution meshes of the same point cloud with different random series. Then the random series, which created the mesh closest to the average of those meshes is chosen for the actual surface reconstruction in high resolution. Again, two separate algorithm phases are created that require a predetermined transition point. A solution that is an integral component of the refinement process is more desirable.
Solving a twist can involve the processing of huge mesh segments, making a purely locally operating approach unfeasible. A global approach would analyze the orientation of the entire current surface estimate and then search for and fix inconsistent orientations. Such a process could be performed in intervals throughout the GCS process.
Such a solution, however, would still not integrate to create one consistent approach. It would again executed outside the actual GCS process, necessitating additional pre-processing data structures to analyze the orientation. This solution would also not scale to a current problem at hand, but always independently process the entire current surface estimate.
A *semi-local* solution on the other hand would only process surface segments actually involved in a twisted surface region. To enable such a semi-local solution, a data structure is needed that can efficiently process limited surface regions. During the approach, the mesh is constantly changing, therefore an operation cannot be based on global, pre-processed mesh data. To be able to virtually process regions of arbitrary size, the *working set* of such a data structure can only include the fraction of surface between processed and not yet processed surface.

5.1.3 Geodesic Distances

Making *on-surface* distances available for mesh analyzing, searching, and editing processes, has many application cases in geometry processing. For example, it can be used for *mesh parameterization* to define *texture coordinates* [ZKK02]. It is also used in *mesh segmentation*, where mesh *segments* are determined, which are then replaced by *B-spline* patches to create smooth surfaces [KL96]. When comparing the distribution of distances, the process can be used for *topology matching* [HSKK01]. In animation, a *mesh decomposition* is needed to distinguish surface *components* in the animation of a model, which again can be determined by analysing on-surface distance distribution [KT03]. When using *procedural textures* on complex meshes, geodesic coordinates can be used to determine a transfer function between the space of the procedural function and the topological space of the mesh [OTC$^+$10].
All these applications require a surface distance metric, which can be established with geodesic distances. Geodesic distance calculations can be divided into exact and approximated solutions. A first implemented exact solution [KO00] used a graph based search strategy on a *sequence tree*. The theory for this technique was suggested in [CH90]. The algorithm has quadratic runtime complexity. Current applications mostly use approximated solutions. The *fast marching method* (FMM) [Set95] approximates distances on triangular meshes [KS98]. In this approach, distances are calculated from a starting vertex. All vertices currently reachable are considered and the closest one is added to the "known" vertices. If a vertex is newly added, all vertices reachable from this new vertex are added to the "reachable" vertices. By always adding the closest vertex, the FMM works similar to the algorithm of Dijkstra. Although significantly more precise approximation methods exist, which are only marginally slower than the FMM, still they are the most widely use approaches due to their simple implementation. The accuracy of the approach strongly depends on the given triangulation, since the discrete calculation points are determined by the vertex distribution.
To achieve a higher independent of the given mesh triangulation and to increase accuracy, subdivision techniques were used, which subdivide the initial mesh by creating additional edges [LMS97]. In [KS00], the shortest path is first calculated on the original edges of the mesh with Dijkstra's algorithm and then the area of the resulting path is repeatedly subdivided. The search is then repeated, until a certain subdivision level is reached. A recent breakthrough in shortest path calculation has been achieved in [CWW12], whose approach has a close to linear runtime. Here, the problem is solved by transferring it into a *Poisson equation*.
In [MMP87], an exact solution was presented. Here, for a starting point, the mesh was subdivided to expose so called *windows*. These windows guarantee the possibility to compute an exact shortest path on the given mesh edges. When established, the path can be calculated with Dijkstra's algorithm. The suggested method has a worst-case runtime complexity slightly above quadratic. However, when first implemented [SSK$^+$05] it proved to be significantly more efficient in practice. Due to the intensive subdividing of the windows, the approach has a significantly higher memory consumption than other approaches. Based on the initial approach an approximate solution was presented in the same work, which reduced this problem. In [BK07], the approach was extended to additionally allow line segments as starting points. A more detailed overview of the subject of geodesic path calculation can be found in [BMSW11].

5.2 Approach

In the following, a method is presented which resolves twisted surface regions reliably and efficiently on a semi-local level and which properly integrates into the iterative refinement process. This solution can be compactly expressed due to a novel data structure based on an edge-front.

5.2.1 Semi-Local Processing

Semi-local processing involves surface computations of limited domain around a point of interest. Such computations require a defined on-surface starting point and having control over the expansion of the processed area. Additionally, the processing is supposed to create a small *memory footprint*, to enable the processing of vast surface areas.
First, the *vertex front* (VF) a pre-form of the *minimal edge front* (MEF) is presented. Then a definition followed by a detailed explanation of the MEF data structure is given. The *minimal distance front* (MDF) is presented which enhances the processing strategy to include geodesic distance calculations. Finally, based on those data structures, an efficient way of calculating minimal connecting paths between vertices is presented.

5.2.2 Vertex Front

The need to access vertex neighborhoods gave the inspiration for the VF. First degree neighbors can easily be accessed by iterating through all connected edges of a vertex. When, however, the first neighborhood of an edge, a triangle, or higher degrees of neighborhoods are accessed, an advanced concept is needed.

Algorithm 8 Vertex Front

```
 1: Clean all vertex containers: $\mathcal{V}_{front} = \mathcal{V}_{old} = \mathcal{V}_{new} = \{\}$
 2: Add starting point vertex/vertices to $\mathcal{V}_{front}$
 3: while Expansion level not reached AND front not empty: $n_{exp} > 0 \wedge |\mathcal{V}_{front}| > 0$ do
 4:    repeat
 5:       Select a vertex $\mathbf{v}_x$ from $\mathcal{V}_{front}$ that has not been processed so far
 6:       Get first degree neighborhood $\mathcal{N}_x$ of $\mathbf{v}_x$
 7:       repeat
 8:          Select a vertex $\mathbf{v}_y$ from $\mathcal{N}_x$ that has not been processed so far
 9:          if $\mathbf{v}_y$ is a new vertex: $\mathbf{v}_y \notin \mathcal{V}_{old} \wedge \mathbf{v}_y \notin \mathcal{V}_{front} \wedge \mathbf{v}_y \notin \mathcal{V}_{new}$ then
10:             Add $\mathbf{v}_y$ to new vertices: $\mathcal{V}_{new} = \mathcal{V}_{new} \cap \mathbf{v}_y$
11:          end if
12:       until All vertices in $\mathcal{N}_x$ have been processed
13:    until All vertices in $\mathcal{V}_{front}$ have been processed
14:    Swap containers: $\mathcal{V}_{old} = \mathcal{V}_{front} \wedge \mathcal{V}_{front} = \mathcal{V}_{new} \wedge \mathcal{V}_{new} = \{\}$
15:    Decrement expansions: $\Delta n_{exp} = -1$
16: end while
```

The VF is an expandable set of vertices (see Alg. 8). It is initialized with a set of vertices (see line 2 in Alg. 8), *e.g.*, the vertices of a triangle. These initial vertices can then be expanded. The VF includes three data containers for old $\mathcal{V}_{old}$, current $\mathcal{V}_{front}$ and new $\mathcal{V}_{new}$ vertices. These containers need to offer optimized operations to add, to find, and to iterate through vertices. In the presented implementation, a *red-black tree* is used for each container. When expanding the current front – being a set of vertices – the algorithm iterates through all vertices in $\mathcal{V}_{front}$ (see line 13 in Alg. 8). The surrounding vertices $\mathcal{N}_x$ of each vertex $\mathbf{v}_x$ in $\mathcal{V}_{front}$ are tested for being either vertices of the previous front $\mathcal{V}_{old}$ (empty on initial expansion) or of the current front $\mathcal{V}_{front}$ or if they are actually new and have not yet been added to $\mathcal{V}_{new}$. After processing all vertices, the containers are swapped (see line 14 in Alg. 8). The old front is replaced with the current one, the current front is replaced with the new vertices, and the new vertex container

is cleaned for the next expansion. After the expansion, $\mathcal{V}_{front}$ contains all unprocessed vertices reachable from the former front.
The VF is an efficient way to investigate the surroundings of a vertex. It is easy to implement and works on a semi-local level. Its most severe disadvantage is the front $\mathcal{V}_{front}$ being defined as a loose set of unconnected vertices. These vertices, depending on the underlying mesh, are often scattered and do not expose a consistent ordered front line, as an *edge-front* does (see below). This vertex based front does not clearly separate the surface into two distinct surface areas. A front expansion direction is not defined for the VF. Without an expansion direction the VF cannot take advantage of the 2D characteristics of a surface to make the process more runtime efficient. If a front collides with itself, it is indeterminable for the process. Neighboring vertices in the front cannot be distinguished from those coming from far distant segments of the front.

5.2.3 Minimal Edge Front

A MEF consists of one or multiple closed edge paths of mesh edges that enclose all vertices of a certain *edge-wise* distance to an initial starting point. This starting point can be one or several vertices or a closed not self-intersecting edge path, for instance, the outline of a triangle. The edge paths posses a front side where the expansion takes place and a back side where the already processed surface connects. In that sense, the MEF behaves like a *sweep line* algorithm where calculations only take place at the front of the current *sweeping line*. In contrast to most such techniques, the MEF creates no additional geometric support structures of any kind, but only uses the given mesh edges. Edge paths are allowed to have shared edges and vertices, but only if both paths touch with their back sides. Touching or crossing front sides are invalid.
For example, if a MEF is expanded two times from a single vertex as starting point, it encloses all vertices which are connected to the initial vertex by two mesh edges (see *(a)* in Fig. 5.3).

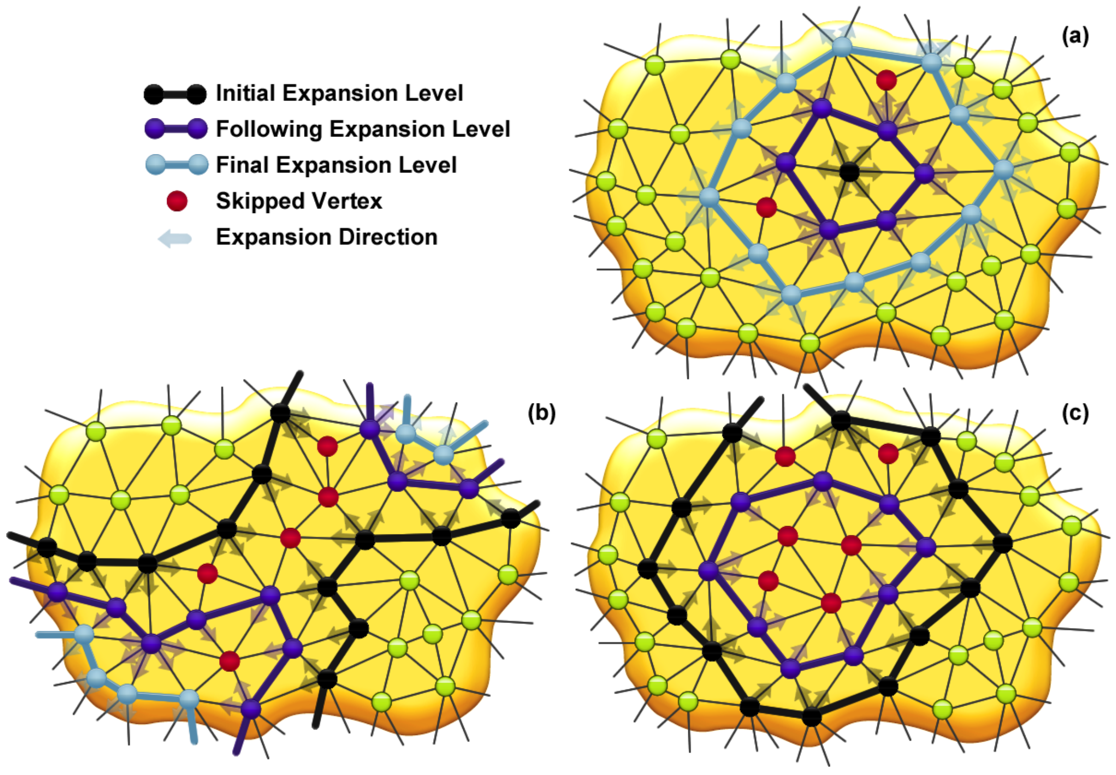


Figure 5.3: Different cases while expanding a minimal edge front: *(a)* expansion from an initial vertex to the second expansion level; *(b)* collision and merging while expanding; *(c)* annihilation of a front that cannot be expanded any further.

A MEF is “minimal” in the way that the vertices which are enclosed by the edge-front cannot be enclosed by a smaller number of mesh edges, unless a front collision would be executed earlier to decrease the number of edges (see below on collision).

Implementation: While the VF contains a loose set of unconnected vertices $\mathscr{V}_{front}$, the front of the MEF is defined by closed edge paths, represented by doubly linked lists (see Alg. 9). So, for every list element $\triangleleft\mathbf{e}\triangleright_x$ of such an edge path previous and posterior edges can easily be investigated. The process includes different containers. In $\mathscr{E}_{front}$ are the edge elements of the current front for a completed expansion level. $\mathscr{E}_{new}$ contains the new edge elements of the front which is currently in progress. $\mathscr{V}_{front}$ contains the vertices of the current front, that are needed to detect collisions. In contrast to the VF, $\mathscr{V}_{front}$ can include duplicates, since two different edge paths can lead through the same vertex twice. As before, these containers can be implemented as *red-black* trees.

Algorithm 9 Minimal Edge Front

```
 1: Clean all containers: $\mathscr{V}_{front} = \mathscr{E}_{front} = \mathscr{E}_{new} = \{\}$
 2: Add starting vertex or edge path to $\mathscr{V}_{front}$ and $\mathscr{E}_{front}$
 3: if Starting point is vertex: $\mathscr{E}_{front} = \{\}$ then
 4:     Translate vertex into edge path
 5: end if
 6: while Expansion level not reached AND front not empty: $n_{exp} > 0 \wedge |\mathscr{V}_{front}| > 0$ do
 7:     repeat
 8:         Select an edge path element $\triangleleft\mathbf{e}\triangleright_x$ from $\mathscr{E}_{front}$
 9:         Calculate the edge path segment $\triangleleft\mathbf{E}\triangleright_x$ in front of $\triangleleft\mathbf{e}\triangleright_x$ and $previous(\triangleleft\mathbf{e}\triangleright_x)$
10:         Minimize edge path segment $\triangleleft\mathbf{E}\triangleright_x$
11:         repeat
12:             Add vertex from the edge path segment $\triangleleft\mathbf{E}\triangleright_x$ to front $\mathscr{V}_{front}$
13:             if Vertex is already present in $\mathscr{V}_{front}$ then
14:                 Split path in $\triangleleft\mathbf{E}\triangleright_x$ in path before and after collision
15:                 Minimize the edge path segments in $\triangleleft\mathbf{E}\triangleright_x$
16:             end if
17:         until All vertices of segment(s) $\triangleleft\mathbf{E}\triangleright_x$ have been added to $\mathscr{V}_{front}$
18:         Connect the new edge path segment(s) $\triangleleft\mathbf{E}\triangleright_x$ to the edge-front
19:         Delete processed previous elements from $\mathscr{V}_{front}$, $\mathscr{E}_{front}$ and $\mathscr{E}_{new}$
20:     until No more edge path elements to process: $\mathscr{E}_{front} = \{\}$
21:     Swap containers: $\mathscr{E}_{front} = \mathscr{E}_{new} \wedge \mathscr{E}_{new} = \{\}$
22:     Decrease expansions: $\Delta n_{exp} = -1$
23: end while
```

Expansion: If the *edge-front* defines a complete extension level, all edge path elements are in $\mathscr{E}_{front}$ and $\mathscr{E}_{new}$ is empty. If this front is expanded all single edge elements in $\mathscr{E}_{front}$ need to be expanded, creating a new edge path which is added to $\mathscr{E}_{new}$. If all former edges of a previous expansion level are expanded and $\mathscr{E}_{front}$ is empty (see line 20 in Alg. 9), a new expansion level is reached (see *(a)* in Fig. 5.3). Then the content of $\mathscr{E}_{new}$ and $\mathscr{E}_{front}$ is swapped (see line 21 in Alg. 9), so $\mathscr{E}_{new}$ is empty again and $\mathscr{E}_{front}$ contains the new expansion level. The parameter n_{exp} determines the number of expansion levels performed to reach the desired final front state.

Single Edge Expansion: The expansion of the front to a new expansion level includes the expansion of single edge path element, as illustrated in Fig. 5.4. The expansion of a single edge path element starts by picking one edge path element of the former front from $\mathscr{E}_{front}$ (see line 8 in Alg. 9). Since edge path elements are doubly linked list elements, the edge path element previous to the picked one can be easily accessed. In between those two edge path elements lies a vertex. The process iterates through all vertices, which are connected to that vertex and that lie in expansion direction. The path of edges $\triangleleft\mathbf{E}\triangleright_x$ connecting those vertices is determined (see *(a)* in Fig. 5.4 and line 9 in Alg. 9). $\triangleleft\mathbf{E}\triangleright_x$ is supposed to be added to the front. First, however, the path needs to be minimized (see *(b)* in Fig. 5.4 and line 10 in Alg. 9). Whenever the edge path

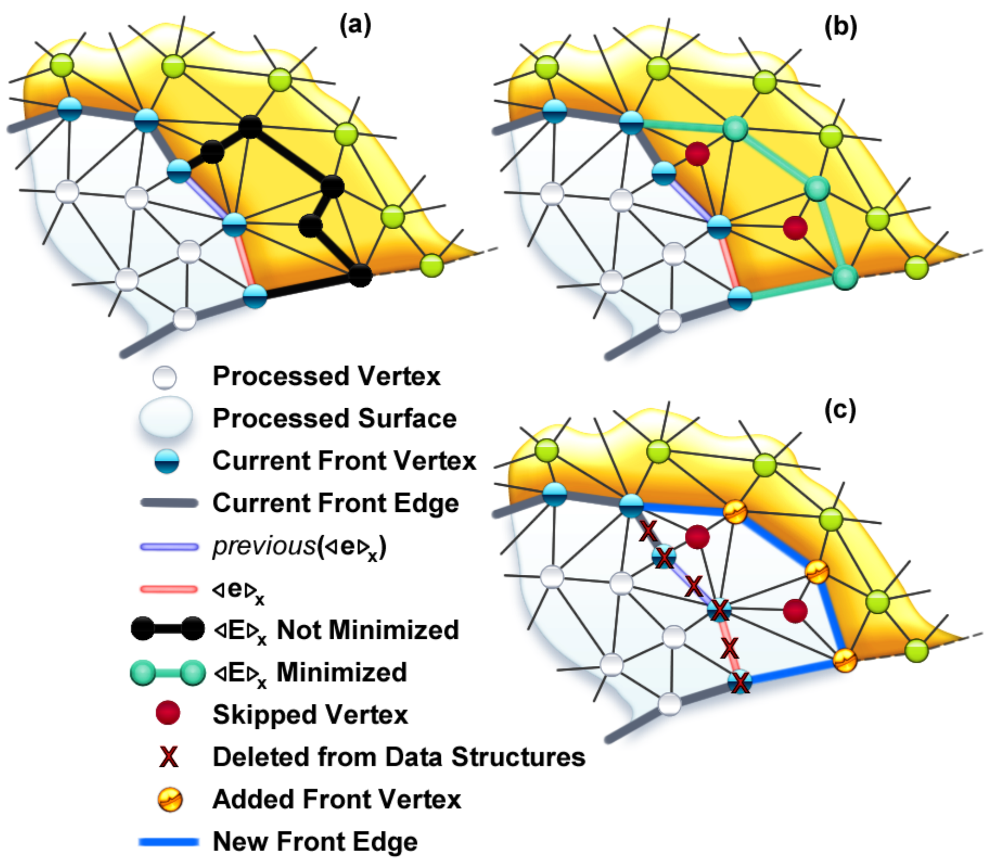


Figure 5.4: Expansion of a single edge path element: *(a)* the new edge path segment $\triangleleft \mathbf{E} \triangleright_x$ in front of $\triangleleft \mathbf{e} \triangleright_x$ and $previous(\triangleleft \mathbf{e} \triangleright_x)$ is determined; *(b)* the segment length is minimized in length; *(c)* the new segment is connected to the front and processed front elements are deleted from their corresponding data structures. Note that more than the initial two edge path elements have been deleted.

passes two edges of a triangle – where the third one is not part of the path – the edge path can be shortened by redirecting the path through that third edge (see *(b)* in Fig. 5.4). Given that no collision takes place, the shortened path $\triangleleft \mathbf{E} \triangleright_x$ can be connected with the edge-front and its edge path elements added to $\mathscr{E}_{new}$ (see *(c)* in Fig. 5.4 and line 18 in Alg. 9). Vertices and edge path elements which lie behind the current front after the expansion are now part of the processed surface and can be deleted from the corresponding data structures (see line 19 in Alg. 9).
Since the MEF works based on edges, a vertex as a starting point represents an anomaly. However, when using one of its triangles as an edge path, adding one edge incident to the vertex into $\mathscr{E}_{front}$ – so that it is expanded for the next expansion level – and the other two into $\mathscr{E}_{new}$ – so that they will not be expanded for the next expansion level – the next expansion builds the front in one edge-wise distance to that vertex (see *(a)* in Fig. 5.3).
Annihilation: If the minimization of $\triangleleft \mathbf{E} \triangleright_x$ leaves no remaining surface area to process, then the front has been annihilated and ceases to exist (see *(c)* in Fig. 5.3). The annihilation of an edge-front often marks the end of a search process, since the front cannot be further expanded.
Boundaries: If an edge-front is expanded at a boundary, its vertices are deleted from $\mathscr{V}_{front}$, since collisions cannot take place anymore, and its edge elements are deleted from the $\mathscr{E}_{front}$, since they cannot be expanded anymore. However, the edge path elements are still kept in the doubly linked list. From a memory efficiency perspective this is not ideal, since all boundary edges passed by the MEF are kept, and thus use up memory. However, this implementation has the advantage that edge paths are always closed and complete, therefor making special case handling unnecessary when accessing edge path elements.

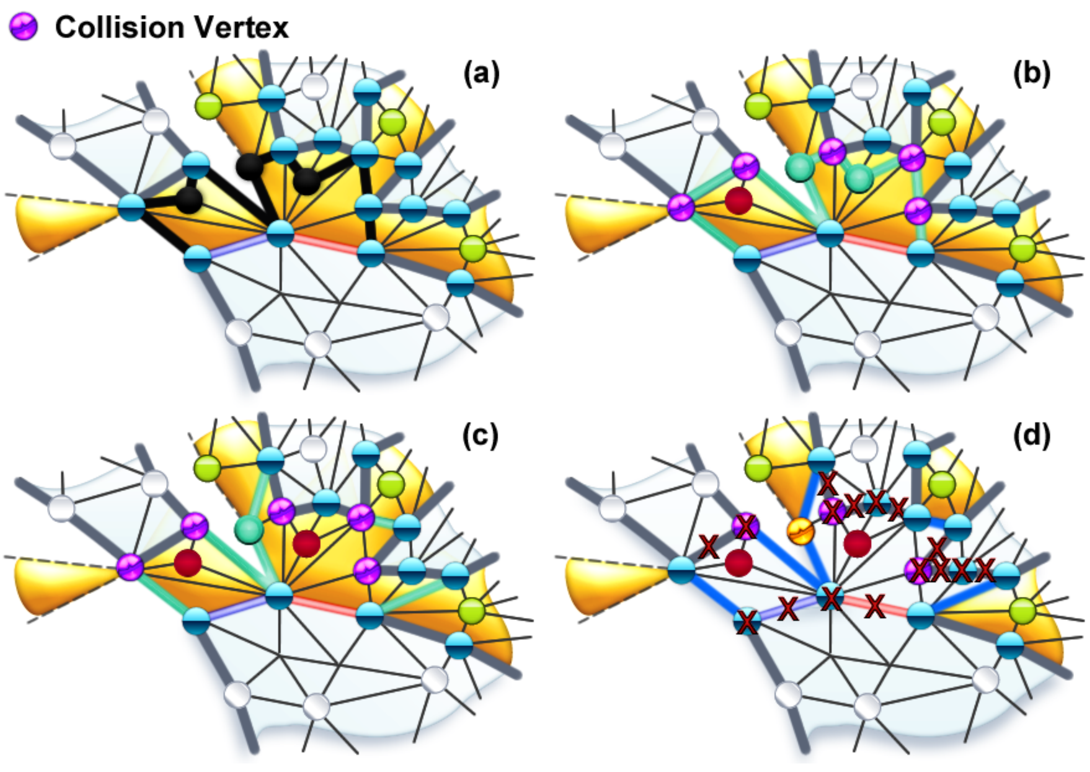


Figure 5.5: A wide range of collision cases at a single edge path element expansion: *(a)* determination of path segment $\triangleleft\mathbf{E}\triangleright_x$; *(b)* the path segment is minimized and five collisions are detected; *(c)* due to the collisions the path is split in six and the resulting path segments are again minimized and two of them are annihilated; *(d)* finally four new path segments are connected to the front.

Collision: Since an edge-front defines a continuous contour, instead of an unorganized vertex set, collisions can and have to be detected. The detection is necessary to inhibit fronts from permeating each other (see *(b)* in Fig. 5.3).
The detection of collisions takes place at a single edge expansion (see Fig. 5.5). If a former edge path element has been expanded and minimized, a new expansion path segment $\triangleleft\mathbf{E}\triangleright_x$ can be added to the front. Before a new path is connected, however, it needs to be tested whether it collides with an existing edge-front. All vertices of the new path are successively added to the front vertices $\mathscr{V}_{front}$ (see line 12 in Alg. 9). If one of these vertices is already present in $\mathscr{V}_{front}$, this indicates a collision (see *(b)* in Fig. 5.5). The path in $\triangleleft\mathbf{E}\triangleright_x$ is then split (see line 14 in Alg. 9) and the new path segments need to be minimized again (see *(c)* in Fig. 5.5 and line 15 in Alg. 9). The collision test has to be vertex based, since a collision can involve one vertex only (see *(b)* in Fig. 5.5). A single edge element expansion can include several collisions (see Fig. 5.5). If all collision tests are performed, the potentially separated segments in $\triangleleft\mathbf{E}\triangleright_x$ are connected to the front.
Bottleneck: When allowing minimum pathways with a perimeter of only three vertices in a mesh (see bottleneck in section 4.2.2), certain cases will require further consideration in the implementation of the MEF. If a new expansion path segment $\triangleleft\mathbf{E}\triangleright_x$ has been calculated and minimized, it is connected to the pre-existent edge-front. If bottlenecks exist in a mesh, it is possible for that pre-existent edge-front to be entirely skipped by the minimization process, while $\triangleleft\mathbf{E}\triangleright_x$ contains a valid new edge-front purely by itself. This creates a variety of special cases if fronts at bottlenecks are expanded or collide. Since those cases are highly implementation specific, they are not listed here.

5.2.4 Minimal Distance Front

The presented MEF is an efficient processing tool to analyze and search for aspects concerning the connectivity in a mesh. From a connectivity focused perspective any vertex connection or edge is equally valued. This setting therefore only considers discrete mesh connectivity aspects in its calculation. If, for instance, vertex positions are altered and thereby also the mesh geometry, the MEF processing would remain unchanged. The MEF, however, exposes a front line straightened by the path minimization. This front line can be directionally set in relation to the surface, establishing on-surface distances to the front line. The *minimal distance front* (MDF) (see Alg. 10) is a MEF with a distance driven selection process for the expanded edge path elements.

Algorithm 10 Minimal Distance Front (based on MEF Alg. 9)

```
 1: Clean all containers: $\mathscr{V}_{front} = \mathscr{E}_{front} = \mathscr{E}_{new} = \{\}$
 2: Add starting vertex or edge path to $\mathscr{V}_{front}$ and $\mathscr{E}_{front}$
 3: if Starting point is vertex: $\mathscr{E}_{front} = \{\}$ then
 4:    Translate vertex into edge path
 5: end if
 6: Calculate the anticipated distances for all edges in $\mathscr{E}_{front}$ after expansion and add to $\mathscr{D}_{exp}$
 7: while Expansions not empty AND next one does not exceed maximum distance:
        $|\mathscr{D}_{exp}| > 0 \wedge min(\mathscr{D}_{exp}) <= d_{max}$
        ~~Expansion level not reached AND front not empty: $n_{exp} > 0 \wedge |\mathscr{V}_{front}| > 0$~~ do
 8:    Get edge path element $\triangleleft \mathbf{e} \triangleright_x$ associated with the smallest anticipated distance $min(\mathscr{D}_{exp})$
    ~~repeat~~
 9:    Select ~~an edge path element~~ $\triangleleft \mathbf{e} \triangleright_x$ from $\mathscr{E}_{front}$
10:    Calculate the edge path segment $\triangleleft \mathbf{E} \triangleright_x$ in front of $\triangleleft \mathbf{e} \triangleright_x$ and $previous(\triangleleft \mathbf{e} \triangleright_x)$
11:    Minimize edge path segment $\triangleleft \mathbf{E} \triangleright_x$
12:    repeat
13:       Add vertex from the edge path segment $\triangleleft \mathbf{E} \triangleright_x$ to front $\mathscr{V}_{front}$
          with on-surface distance $d_{start}$ and direction to starting point $\mathbf{d}_{start}$
14:       if Vertex is already present in $\mathscr{V}_{front}$ then
15:          Split path in $\triangleleft \mathbf{E} \triangleright_x$ in path before and after collision
16:          Minimize the edge path segments in $\triangleleft \mathbf{E} \triangleright_x$
17:       end if
18:    until All vertices of segment(s) $\triangleleft \mathbf{E} \triangleright_x$ have been added to $\mathscr{V}_{front}$
19:    Connect the new edge path segment(s) $\triangleleft \mathbf{E} \triangleright_x$ to the edge-front
20:    Delete processed previous elements from $\mathscr{V}_{front}$, $\mathscr{E}_{front}$, $\mathscr{E}_{new}$ and $\mathscr{D}_{exp}$
21:    Anticipate the expansion distances of all new edge path elements and add those to $\mathscr{D}_{exp}$
22:    Add all edge path elements to current front: $\mathscr{E}_{front} = \mathscr{E}_{front} \cap \mathscr{E}_{new} \wedge \mathscr{E}_{new} = \{\}$
    ~~until No more edge path elements to process: $\mathscr{E}_{front} = \{\}$~~
    ~~Swap containers: $\mathscr{E}_{front} = \mathscr{E}_{new} \wedge \mathscr{E}_{new} = \{\}$~~
    ~~Count down expansions: $\Delta n_{exp} = -1$~~
23: end while
```

To select an edge path element for expansion depending on the resulting distance to the *starting point* makes the anticipation of these distances necessary. All these anticipated distances with their associated edge path elements are added into a container, which orders them by their anticipated distances. Again, a *red-black* tree is suitable for this task. The MEF has discrete expansion levels. Those are defined as the total of single expansions needed to move the front one single edge forward (see line 20 in Alg. 9). The MDF is not bound to such distinct levels as defined by n_{exp}, instead a front defines the farthest distant edge path to a starting point not exceeding a certain on-surface maximum distance d_{max} (see line 7 in Alg. 10). In Fig. 5.6, the difference of these expansion behaviors is illustrated.

If front vertices are added to $\mathscr{V}_{front}$, additionally their on-surface distance d_{start} and the on-surface direction toward the starting point $\mathbf{d}_{start}$ are added (see line 13 in Alg. 10). For the initial vertices d_{start} is zero. For a single vertex $\mathbf{d}_{start}$ is set to the *zero vector*. If a vertex is part of an edge path

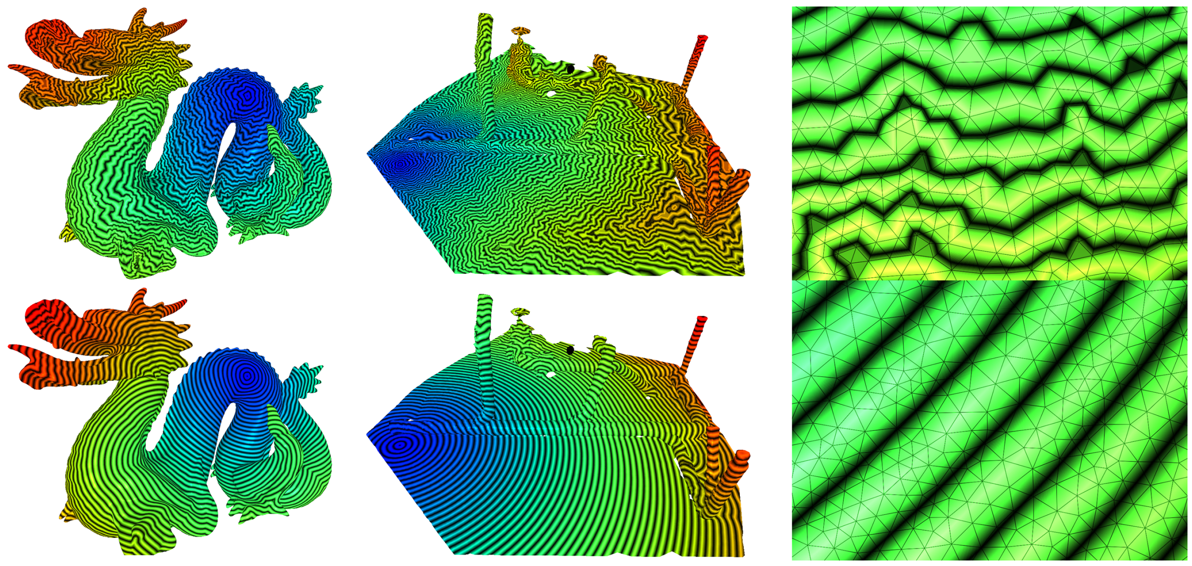


Figure 5.6: Comparison of MEF and MDF: Distance field on the Dragon (left column), the Heating Pipes (middle column) and a distance field close up (right column). The distances in the MEF (top row) represent discrete edge counts, while the MDF (bottom row) represents geometric distances. Due to skipped vertices and discrete distances in case of the MEF, all vertices of a triangle can have the same distance, creating a unicolored triangle.

that is the initial starting point of the algorithm, $\mathbf{d}_{start}$ can be calculated similar to a *boundary normal* of a vertex at a mesh boundary, conceiving the backside of the edge path (opposite side to the front expansion direction) as a surface boundary.

Expansion: The MDF continuously expands with the expansion of every single edge path element. In contrast, the MEF expands all former edge path elements to reach the next discrete expansion level. If a single edge path element in the MDF is expanded, the element with the smallest anticipated on-surface distance is selected (see line 8 in Alg. 10). The smallest distance is chosen to keep the front at minimum distance to the starting point. The expansion of a single edge path element itself remains unchanged to one presented for the MEF.

If an edge path element $\triangleleft\mathbf{e}\triangleright_x$ has been expanded, the vertices of the new edge path segment $\triangleleft\mathbf{E}\triangleright_x$ are added to $\mathscr{V}_{front}$ (see line 13 in Alg. 10). For every given vertex $\mathbf{v}_x$ of the new edge path additionally the distance d_{start} and vector $\mathbf{d}_{start}$ need to be calculated.

The previous edge-front is seen as a collection of separated spatial subdivisions delimited by planes between them. Those subdivisions are either associated with an edge, or between two edge subdivisions with a vertex.

A subdivision of an edge $\mathbf{e}_{sub}$ between vertex $\mathbf{v}_{left}$ at its left and $\mathbf{v}_{right}$ at its right is a space determined by two planes, one intersecting $\mathbf{v}_{left}$ and the other one intersecting $\mathbf{v}_{right}$ (see *(b)* in Fig. 5.7).

For a vertex $\mathbf{v}_x$ to be considered inside a subdivision, it has to lie on or above both of these planes (see *(a)* in Fig. 5.7). Both planes run parallel to the normal of the triangle $\mathbf{t}_{sub}$ that lies on the backside of $\mathbf{e}_{sub}$ and is parallel to $\mathbf{d}_{start}$ of either $\mathbf{v}_{left}$ for the left plane and $\mathbf{v}_{right}$ for the right plane (see *(b)* in Fig. 5.7).

The calculation starts by identifying the subdivision which covers the vertex $\mathbf{v}_x$ under investigation. First, the subdivision associated with the edge of $\triangleleft\mathbf{e}\triangleright_x$ is tested. $\mathbf{v}_x$ might be inside of this subdivision, right, or left from it. If $\mathbf{v}_x$ lies left from it, the previous subdivision of the front $previous(\triangleleft\mathbf{e}\triangleright_x)$ is tested if $\mathbf{v}_x$ is right from it the next subdivision $next(\triangleleft\mathbf{e}\triangleright_x)$ is tested. If $\mathbf{v}_x$ lies first at the left and for the following tested subdivision at the right or vice versa, $\mathbf{v}_x$ lies in the subdivision of the vertex in between. It is sensible to limit the number of tested subdivisions. In the presented implementation, maximally 5 subdivisions are tested. If this limit is exceeded,

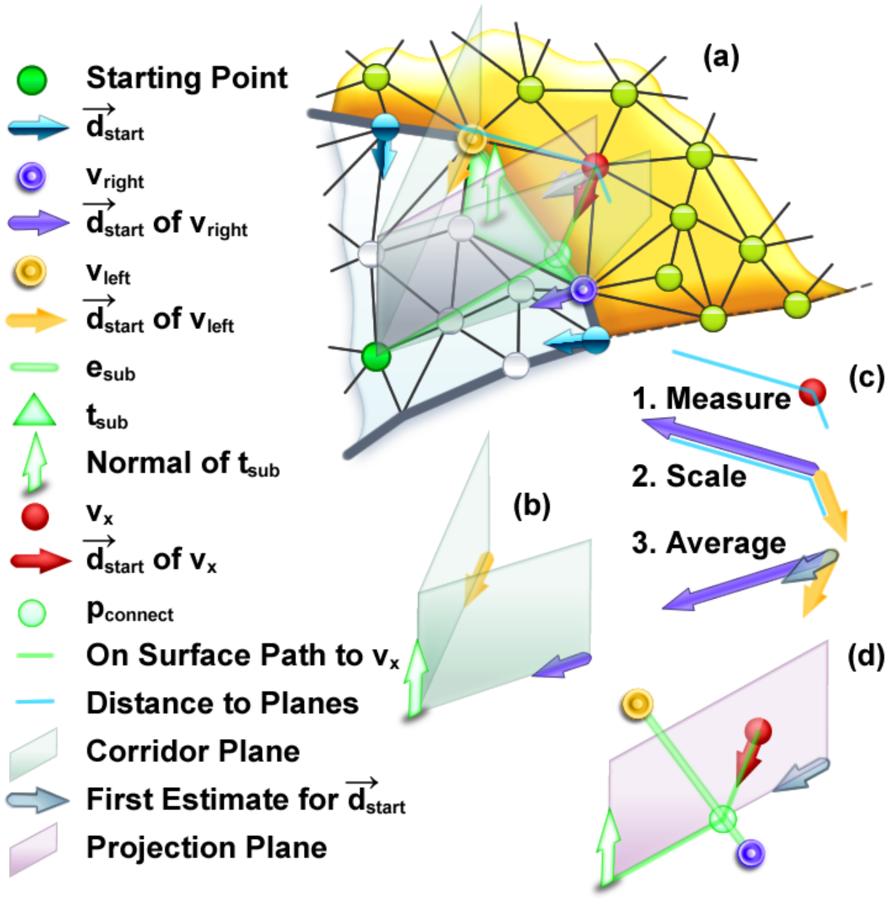


Figure 5.7: The calculation of d_{start} and $\mathbf{d}_{start}$: *(a)* with all aspects combined; *(b)* the subdivision of edge $\mathbf{e}_{sub}$; *(c) 1.* the distance of $\mathbf{v}_x$ to the subdivision planes is determined; *2.* it is used to scale $\mathbf{d}_{start}$ of $\mathbf{v}_{left}$ and $\mathbf{v}_{right}$; *3.* finally the normalized average of these vectors is calculated as a first estimate of $\mathbf{d}_{start}$ for $\mathbf{v}_x$; *(d)* this first estimated vector is projected onto the surface to determine the intersection point $\mathbf{p}_{connect}$ of $\mathbf{e}_{sub}$ with the starting point path of $\mathbf{v}_x$. Note that the subdivision planes do not necessarily intersect at the starting point.

the result of the search is set to the subdivision associated with the vertex in between $\triangleleft \mathbf{e} \triangleright_x$ and $previous(\triangleleft \mathbf{e} \triangleright_x)$ as a fallback solution. This fallback is also used in case a subdivision cannot be calculated, since $\mathbf{t}_{sub}$ does not exist or an accessed vector $\mathbf{d}_{start}$ is the zero vector.
For $\mathbf{v}_x$ lying in the subdivision of a vertex $\mathbf{v}_{sub}$ the calculations are simple. d_{start} for $\mathbf{v}_x$ is the distance form $\mathbf{v}_x$ to $\mathbf{v}_{sub}$ plus the distance form $\mathbf{v}_{sub}$ back to the starting point d_{start}, which already has been calculated and can be accessed through $\mathscr{V}_{front}$. The vector $\mathbf{d}_{start}$ is the normalized vector running from $\mathbf{v}_x$ to $\mathbf{v}_{sub}$.
If $\mathbf{v}_x$ lies in the subdivision of an edge $\mathbf{e}_{sub}$, a first estimate of $\mathbf{d}_{start}$ is approximated (see *(c)* in Fig. 5.7). The direction of $\mathbf{d}_{start}$ for $\mathbf{v}_x$ needs to lie in between $\mathbf{d}_{start}$ of $\mathbf{v}_{left}$ and $\mathbf{d}_{start}$ of $\mathbf{v}_{right}$. To what degree one of these vectors is represented in $\mathbf{d}_{start}$, is supposed to resemble to which plane $\mathbf{v}_x$ lies closer to. Since a low distance implies more importance, the relation from representation to distance is inversely proportional. Therefore, $\mathbf{d}_{start}$ of $\mathbf{v}_{left}$ is scaled to the distance of $\mathbf{v}_x$ to the right plane and $\mathbf{d}_{start}$ of $\mathbf{v}_{right}$ is scaled to the distance of $\mathbf{v}_x$ to the left plane. Then the normalized average of these vectors is used as a first estimate for $\mathbf{d}_{start}$ for $\mathbf{v}_x$. Now, $\mathbf{d}_{start}$ represents the direction to the starting point under the assumption that the surface progresses like a plane.
If the surface curved, this first estimate needs to be projected onto the surface (see *(d)* in Fig. 5.7). This is achieved by setting up another plane which has $\mathbf{v}_x$ as its origin point and runs parallel to the estimated vector for $\mathbf{d}_{start}$ and to the normal of triangle $\mathbf{t}_{sub}$. The point $\mathbf{p}_{connect}$, where

this plane intersects with $\mathbf{e}_{sub}$, represents the point where the on-surface path from the starting point to $\mathbf{v}_x$ crosses the front. With this intersection point distance d_{start} and vector $\mathbf{d}_{start}$ can be determined for $\mathbf{v}_x$.
The distance d_{start} of $\mathbf{v}_x$ is the distance from $\mathbf{v}_x$ to $\mathbf{p}_{connect}$ plus the distance back to the starting point. The latter distance is again calculated as the average of d_{start} of $\mathbf{v}_{left}$ and $\mathbf{v}_{right}$ inversely proportionally weighted by the distances to the subdivision planes. The final $\mathbf{d}_{start}$ for $\mathbf{v}_x$ is the normalized vector from $\mathbf{v}_x$ to $\mathbf{p}_{connect}$.
If a single edge expansion has been finished and the segment(s) in $\triangleleft\mathbf{E}\triangleright_x$ have been connected to the front (see line 19 in Alg. 10), the process iterates through all new front edges. The on-surface distances these edges would lead to, in case of their expansion, are anticipated.Then edge path elements with their corresponding, anticipated distances are added to $\mathscr{D}_{exp}$ (see line 21 in Alg. 10).
The anticipated distance for an edge path element $\triangleleft\mathbf{e}\triangleright_x$ is calculated as the maximum of the distances d_{start} its expansion would create. So, the d_{start} values of all vertices in the edge path segment $\triangleleft\mathbf{E}\triangleright_x$ that would be created have to be determined. d_{start} can be calculated as before when adding new vertices to the front.

5.2.5 Calculating Connection Path

One of the many applications of the presented data structures is to calculate a connection path between two vertices in a mesh efficiently in runtime and memory consumption. To accomplish this, two fronts are initialized, one at each vertex. Both are alternatingly expanded and new front vertices are tested for being present in both fronts. If so, the fronts overlap at this vertex. If one front cannot be further expanded, the initial vertices are not connected, otherwise an overlapping vertex must have been found. This vertex then lies in the middle of the path between the initial search vertices. With the new vertex the process can be recursively repeated, until the point where two initial search vertices are adjacent to one another. The edges of adjacent vertices are added to the connection path. When all recursive search processes for vertices have been brought down to adjacent vertices, the entire path has been determined.
In order to search for the edge-wise shortest connection path, the VF as well as the MEF can be applied (see Fig. 5.8). The MEF is more efficient in runtime and memory consumption. Since every edge is equally valued an edge-wise shortest connection path is normally one of many possible paths of the same edge-wise length.
When using the MDF, the shortest on-surface path is determined (see Fig. 5.8). To calculate the length of this path, the sum of the distances d_{start} to the first overlapping vertex of both initial fronts can be built. This length resembles the one of the path as actually projected onto the surface, a path that passes across a triangle. However, the resulting path is constructed of exiting mesh edges and does not resemble the projected path. This projected path would be a valuable additional asset in geometry processing, since it is independent of the given mesh edges. With the available $\mathbf{d}_{start}$ vectors this path could potentially be calculated. Note that the MDF based path calculation is only one possible example of applying the presented data structure.

5.2.6 Twist Solving

The twist solving mechanism for the GCS approach involves three steps. First, a twist within the current surface estimate is identified, second, the twisted surface regions are separated from one another, and third, the twisted segment is turned around.
The third step is fairly easy. Depending on the underlying data structure, explicit normals are inverted, or vertex orders are reversed when changing the orientation of a mesh segment.

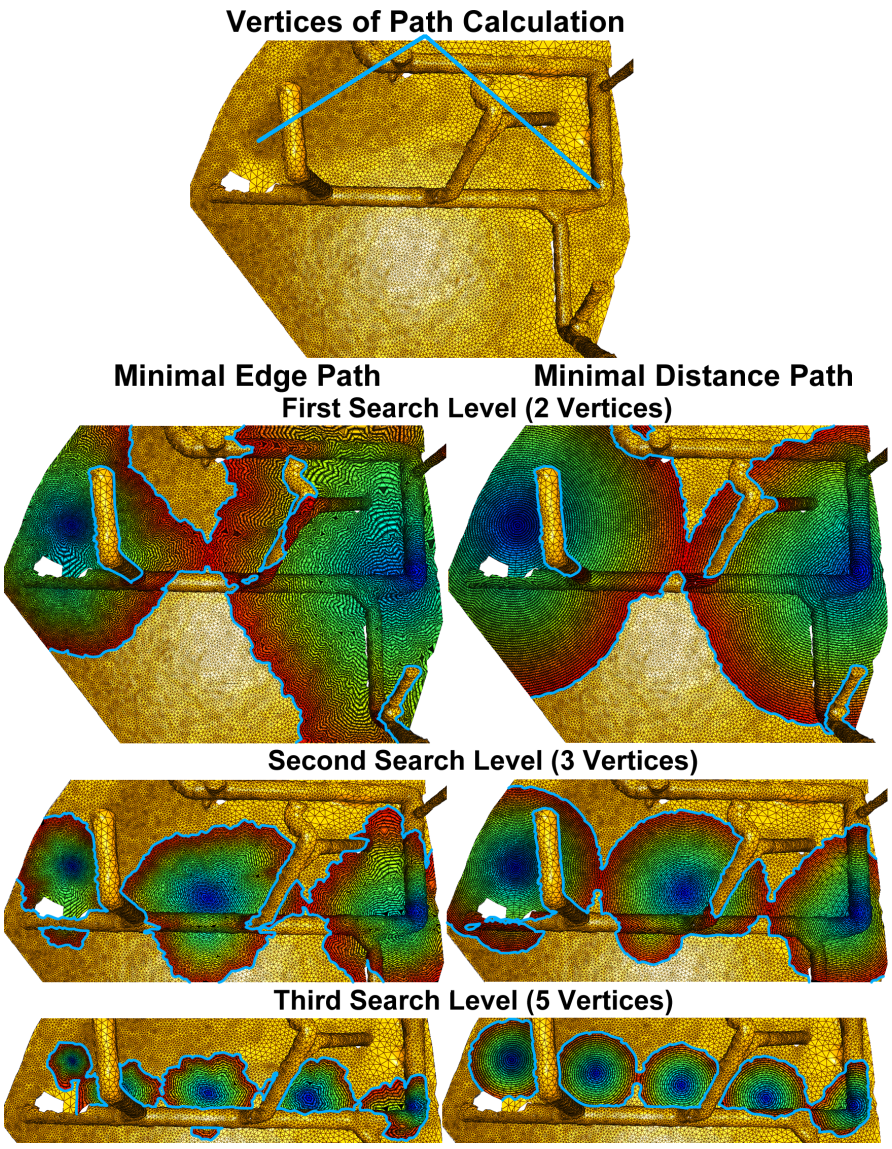

Vertices of Path Calculation
Minimal Edge Path
Minimal Distance Path
First Search Level (2 Vertices)
Second Search Level (3 Vertices)
Third Search Level (5 Vertices)

Figure 5.8: Calculation of a connection path between two vertices (top) with the MEF (left column) and the MDF (right column). Showing the first (second row), the second (third row) and the third (bottom row) recursive search level. Note how the expansion of the MEF is determined by the local triangle resolution and how the equal edge-wise expansions differ in size and shape.

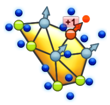
The first step is invoked in combination with the coalescing operation. This is a convenient location to attach this function, since the operation already involves a boundary vertex $\mathbf{v}_x$, the search for an opposing boundary vertex $\mathbf{v}_{opp}$, and a surface orientation test (see line 18 in Alg. 11).

5.2.6.1 Twist Detection

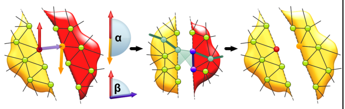
A twist between two boundaries is determined by opposite normal orientations at the same planar domain. This is the case if, first, the size of the angle α between the normals $\mathbf{n}_x$ and $\mathbf{n}_{opp}$ of the opposing vertices $\mathbf{v}_x$ and $\mathbf{v}_{opp}$ exceeds 170° (see line 13 in Alg. 11), and second, if $\mathbf{v}_x$ and $\mathbf{v}_{opp}$ lie at the same planar domain, *i.e.*, if the angle β between $\mathbf{n}_x$ and the vector from $\mathbf{v}_x$ to $\mathbf{v}_{opp}$ lies in between 80° and 100° (see Fig. 5.9).

Algorithm 11 Twist Solving (based on filter chain Alg. 7)

```
1:  Initialization
2:  repeat
3:      repeat
4:          repeat
                Basic Step
```



```
5:              Select random sample $\mathbf{p}_x$ of $\mathscr{P}$
6:              Find the winning vertex $\mathbf{v}_x$ that exposes the smallest Euclidian distance to $\mathbf{p}_x$
7:              Discontinuity Adaption
8:              Curvature Adaption
9:              Decrease signal counters of all other vertices by a fraction: $\forall sc_i(i \in \mathscr{M})\ \Delta sc_i = -\beta \cdot sc_i$
10:             if $\mathbf{v}_x$ is boundary vertex AND coalescing partner $\mathbf{v}_{opp}$ was found then
11:                 Coalescing
12:             else
                    Solve Twist
```



```
13:                 if $\mathbf{n}_x$ and $\mathbf{n}_{opp}$ and their surrounding indicate a twisted surface region then
14:                     while A connection path between $\mathbf{v}_x$ and $\mathbf{v}_{opp}$ has been found do
15:                         Cut connection path between $\mathbf{v}_x$ and $\mathbf{v}_{opp}$ at its smallest width
16:                         if Connection path could not be cut then
17:                             break
18:                         end if
19:                     end while
20:                     if Separate surface segments were created then
21:                         Swap the orientation of the smaller surface segment
22:                     end if
23:                 end if
24:             end if
25:             Filter Chain
26:             Increment the iteration counter: $\Delta t = 1$
27:         until The basic step has been performed $c_{add}$ times: $t$ mod $c_{add} = 0$
28:         Vertex Split
29:     until The basic step has been performed $c_{del}$ times: $t$ mod $c_{del} = 0$
30:     Edge Collapse
31: until A certain number of vertices is reached: $|\mathscr{M}| >= n_{final}$
```

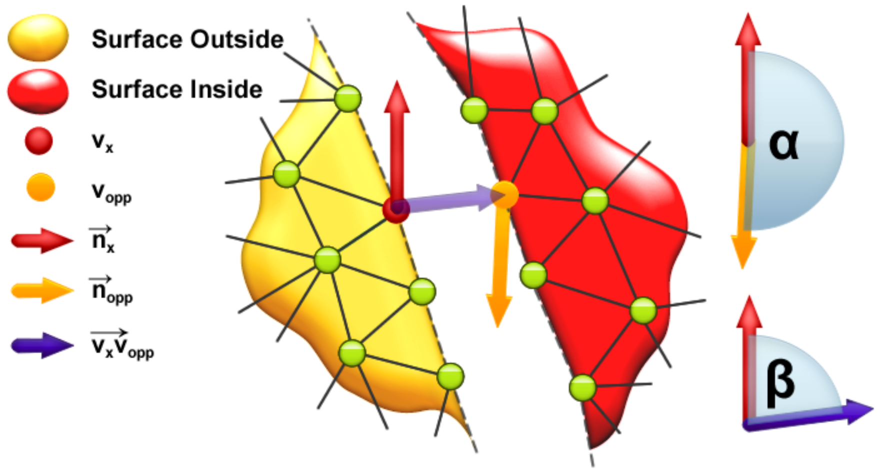


Figure 5.9: Detection of a twisted surface through angles α between two vertex normals and β between a normal and the vector connecting the two vertices under consideration.

Since resolving a twist is very costly in runtime, superfluous resolving operations need to be avoided. Thus, the above test is repeated for both neighbors of $\mathbf{v}_x$ and $\mathbf{v}_{opp}$, and only if all three tests expose a twist a valid detection is assumed.
This kind of detection is chosen arbitrarily and one can think of several alternatives. However, the specific kind of this test is not significant for the overall algorithm's validness or performance, since it is not required to detect a twist at its earliest stage as long as it *is* detected at some time. The latter is guaranteed by the refinement process, since twisted surface areas are refined until the twist is recognized by the detection mechanism. This approximate detection can only fail if the twist is not exposed at any boundary or if the twisted surface has not been clearly exposed before the process ends.

5.2.6.2 Twist Separation

If the surfaces, which $\mathbf{v}_x$ and $\mathbf{v}_{opp}$ belong to, are detected as being twisted, one of them needs to be turned around. As long as the surfaces are connected, the turning operation would fail since both surfaces would be turned together. It seems to be surprising that differently orientated surface areas could be connected since the coalescing process would avoid such a connection. Nevertheless, this case may happen if these surfaces are connected during an earlier refinement stage where the difference of the normals has not yet been developed that far. Fig. 5.10 gives an impression of the various kinds of connections that may exist between twisted surfaces.
To isolate these different structures by heuristics, using geometric properties such as vertex positions and normals, is extremely difficult and unreliable. The presented process therefore relies only on the mesh connectivity for detecting these cases. This is achieved by using the presented MEF.
To find out if $\mathbf{v}_x$ and $\mathbf{v}_{opp}$ are connected, the edge-wise shortest *connection path* between them is calculated (see *(a)* in Fig. 5.11). This is done as presented in section 5.2.5 with initially expanding two MEFs. If one front is annihilated and no connection path exists, then $\mathbf{v}_x$ and $\mathbf{v}_{opp}$ are unconnected. Since the expansion of both fronts has been done concurrently, the vertex of the annihilated front belongs to the smaller surface segment. At this point the separateness of $\mathbf{v}_x$ and $\mathbf{v}_{opp}$ is proven and the orientation of one segment can be turned around. The smaller one is chosen for this operation to minimize the effort.
If a connection path between $\mathbf{v}_x$ and $\mathbf{v}_{opp}$ has been found, the corresponding mesh parts must be separated before turning one of them around. To separate them, the shortest cut that interrupt this connection path is searched for.

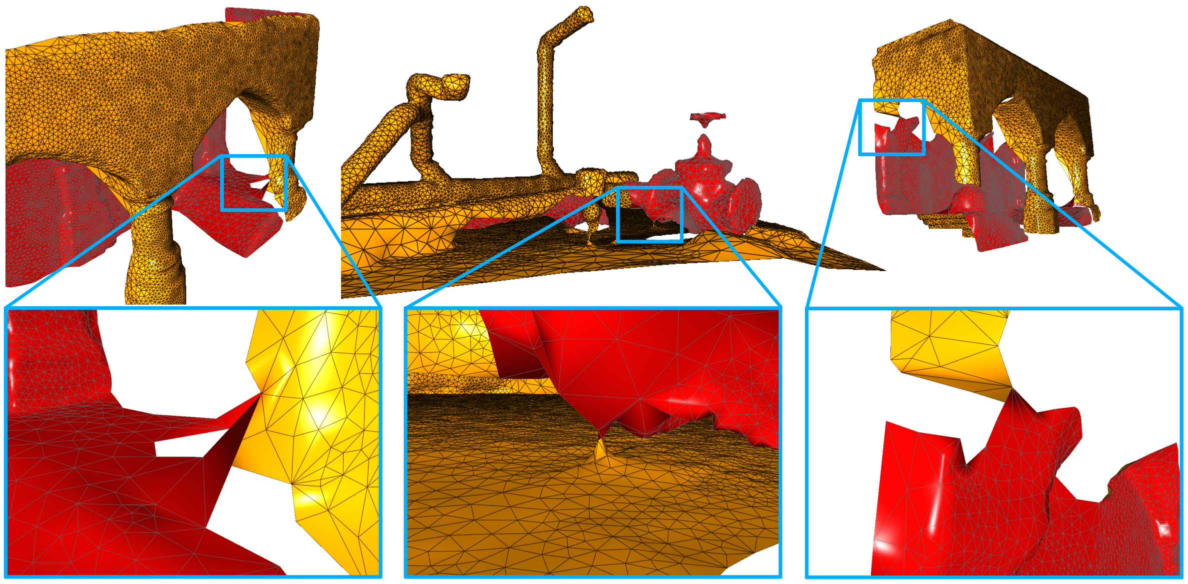


Figure 5.10: Several examples of twisted connections of surface pieces. Twists at boundary vertices (right and left), and a self intersection (middle). Note that contrarily oriented triangles are never connected directly.

In a first stage, the shortest cut across the path that starts and ends at a boundary vertex is determined. The search starts with the cut length of zero, involving a multiple boundary vertex (see *(b)* in Fig. 5.11). The search is conducted for all vertices on the path. If a zero length cut cannot be found, the search length is increased by one. To search for a cutting length higher than zero along the path, the two closest boundary vertices to a given vertex of the connection path are determined by applying the MEF. The connection between these two boundary vertices defines a cutting path. In *(c)* in Fig. 5.11, a cutting path of length one is shown. The cut need to be tested whether it would actually interrupt the path between $\mathbf{v}_x$ and $\mathbf{v}_{opp}$. In *(e)* in Fig. 5.11, a cutting path of length one that fails to interrupt the connection path is shown. An additional test whether the cut would detach $\mathbf{v}_x$ or $\mathbf{v}_{opp}$ from the surface as shown in *(f)* in Fig. 5.11 is needed. If no sufficient cut is found in the first stage, the second search stage is started. Here, circular cuts are investigated, which do not necessarily involve boundary vertices, and which are defined by a cutting path with circular connected edges. The search for circular cuts starts with a length of three as shown in *(d)* in Fig. 5.11. To test a vertex on the path, a MEF is initialized at that vertex and the front is expanded until a collision of the front takes place, or the current search length is exceeded. If this path interrupts the connection path between $\mathbf{v}_x$ and $\mathbf{v}_{opp}$ and the triangles intersect each other, as in *(d)* in Fig. 5.11, it is a valid cutting path.
If a cutting path is found that satisfies the presented criteria, that cut is performed (see line 15 in Alg. 11). All edges of the side of the cutting path that includes fewer triangles are deleted (see Fig. 5.11). Now $\mathbf{v}_x$ and $\mathbf{v}_{opp}$ are tested again for a connection and the process repeats itself until $\mathbf{v}_x$ and $\mathbf{v}_{opp}$ are separated (see line 14 in Alg. 11). If a maximum cutting length for boundary to boundary and circular cut is set, it is possible for surface regions to remain connected and no orientation swap can be performed (see line 20 in Alg. 11).

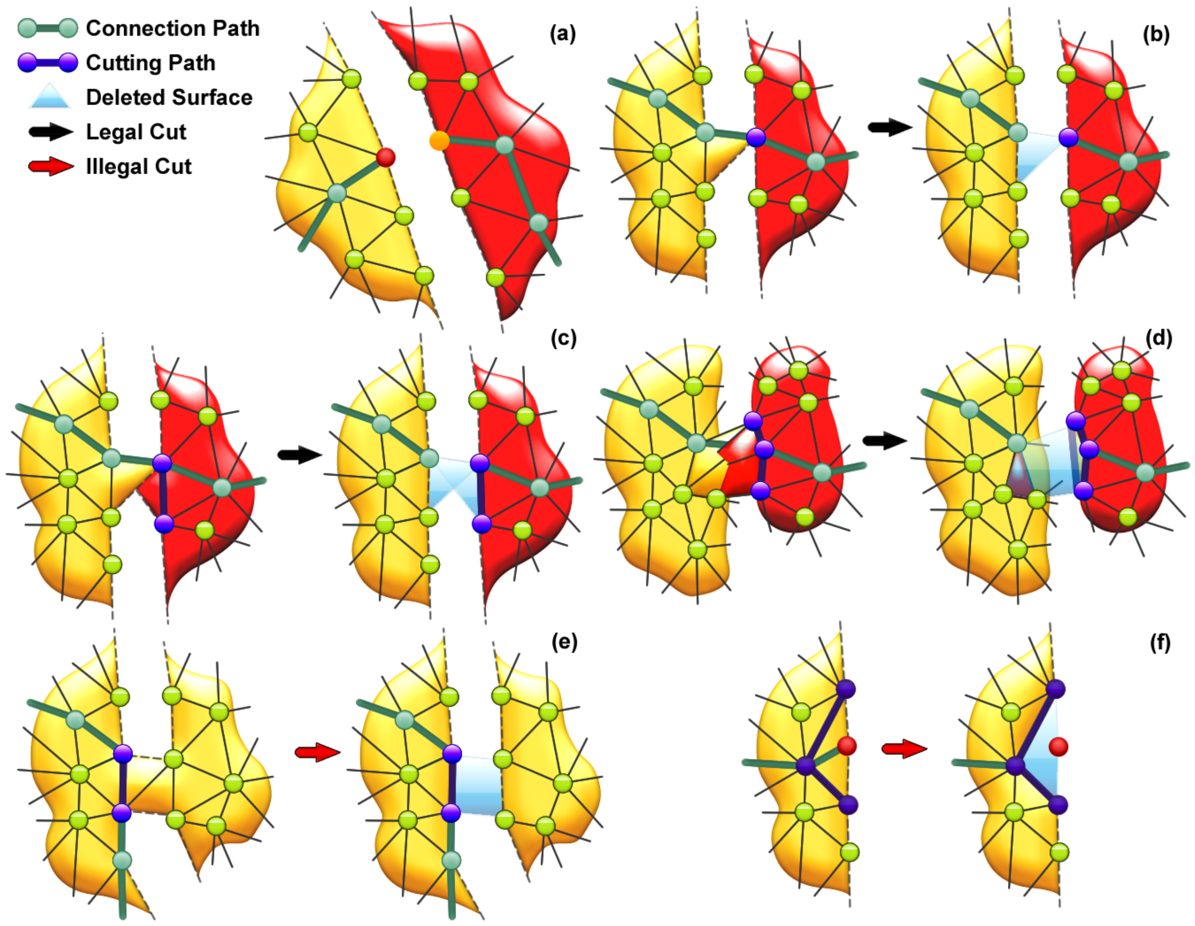


Figure 5.11: Cutting connection path: *(a)* $\mathbf{v}_x$ and $\mathbf{v}_{opp}$ with their connection path; A cut of *(b)* length zero and *(c)* length one from boundary to boundary; *(d)* a circular cut of length three through a self-intersecting surface; *(e)* an illegal cutting path that would not interrupt the connection path; *(f)* an illegal cutting path that would detach $\mathbf{v}_x$ from the surface.

5.3 Additional Usage

5.3.1 Mesh Distance Processing

Edge-Wise Distances: The presented semi-local data structures resemble very general mesh processing tools. They, for instance, allow easy implemention of the crumb and the bottleneck filter from section 4.2.2 on filters. A crumb can be investigated by adding a vertex of a mesh segment into a VF or a MEF as a starting point. If the front is annihilated before the maximum crumb size is reached, the segment is identified as a crumb. For a bottleneck a MEF simply needs to be expanded twice. If a collision takes place, a bottleneck has been identified.
Geodesic Distances: The MDF allows for geodesic mesh processing, which can be used for unwrapping (see section 3.2.2) and mesh segmentation (see section 2.3.1.3). The MDF also has obvious potential in geodesic applications, where on-surface distances are investigated.

5.3.2 Normal Estimation

Common normal estimation processes mostly attempt to calculate normals at a local point level (see section 2.3.3). Noise and outliers often cause such approaches to fail. The GCS process is very capable in creating soundly oriented surfaces at *locally* challenging point constellations. With the presented twist solving mechanism the process is also capable in creating *globally*

sound orientations. Since the orientation process is superior to most approaches, a GCS surface estimate can be used as a normal estimation process.
Here, a surface estimate is created. Then the process iterates through all points in $\mathscr{P}$. For any point the closest three vertices are calculated. Depending on the respective distances to these vertices a weighted average of their normals is built to estimate the normal of this point.
The calculated $\mathscr{P}_{ori}$ can then be used in combination with a distance function based surface reconstruction approach (see Fig. 5.12).

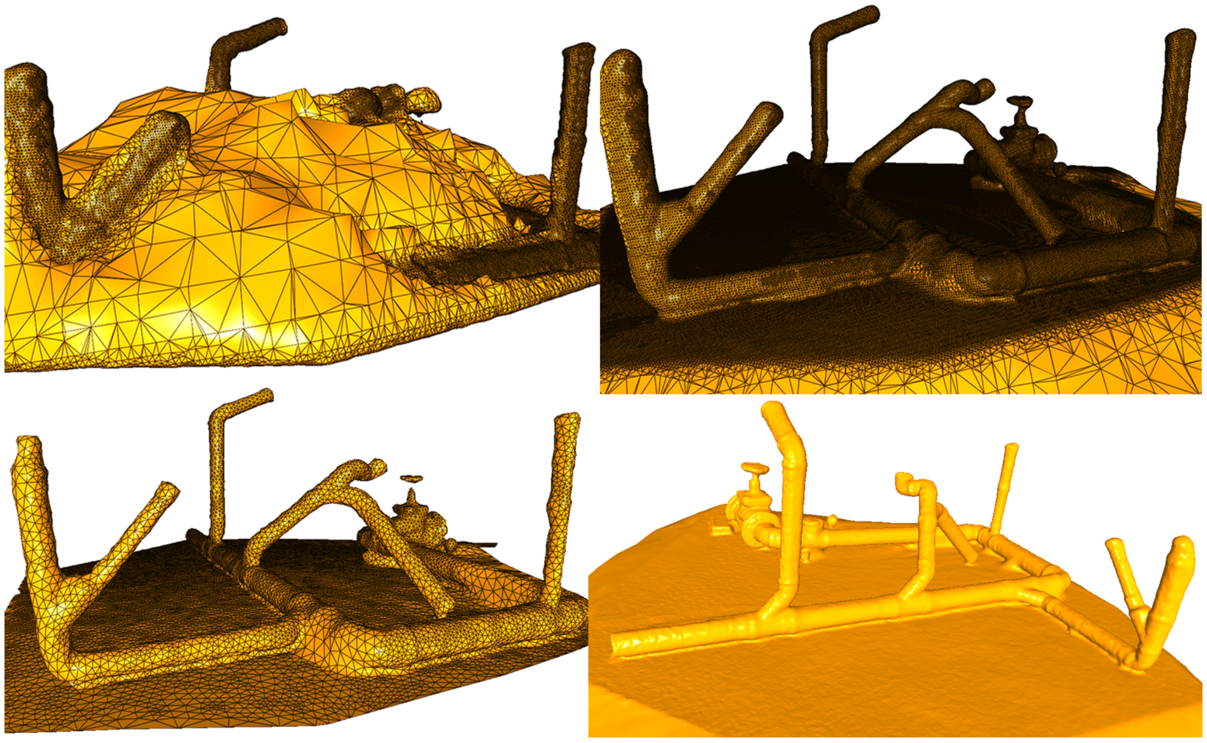


Figure 5.12: The Heating Pipes model contains noise and outliers. Reconstruction of the model with *Poisson* surface reconstruction using a *nearest neighbor* based normal estimation (top left). GCS reconstruction with the twist solving process (bottom left). Poisson surface reconstruction with the GCS based normal estimation (right column).

5.4 Results

Complexity Analysis: In all upcoming complexity discussions concerning VF, MEF, and MDF, the algorithms are considered to be performed on a flat infinite regular grid, where every vertex is connected to exactly six neighbors and all edges in the grid are of the same length. The benefit of this assumption is that the VF, MEF and MDF expand almost the same way – that is to say circular. In this setting, the number of processed surface elements – area of a circle – and the length of the front – circumference – are easy to calculate. Also, cases such as front annihilations, collisions and fronts reaching boundaries do not have to be consideration.
Shortest Path Calculation: The calculation of the shortest connecting path nicely demonstrates the advantages of the presented semi-local processing principle. If the vertices under investigation are not connected, only the smaller surface segment is fully processed and the calculation is thereby held as local as possible. When determining a connection path of a length d_{path}, the maximal memory consumption is the one needed for the two initial circular fronts whose radii are half of the size of the path length:

$$2 \cdot (2\pi \frac{d_{path}}{2}) \tag{5.1}$$

The processed area is the area of the two initial circular areas, plus the ones of the recursive descent:

$$2 \cdot \pi(\tfrac{d_{path}}{2})^2 + 4 \cdot \pi(\tfrac{d_{path}}{4})^2 + 8 \cdot \pi(\tfrac{d_{path}}{8})^2 + \ldots = \sum_{n=1}^{\infty} 2^n \cdot \pi(\tfrac{d_{path}}{2^n})^2 = 1 \cdot \pi(\tfrac{d_{path}}{1})^2 \tag{5.2}$$

The first summand in the formula above represents the processed area of the initial two circles, which are processed to find the first overlapping point; the second summand represents the circles for searching the next two overlapping points, processing four circles. The following summands in the series represent deeper recursion levels. For every recursion level the circle radii are halved, the number of circles doubles while the area to process is therefore halved as a result. When infinitely proceeding this recursive descent – assuming the distance to be continuous – the last or n-th summand represents an exponentially receding area converging toward zero. When adding up all summands, this last summand is the missing fraction to a sum of one. Assuming an infinitely receding area, the addition of all fragments of the series results to one.

The distance is based on mesh edges, thus, it actually is a discrete instead of a continuous value. Therefore, the processed area is slightly smaller than the formula suggests, depending on the number of edges in a calculated path. The longer a path is edge-wise, the smaller is the area of the last summand in relation to the calculated processed area in total.

In both formulas, the calculation effort eventually only depends on the length of the path and is independent of the mesh size. With a memory consumption proportional to the square root of the processed surface area – assuming the front line to be an actual 1D contour – the operation has a very small memory footprint. The runtime complexity of all the fronts is $O(n \log n)$, where n is the number of processed mesh elements. All n elements are processed and accessed a constant number of times and balanced trees are used for all element accesses $O(\log n)$.

Since the same surface is processed multiple times, the presented shortest path calculation has a bigger constant c in runtime than the *Dijkstra-based* shortest path calculation algorithms mentioned in the introduction. However, this advantage is achieved by storing all visited vertices. Therefore, their memory consumption is proportional to the processed surface area, instead of being proportional to its square root. Additionally, the presented path calculation could be optimized by taking advantage of the search direction as in the A^* algorithm. Additionally, the already calculated fronts of previous recursive search levels could be used to narrow the search domain.

Minimal Edge Front: The presented MEF is a very fast way for gaining edge-wise distances and other mesh connectivity related information. On average for 1000 different starting vertices, a MEF needs $0.343s$ to visit all $438K$ vertices of the Stanford Dragon model, while the maximum edge count is 3069. The same test performed with the VF needs $0.531s$ on average. The MEF exploits the properties of a 2D surface for its efficiency by using a minimized directed front line that represents a 1D contour. Only elements in front expansion direction are processed and the front is minimized, thereby many elements are skipped during the process.

Additionally, the connected and sealed edge-front is a more meaningful representation of the surface area under investigation. An edge-front can detect and localize collisions and due to the straightened front line additional surface aspects can be set in relation to the front.

Minimal Distance Front: Geodesic distance calculations with an edge-front can be performed with the MDF. Using on-surface distances makes the MDF expansion independent from the given mesh triangulation and it performs equally well even on challenging triangulations, as illustrated in Fig. 5.13.

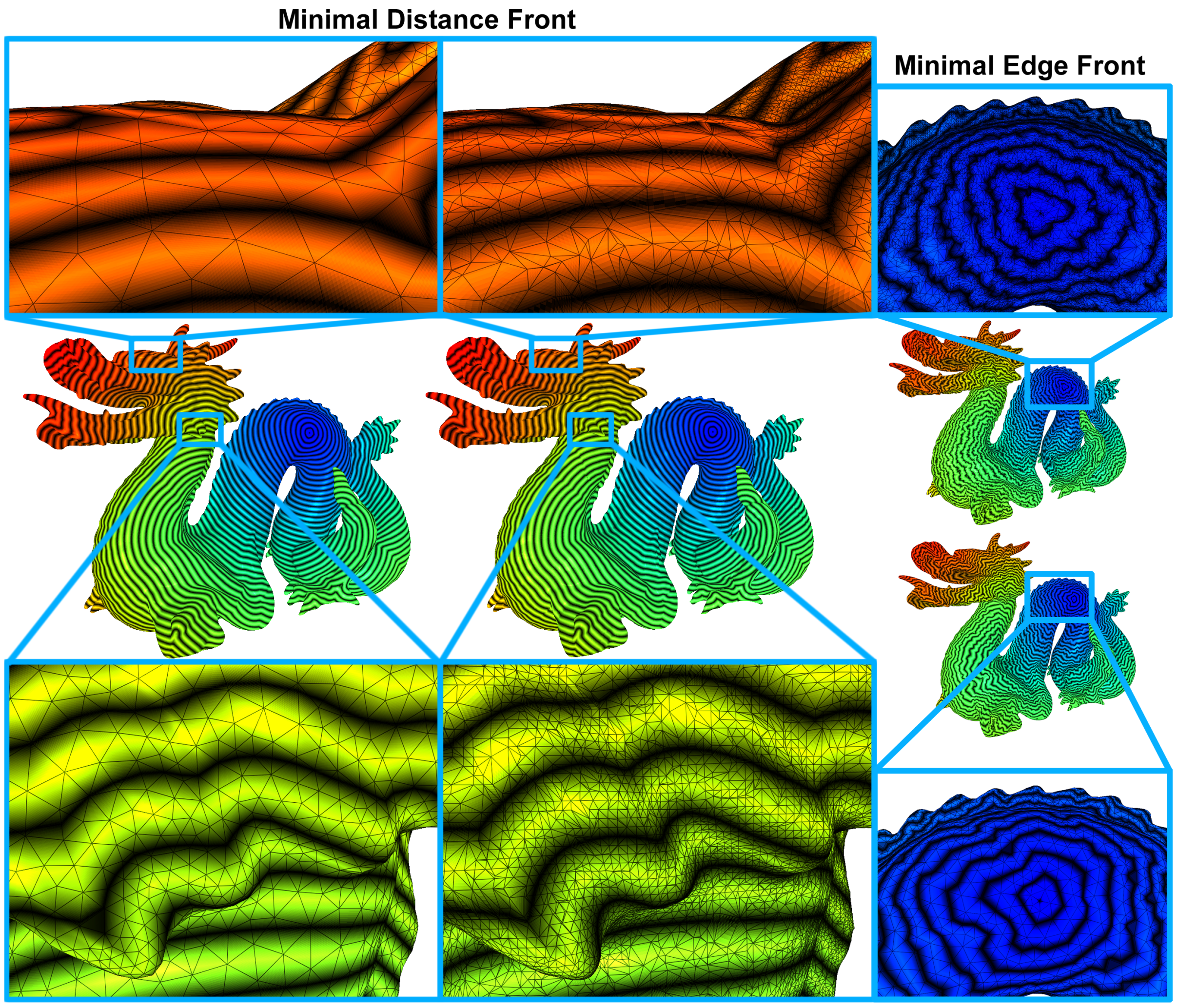


Figure 5.13: An illustration of a distance field on a reconstruction (# triangles: 100K) of the Dragon model (first column) and the original (# triangles: 871K) Dragon model (second column). Although the Dragons expose very different triangle distributions and shapes, the local distance lines progress nearly equally on both models. The same demonstration for the MEF (third column) exposes very different progressing fronts for the reconstructed (bottom) and original (top) model already at the starting point.

If a vertex cannot be attributed to a corridor in the MDF distance calculation, the fallback solution uses the distance to the vertex where the initial expansion took place. This distance estimate is likely to overshoot the actual *front-to-vertex* distance. Since only minimum distance expansions are selected, these possibly oversized estimates are less likely to be chosen. This provides the distance approximation process with a certain self-correction property, thus making it more robust.

If the MDF is performed 1000 times on the Dragon model, it takes 1.420*s* on average and uses only 2334 edges at its maximum. The latter measurement confirms a more straightened circular front, since fewer edges are required. The MDF is less efficient than the MEF, since it additionally involves the distance calculation. Also, the MDF with its anticipated distances always needs to be one step ahead of the current front. Additionally, the distance anticipation has to be performed for all expandable edges, although many of them are likely to be skipped and never to be expanded.

However, the presented implementation has been built on top of the MEF implementation, leaving quite some potential for optimization. For instance, the costly distance calculation is done twice for every vertex, once for anticipated distances, and again when actually adding

Model (# triangles)	time in sec	difference in mesurement
Hand (76K)	0.11	0.115%
Happy (1.1M)	2.16	0.083%
Statue (10M)	23.5	0.112%
Asian Dragon (7.3M)	17.2	0.077%

Table 5.1: The MDF was performed on four different models. The average time for the MDF to visit the entire mesh surface from 1000 different starting points was measured. Also, the distance differences for 1000 randomly picked pairs of vertices were measured with the calculation performed twice, once starting from the one and once from the other vertex.

them to the front. This redundancy can be removed by saving the initial calculation in another container, for instance.

The front-to-vertex calculation heuristic assumes that the path from the current front to new vertices is always a straight line. This heuristic fails if two or more triangles are passed which expose curvature. By using the available $\mathbf{d}_{start}$ vectors to project the shortest path onto those triangles the correct path could be determined, nevertheless.

Despite these imperfections the presented implementation of the MDF calculates entire distance fields even for meshes consisting of millions of triangles within a few seconds and distance calculations of the same path on average differed in the one per mill range, as shown in Fig. 5.14) and Table 5.1. Assuming this latter estimate represents the actual distance error range, the approach does come close to the accuracy of Surazhsky's approach, while not losing accuracy on complex models, such as the Happy Buddha, as the FMM approach does (these statements refer to the measurements presented in [SSK$^+$05]). Additionally, the already fast approach has major potential for runtime optimization.

Solving Twisted Surface: Twist resolving in an iterative refinement algorithm is a novel addition to the GCS approach.

The runtime complexity of the algorithm without the presented solution is $O(n \log n)$ for the average case as discussed in chapter 3 with the basic GCS approach, where n is the number of vertices in the final mesh. In the following, the complexity of the presented operations is discussed informally, *i.e.*, several assumptions are made, which are reasoned, but would need formal proving to satisfy the standards of a mathematical proof.

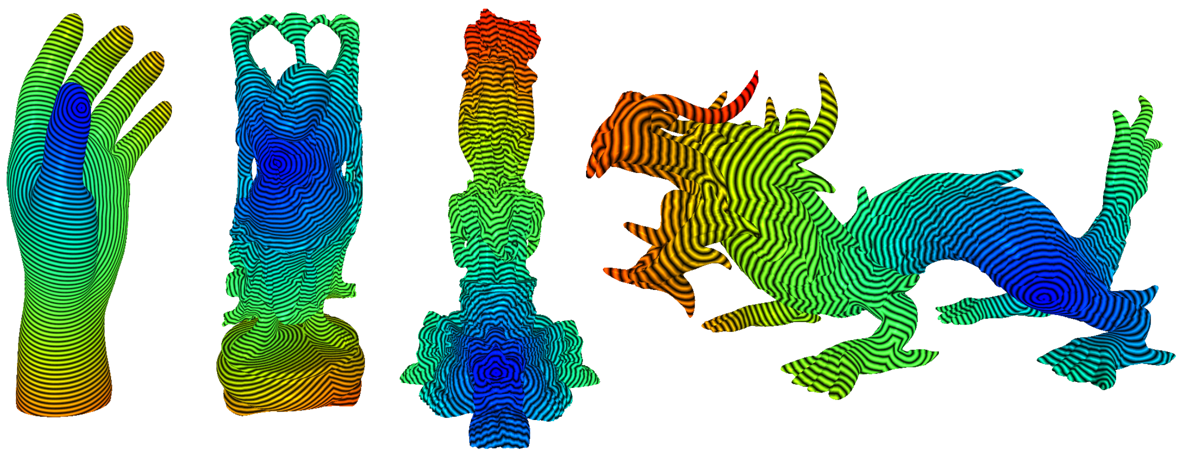


Figure 5.14: The Hand, the Happy Buddha, the Thai Statue and Asian Dragon model with a distance field calculated with the MDF.

If the twist detection is assumed to work correctly and the unknown surface is orientable, then – once detected – twists are resolved and cannot reappear, which strongly limits the number of twists that can realistically occur in a reconstruction process.
The worst case concerning memory and runtime complexity would take place at the end of the algorithm if both twisted surface segments have the same size of $\frac{n}{2}$. In this case, the searching mechanism visits the biggest possible number of vertices. Since all data structures within the MEF have a memory usage proportional to number of edge elements in the front, the memory complexity of the MEF is $O(\sqrt{n})$, assuming that the dimension of the front is one and it expands uniformly, *i.e.*, non-fractal, over the surface.
The runtime complexity can be determined by the number of vertices visited while searching the connection path and the number of search passes that are performed until the regions are separated. The search for a cut along the connection path and the switching operation are both done in addition to the search for the connection path, but both do have smaller complexities.
The number of vertices visited by the search operation is proportional to n and the search has to be accomplished for all incorrect connections that require cutting. Since the gap between the differently oriented surface segments can be assumed as being one-dimensional and not exposing a fractal structure, the number of possible wrong connections is proportional to $\sqrt{n}$. This creates an overall worst case complexity of $O(n^{\frac{3}{2}})$.
However, that a twist in a huge surface area remains undetected throughout the entire process is very unlikely. Twists are detected at a relatively constant stage in their development. Thus, in the average case the effort of the twist resolving process does not depend on n, but twits are rather resolved during the early stages of the mesh development.
The Stanford Dragon model serves as a good example for a point cloud that should not produce twisted surfaces at all, the Vault model is a suitable example for a simple twisted surface, and finally, the complex Heating Pipes model includes noise, outliers, and non-uniform sample densities which require multiple twist resolvings. Finally, the algorithm is tested with an extreme model concerning twists – the point cloud of a Möbius strip. All tests were performed without and without enabling the presented twist resolving step (see Table 5.2 and Fig. 5.15).
The Heating Pipes model was run using many different random seeds leading to different numbers of resolving processes, but finally a soundly oriented surface was always created. This reliability is achieved through the use of mesh connectivity rather than the geometric properties of the vertices. The maximum number of twist resolving processes during the Heating Pipes model reconstruction was three. The impact on runtime is obviously beneath the standard deviation of the processes duration (see Table 5.2), since the Heating Pipes reconstruction was even faster in the test including the twist resolving process.
To prove stability and robustness of the presented method, it was also applied to the toughest conceivable model in relation to surface orientation, the non-orientable Möbius strip. For the Möbius strip the twist resolving process was started 274 times until the demanded vertex resolution was reached, although this had a significant impact on the overall runtime of the process, it still performed soundly.

5.5 Conclusion

The presented edge-front data structures show great potential for computer graphics applications. The edge based front line allows the differentiation of events, such as collisions. Surface related properties can be set into relation to the front, such as the distance of vertices. The edge-front also exploits the characteristics of a 2D surface to gain efficiency. Some extensions to the edge-front model might be "static edge path elements" that do not expand, but confine the space in which

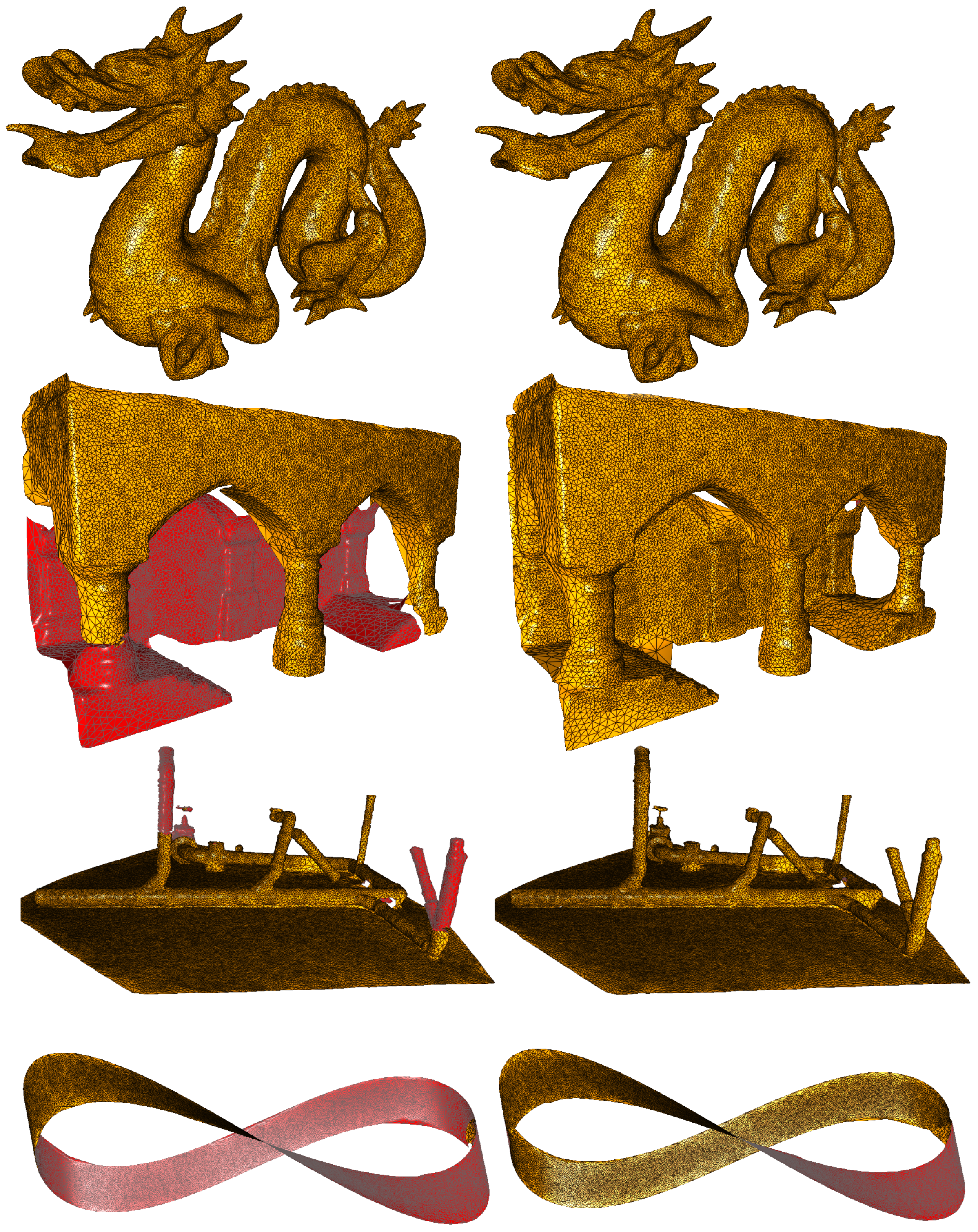


Figure 5.15: The Dragon, Vault, Heating Pipes and Möbius strip model reconstructed *without* (left) and *with* (right) the twist solving mechanism (also see Table 5.2).

Point Cloud (# samples)	# triangles	time without twist solving	time with twist solving	resolving processes
Dragon (438K)	100K	0:0:51	0:0:51	0
Vault (368K)	40K	0:0:16	0:0:16	1
Pipes (918K)	100K	0:1:13	0:1:12	3
Möbius Strip (163K)	40K	0:0:13	0:1:16	274

Table 5.2: The table shows reconstruction results of different models *without* and *with* the twist resolving method. The last column shows the number of performed twist resolving processes.

the edge-front expands. For instance, in the connection path calculation the edge-fronts at the point when an overlapping vertex is found could be saved in that way. When calculating the next recursive search level, the saved edge path of the previous level could be used to confine the search space, thus making the search more efficient. A disadvantage of the edge-front data structure is its complex implementation in comparison to vertex based front.
The geodesic calculations performed with the MDF proved the potential of the presented processing strategy. The data structure is virtually parameter free and can easily be altered for individual problem cases. A front can be started from a point as well as from a line and it can deal with challenging triangulations and *non-solid* meshes.
Nevertheless, the presented MDF is not yet a fully-fledged geodesic algorithm. In the presented form, the algorithm cannot be used for arbitrary positions projected onto a mesh, but only for vertices. The same holds when starting the algorithm from a line. Here, only existing mesh edges can be used. To start from arbitrarily shaped lines, as in [BK07], a mechanism for adding the required edges to a mesh would be needed. To calculate actual shortest distance paths independent of the existing edges and triangles of a mesh, the introduction of *temporary vertices* and *temporary edges* could be considered. The necessary vectors $\mathbf{d}_{start}$, which represent the on-surface projected path back to the starting point, are already available in the current implementation. With this additions to the presented algorithm, it could be fully compare with other geodesic algorithms. However, this would exceed the scope of this thesis.
When using the MDF for *unwrapping* in *texturing* or for a *surface segmentation*, an alternative collision behavior might be reasonable. If fronts would not merge at collision, but remain and build a *collision line* instead, a MDF would always contain only one single continuous edge-front. Using the collision line as a cutting line, a surface could be cut into one single connected sheet to be fitted into a *texture space*. For this task it would also be reasonable to add curvature into the expansion selection process to avoid flat surface regions from being cut.
As discussed an iterative refinement process for surface reconstruction offers a lot of advantages. However, up to now, this process was limited to point data which do not cause twists in the produced surface. With the presented method this limitation is removed, making iterative refinement approaches a yet more universal reconstruction tool.
The MEF delivers a very general solution to the problem of separating two surface areas, that does not depend on geometry based heuristics. The method is based on the assumption that surfaces with different orientations can always be encountered at a boundary edge. This assumption can fail if two entirely separate objects are contained in one point cloud or if the only crossing of two differently oriented surfaces is a self-intersection, which is unlikely, but possible. If a point cloud contains structures that are represented only by a few points, it may happen that a twisted surface cannot be recognized due to that low resolution.

Furthermore, the process has shown weaknesses when solving twists that are caused by thin structures – surfaces with contradicting orientations that lie close together – which the approach covered as one surface at initial stages. These structures can then lead to greater calculation times. Nevertheless, finally, twists are resolved correctly.

Chapter 6

Growing Surface Structures

In previous chapters, GCS approaches have been proven to be efficient surface reconstruction algorithms. GCS can easily be adjusted to a given problem, such as reconstruction, by inventing simple local learning rules, while still remaining simplistic and robust.
The *growing surface structures* (GSS) algorithm [AB13] (see Fig. 6.5 and Alg. 12) is a major conceptual change in the GCS approach. Instead of “adjusting” the learning behavior, the central learning scheme is shifted from optimizing the distribution of vertices to the creation of a valid surface model. Where in former GCS approaches the created topology is only implicitly represented in the process, it is explicitly integrated and represented in the refinement process of the GSS approach. Here, the closest surface structure, such as a vertex, an edge or a triangle is determined for a given sample and the actual *sample-to-surface* distance is measured. With this additional information the adaptation process can be focused on the created topology.

6.1 Introduction

The foundation of the placement of vertices in the GCS algorithm lies in the competitive learning strategy of the *k*-means clustering approach, which aims to create sensible vertex distributions. This placement is kept in the SOM algorithm, but extended by moving neighboring vertices, integrating topology guided movements. The vertex distribution is stabilized by an underlying topology. If the actual result of the process is the vertex distribution, as in vector quantization, this focus on vertices is a reasonable choice. If, however, the result is represented by the created surface and its topology, as in surface reconstruction, the inherited focus on vertices might discard valuable surface related information.
GCS uses a refinement strategy for reconstruction, where a current surface estimate $\mathscr{S}$ is successively refined to make it progressively more similar to $\mathscr{S}_{phy}$. Improving a process working with such a strategy, means to minimize the refinement operations needed to make $\mathscr{S}$ more similar to $\mathscr{S}_{phy}$. Since a newly created surface is built on older surface stages, a closer surface estimate reached earlier provides later refinements with a superior basis. $\mathscr{S}$ progressing toward $\mathscr{S}_{phy}$ can be measured as the distance of corresponding points, as in the surface fitting problem, and more importantly as how topologically correct $\mathscr{S}$ represents $\mathscr{S}_{phy}$.
To minimize the number of refinement steps needed, the adaptation operations need to gain the maximum information out of every sample and avoid incorrect adaptations. Information about $\mathscr{S}_{phy}$ in an iterative refinement strategy is derived as a combination of a current estimate $\mathscr{S}$ in relation to a given sample. However, instead of putting samples in relation to a surface $\mathscr{S}$, the GCS algorithm sets them in relation to a vertex distribution and due to this limited perspective discards valuable information.

The GCS approximation can be set to follow different aims, such as a likelihood distribution (signal counters), an error minimization (distance error), and a topology optimization, which each lead to different results.

6.1.1 Likelihood Distribution

Vertices represent a *likelihood distribution* if for every given vertex $\mathbf{v} \in \mathscr{M}$ the likelihood to be the closest neighbor to a randomly chosen sample $p \in \mathscr{P}$ is equal. If vertices being closest neighbors are seen as the result of a probability experiment, this approximation resembles *entropy maximization*. This means that the information carried by any given sample is of the same importance to the process. Such representations are especially important in *pattern recognition* and *statistical analysis*.
To implement a likelihood distribution, every vertex carries a signal counter. To approximate the likelihood distribution (see line 10 in Alg. 4), these counters are simply incremented when a vertex is closest to an input sample. If a new vertex is added (see line 14 in Alg. 4), the highest error term refers to the space where most samples share the same vertex.
Since older signals tend to be less representative, all signal counters are decreased by a certain factor at every iteration cycle (see line 11 in Alg. 4). By using a likelihood distribution signal counters can also be used to determine misplaced vertices in spaces that contain few or even no samples, since their signal counters are very low due to the constant decreasing. This concept has therefore been used in most implementations for surface reconstruction.
These algorithms, however, determine likelihoods of vertices, instead of the likelihood of a surface. If, for instance, a flat surface is approximated, the algorithm creates lots of vertices, which are in proportion to the number of samples, although the area could be accurately approximated with a few triangles only.

6.1.2 Distance Minimization

If the approximation is changed to account for a *quantization error*, vertices are placed to expose the smallest Euclidian distance to the samples in $\mathscr{P}$. If the samples $\mathscr{P}$ are equally distributed, the goals of a likelihood distribution compared to ones of a *distance minimization* are nearly equal.
If, however, some regions are represented by a denser sampling than others, these regions are represented by fewer vertices in the distance minimization scenario. The error, which is measured as the Euclidian distance, can be lowered more significantly in regions where samples lie farther apart. Thus, vertices are more likely to be added there.
This approximation is typically used for vector quantization in data compression. To implement this behavior, every vertex carries an error value *err* which is increased by the distance or the

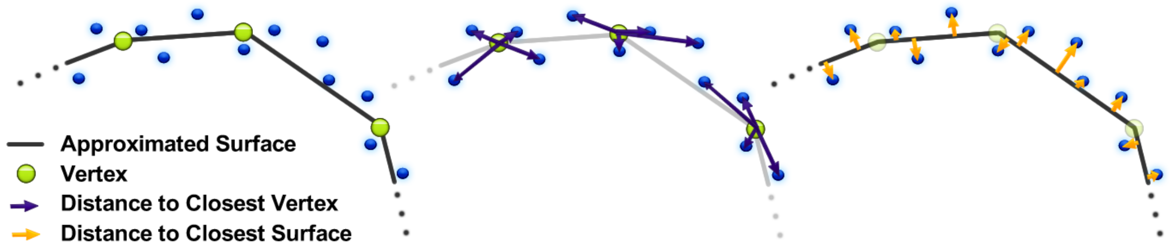


Figure 6.1: Samples and an approximated surface (left), distance between vertices and samples (middle), distance between surface and samples (right). In the case of a surface approximation, using the distance from samples to the surface is more sensible.

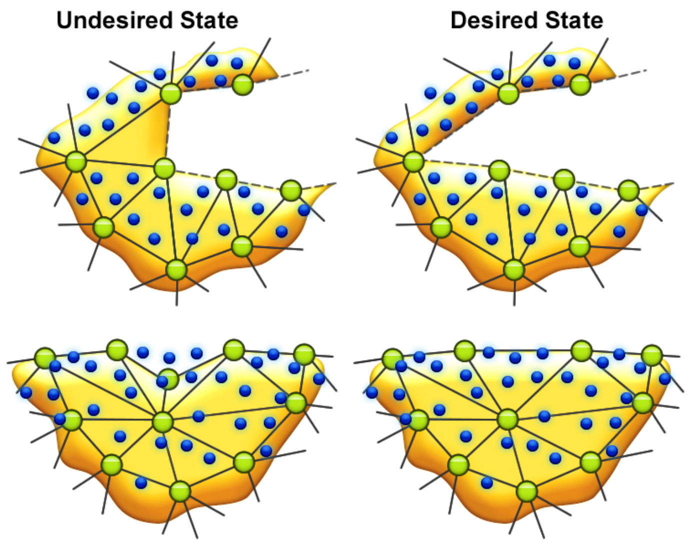


Figure 6.2: Problems of vertex focused approximation errors: Two surfaces with the same vertices, the same samples, and thus, the same *sample-to-vertex* approximation errors. Both surfaces expose an undesired (left) and a desired (right) solution. A triangle placed in an empty space (top) and an incorrect dent in the surface (bottom).

squared distance between the winning vertex and the given sample. The highest approximation error refers to the space where the samples lie farthest away from a vertex, thus a new vertex is added there.

In contrast to the previously discussed likelihood values in the form of signal counters, removing a vertex with low distance error would make no sense, since these vertices indicate that they are well placed. If this approximation error is used, the deletion process for topologically incorrect vertices needs to be handled separately.

Despite this disadvantage, minimizing the distance error might be more convenient for placing a surface as close as possible to given input samples. However, the presented approximation minimizes the distance to the vertices, instead of the one to the surface (see Fig. 6.1).

So far, the idea of involving the surface approximation error in the refinement process has been addressed only indirectly. In the presented SGC approach [AB12c], the curvature adaptation compares the average surface curvature to the one of a winning vertex and curved areas lead to higher signals (see line 14 in Alg. 5) and thus more subdivisions in such areas. In [JIS03], vertices additionally contain normals and the algorithm counts how many times these normals are adapted to increase subdivisions in such areas.

These changes lead to an implicit representation of the surface approximation error within the algorithm, since curved surface regions need more subdivisions to be correctly approximated. But the surface approximation error itself is not explicitly represented.

A more direct approach was presented in [MSP$^+$08], where distances of newly added triangles to the points in $\mathscr{P}$ are directly calculated. If distances can be lowered by an edge swap operation, it is performed. This leads to a certain improvement of the geometrical quality of the approximation. At the same time, however, it discards one of the fundamental advantages of the GCS algorithm, the independence of the sample set size, since in this approach $\mathscr{P}$ is accessed directly for the additional distance calculation.

6.1.3 Topology Optimization

In the presented algorithm development, the SOM introduced topology. A topology can increase the performance of creating smoother and more stable distributions of vertices. But the created

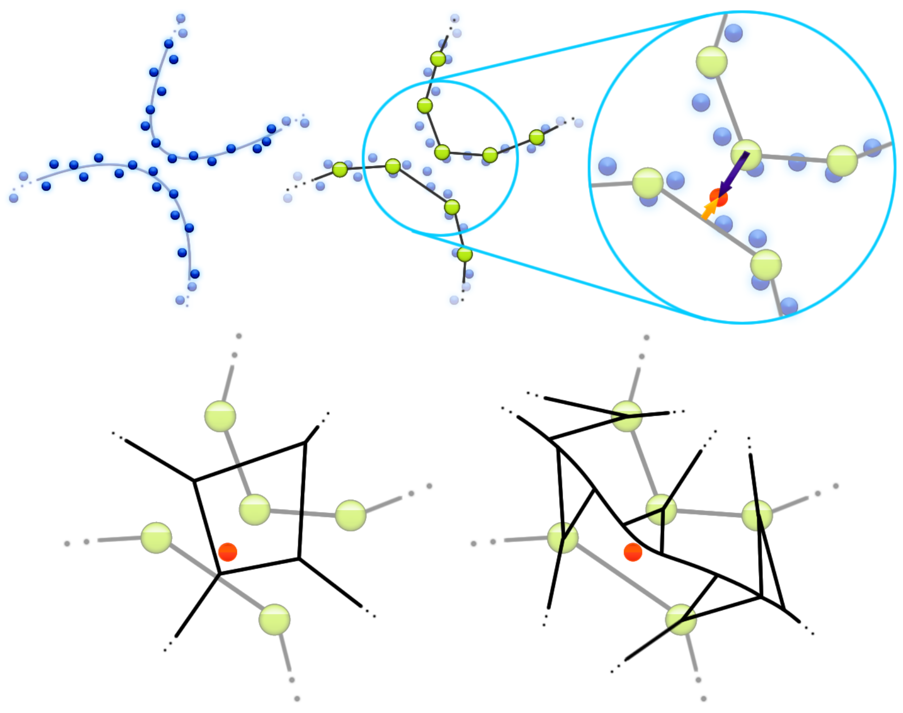


Figure 6.3: Samples that originate from a curved surface (top left); A fitting approximation of this surface and a magnification showing the sample-to-vertex distance and the sample-to-surface distance (top right); Vertex Voronoi regions, which assign a sample to the incorrect surface area (bottom left); Surface Voronoi regions, which assign the same sample correctly (bottom right).

topology can also be used to map data of a high dimensional input space onto a space of lower dimension to visualize or analyze data with less computational effort, as in classical *dimensionality reduction*. It can also be used to directly reconstruct the subspace the samples in $\mathscr{P}$ originate from, as in *surface reconstruction*. Where the SOM as well as the basic GCS algorithm use a static surface topology, the SGC approach can approximate arbitrary topologies. A significant problem of all GCS based algorithms concerning topology is the missing representation of the actual surface in the adaptation process. With the same constellation of vertices, however, different triangles can be constructed. Since the presented approximation techniques solely focus on *sample-to-vertex* properties, these different triangle constellations have no effect on the approximation behavior (see Fig. 6.2). As a result, many insufficient approximation states are simply invisible to the process and therefore cannot be treated.
Although the process refines an existing 2D surface $\mathscr{S}$, the surface is still represented as a collection of *Voronoi regions* of the vertices, since sample-to-vertex and not *sample-to-surface* distances are considered. This implicitly includes the assumption that the Voronoi regions of two connected vertices are not interrupted within their attached surface. But for close or complex shaped surfaces, this is not the case (see Fig. 6.3). Here, the actual representation of $\mathscr{S}$ becomes apparent, a permeable space of independent Voronoi regions.
Topology optimization and error minimization (surface fitting) are two different things. Error minimization tries to create the smallest possible sample-to-surface distances. Topology optimization is concerned with creating a topology with $\mathscr{S}$ that is as close as possible to the topology of $\mathscr{S}_{phy}$.
In order to be topologically correct, every point on one surface needs to have a unique equivalent on the other surface and vice versa, while neighbor relations are preserved. This means the shortest on-surface path between any two corresponding points on $\mathscr{S}$ and $\mathscr{S}_{phy}$ should always pass through the same corresponding points on both surfaces (see Fig. 6.4).

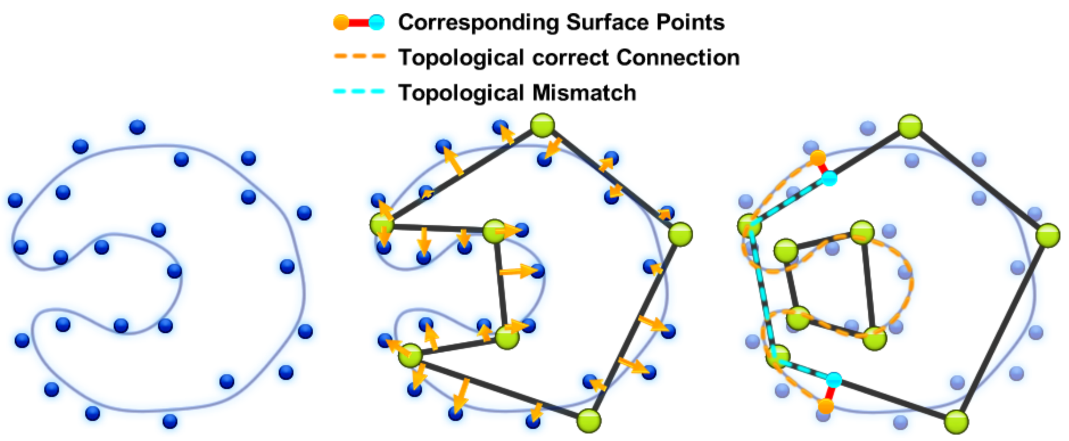


Figure 6.4: Surface and samples (left); An approximation of that surface and the sample-to-surface distances (middle); Another approximation of that surface exposing a topological mismatch, but having a similar distance error (right).

An approximation technique focusing on topology optimization during the refinement process has not yet been presented in the GCS process.

6.2 Approach

The GCS algorithm has proven to be a high quality surface reconstruction tool. However, the analysis of the algorithm has shown that topology is only created implicitly. In the following, changes to the general approximation concept and improvements that can be made based on these changes are presented.

6.2.1 Topology Focused Approximation

The basic algorithm concept focuses on placing vertices in positions likely to reach the chosen approximation aim, such as a likelihood distribution or a distance minimization. Up to now no setting has been presented to aim for topology optimization. To put the actually created surface topology into focus, the approximation error needs to be set in relation to the sample-to-surface distance. Instead of a vertex the process needs to find the closest surface structure. Setting samples in relation to a triangle or an edge gives rise to several different local surface modifications.

In the basic GCS implementation, either signal counters or error values are carried by the vertices. The surface structure element that most distance errors are measured toward and which is also the building block of the surface discretization is, however, a triangle. It is therefore the most sensible structure to carry the local approximation error values. The most reasonable place for the error value of any topology focused function approximation is always the simplex of the highest dimension in the GCS algorithm. This simplex is the building block of the subspace that is approximated and distance values toward these elements are the most significant for the current approximation state. Thus, they are the most valuable ones for decisions concerning modifications of the current subspace estimate. Using them allows for better judgments about current local approximation states and the choice of location for subdivisions.

The new approximation error also allows and demands for a differentiation between topology changing deletions to correct topologically misplaced surface structures and non-topology changing deletions that remove geometrically redundant structures from the surface. This

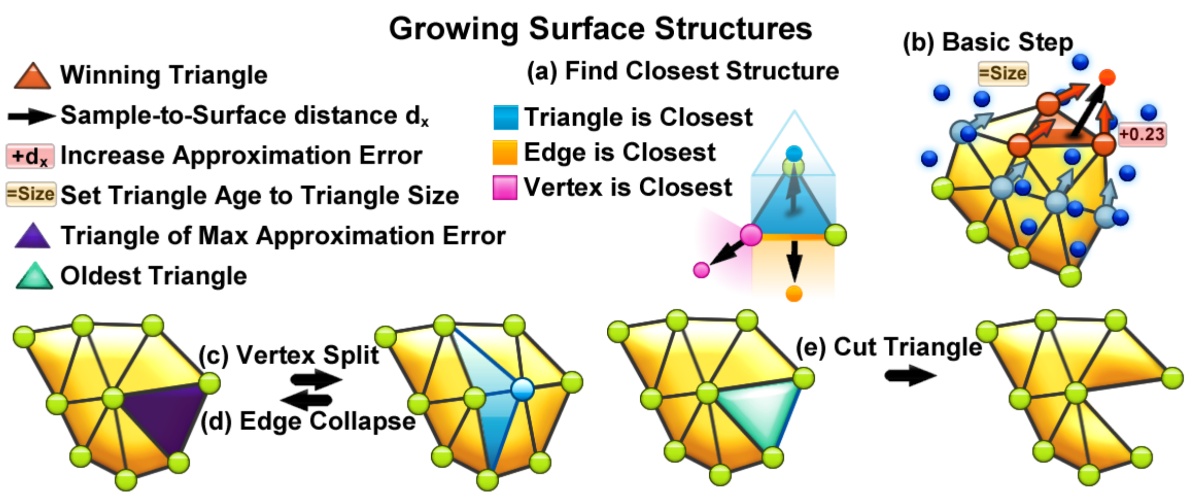

Growing Surface Structures
Winning Triangle
Sample-to-Surface distance d_x
$+d_x$ Increase Approximation Error
=Size Set Triangle Age to Triangle Size
Triangle of Max Approximation Error
Oldest Triangle
(a) Find Closest Structure
Triangle is Closest
Edge is Closest
Vertex is Closest
(b) Basic Step
=Size
+0.23
(c) Vertex Split
(d) Edge Collapse
(e) Cut Triangle

Figure 6.5: Growing surface structure: *(a)* the closest surface structure is searched for; *(b)* the basic step moves a triangle, increases its error values, and sets its age to the triangle size; The *(c)* vertex split and *(d)* edge collapse operation; *(e)* the surface cutting operation.

distinction is realized by adding an “age” indicator to every triangle, which is then used to determine topologically misplaced surface structures.

6.2.2 Implementation of Growing Surface Structures

With the conceptual changes presented above, additional and more accurate information about the current approximation state is available within the GCS process. This information can be used to create a better approximation in *sample-to-surface* distance accuracy as well as in topological correctness, as presented in the following implementation (see Fig. 6.5 and Alg. 12).

6.2.2.1 Search for Closest Structure

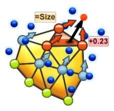
For runtime efficiency reasons the presented implementation does not use a triangle based spatial subdivision data structure, but still uses a vertex based *k*-d tree. By examining a number of vertices and testing their surrounding triangles, the closest structure to a given sample is heuristically determined. This heuristic might fail if a curved surface lies very close to a flat surface, but this case is considered to be rare (see Fig. 6.6). The new search process has three possible outcomes: a vertex, an edge, or a triangle (see *(a)* in Fig. 6.5 and line 6 in Alg. 12).

6.2.2.2 Surface Movement

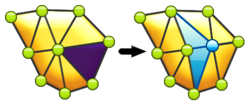
Instead of having only a vertex as a closest element, as in the basic GCS process, now it can additionally be an edge or a triangle. The type of movement is differentiated depending on sample-to-surface distances. The known distance d_x of a given sample $\mathbf{p}_x$ to the surface can be compared to the average sample-to-surface distance $\overline{d_{\mathscr{P}}}$. If d_x falls below a limit in relation to $\overline{d_{\mathscr{P}}}$, the movement is entirely discarded, since the sample already lies close enough to the surface (see line 10 in Alg. 12). If d_x is higher than lim_{skip}, but falls below a limit in relation to $\overline{d_{\mathscr{P}}}$, only the closest vertex is moved, since the surface can be considered generally correct, but the placement of the edges can be optimized (see line 11 in Alg. 12).
For larger distances all vertices of the given structure are moved toward the sample (see *(b)* in Fig. 6.5). The neighborhood movement (see line 15 in Alg. 12) is kept unchanged

Algorithm 12 Growing Surface Structure
(based on GCS Alg. 4 and additionally incorporating aspects of SGC Alg. 5 and the filter chain Alg. 7)

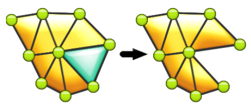
```
1:  Initialization
2:  repeat
3:      repeat
4:          repeat
                Basic Step
```



```
5:              Select random sample $\mathbf{p}_x$ of $\mathscr{P}$
6:              Find the winning structure $[\mathbf{s}]$ that exposes the smallest Euclidian distance $d_x$ to $\mathbf{p}_x$
7:              Increase the approximation error $err$ of $[\mathbf{s}]$ if $[\mathbf{s}]$ is a triangle and to all adjacent triangles if
                $[\mathbf{s}]$ is an edge or a vertex: $err_{new} = (d_x + err_{old}(k-1))/k$
                ~~Decrease signal counters of all other vertices by a fraction: $\forall sc_i(i \in \mathscr{M})\ \Delta sc_i = -\beta \cdot sc_i$~~
8:              Set the age $a$ of $[\mathbf{s}]$ if $[\mathbf{s}]$ is a triangle and of all adjacent triangles if $[\mathbf{s}]$ is an edge or a
                vertex to their initial age: $a = size(\mathbf{t})/\overline{s_{\mathscr{T}}}$
9:              Increase the age of all other triangles according to $\beta$: $\forall a_i(i \in \mathscr{T})\ \Delta a_i = \beta \cdot a_i$
10:             if $d_x$ in relation to $\overline{d_{\mathscr{P}}}$ indicates an adaption is needed: $d_x / \overline{d_{\mathscr{P}}} > lim_{skip}$ then
11:                 if Only a slight adaption is needed: $d_x / \overline{d_{\mathscr{P}}} > lim_{single}$ then
12:                     Set $[\mathbf{s}]$ to be the closest vertex in $[\mathbf{s}]$ to $\mathbf{p}_x$
13:                 end if
14:                 Move all vertices in $[\mathbf{s}]$ as much toward $\mathbf{p}_x$ as determined by the learning rate $lr$
15:                 Perform Laplacian smoothing on all neighbors $\mathscr{N}_{[\mathbf{s}]}$ of $[\mathbf{s}]$ as much as determined by the
                    neighbor learning rate $lrn$
16:             end if
17:             Coalescing
18:             Filter Chain
19:             Increment the iteration counter: $\Delta t = 1$
20:         until The basic step has been performed $c_{add}$ times: $t$ mod $c_{add} = 0$
            Vertex Split
```



```
21:         Select triangle $\mathbf{t}_x$ with the highest approximation error value: $\forall err_i(i \in \mathscr{T}) <= err_x$
22:         Select the longest edge of $\mathbf{t}_x$ and perform a vertex split operation.
23:         Split $err_x$ evenly between the old and newly created triangles.
24:         Set all triangle ages $a$ to their initial age: $a = size(\mathbf{t})/\overline{s_{\mathscr{T}}}$.
25:         Filter Chain
26:     until The basic step has been performed $c_{del}$ times: $t$ mod $c_{del} = 0$
        Remove
```



```
27:     Select triangle $\mathbf{t}_x$ with the lowest approximation error value $\forall err_i(i \in \mathscr{T}) >= err_x$
28:     if $\mathbf{t}_x$ exposes an edge $\mathbf{e}_{xy}$, which is geometrically redundant: $\mathbf{n}_x \cdot \mathbf{n}_y < max_{\nabla n}$ then
29:         Perform an edge collapse operation on $\mathbf{e}_{xy}$
30:     end if
31:     Select triangle $\mathbf{t}_y$ with the highest age $a_y$: $\forall a_i(i \in \mathscr{T})\ a_i <= a_y$
32:     if The age $a_y$ of $\mathbf{t}_y$ is too high: $a_y > max_a$ then
33:         Cut out $\mathbf{t}_y$ entirely.
34:     end if
35:     Filter Chain
36: until Approximation accuracy exceeds a certain threshold
```

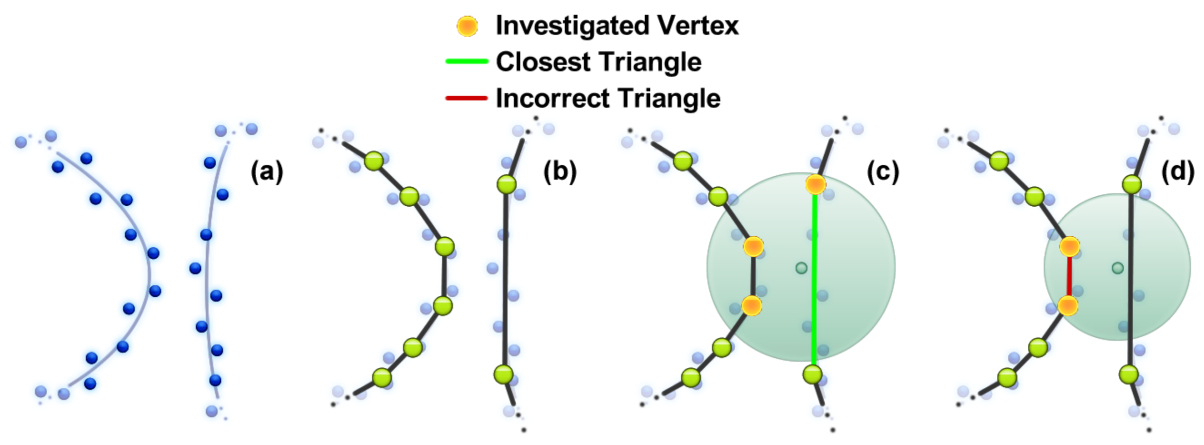


Figure 6.6: The search for a closest surface structure based on investigating a number n_v of surrounding elements of a vertex: *(a)* a sampling of a curved and a flat surface close to one another; *(b)* approximation of that surface; *(c)* the search heuristic works correctly for $n_v = 3$, where three vertices are investigated; *(d)* the search heuristic fails for $n_v = 2$, where the vertex connected to the closest triangle is not investigated.

and conducted with the Laplacian smoothing mechanism for all first neighbors $\mathscr{N}_{[\mathbf{s}]}$ of the examined structure.

6.2.2.3 Distance Error

A sample can either be closest to a vertex, an edge, or a triangle. If it is closest to a triangle, its distance error is changed. In case of an edge, this is done for both triangles connected alongside that edge. In case of a vertex, this is done for all triangles connected to this vertex.
If the error value was directly set to the given distance, all previous distance errors would be lost. If all distance errors were accumulated, distance errors of the beginning of the process, exposing huge distances, would solely determine the subdivision process. If all error values were just constantly decreased, as signal counters in a likelihood distribution, a constant distance error improvement over the entire surface would falsely be implied.

Therefor a formula is used (see line 7 in Alg. 12) where the influence of an error value halves after k additional updates. The following formula creates this local *half-life* λ for distance error updates:

$$\begin{gathered}(\tfrac{k-1}{k})^{\lambda} = 0.5 \\ k = \frac{1}{1-\sqrt[\lambda]{0.5}} \\ err_{new} = \frac{d_x + err_{old}(k-1)}{k}\end{gathered} \tag{6.1}$$

6.2.2.4 Refinement

Instead of a vertex the *triangle* with the highest approximation error is determined. Subdivision is performed with a vertex split operation using the longest of the triangle edges (see *(c)* in Fig. 6.5). The error value of each new triangle is set to the half of the error value of its predecessor (see line 21 in Alg. 12). Triangles ages are set to their initial value.

6.2.2.5 Deletion

The deletion process is one of the most important changes in the new algorithm. When using a sample-to-surface distance error, the error values can be used to locate triangles that are geometrically redundant (see line 28 in Alg. 12). In order to create a model representation that is as memory efficient as possible, these triangles can be deleted by an edge collapse operation of one of their three edges (see line 29 in Alg. 12).
The best triangle edge for the collapse operation is the one which is surrounded by triangles whose normals expose the smallest differences to each other. Collapsing this edge changes the surface geometry the least. This edge can be determined as the one with the highest dot product of the normals of its two vertices, since these normals are calculated based on the surrounding triangles. It is reasonable to set a curvature threshold to avoid decreasing the surface approximation quality when collapsing vertices that are actually exposing curvature.
In addition to the distance error value a triangle age is needed that indicates if a triangle reached the maximum age max_a. This is supposed to happen if the triangle has not been winning for a certain number of times γ. Those triangles are considered to be misplaced and topologically wrong. A misplaced triangle is detached from the rest of the mesh and then deleted (see *(e)* in Fig. 6.5). After the deletion process, as also done in previous sections, the mesh can be cleaned by the application of certain filters (see line 18 in Alg. 12).
The concept to locate misplaced constructions by an age is inspired by the GNG edge removal process (see line 24 in Alg. 6). For GNG the age increment is, however, constant. Since the likelihood for a triangle to win is proportional to its size, the aging process needs to be more differentiated. For every iteration the age of all triangles is therefore increased by a tiny factor β (see line 9 in Alg. 12), which relates to the overall number of triangles $|\mathscr{T}|$ in the mesh:

$$\beta = \sqrt[(\gamma \cdot |\mathscr{T}|)]{max_a} - 1 \tag{6.2}$$

Instead of increasing the age of the triangles by constant incrementation, as in the GNG process, it is done by multiplication, again allowing the use of a tumble-tree [AB10b]. The aging process reaches an upper bound, whereas the signal counter reaches a lower bound. This, however, is only done to resemble an *aging* process.
For all triangles selected during the basic step, whose error values are updated (see above), the age is renewed (see line 8 in Alg. 12). Since small triangles are less likely to be winners, the initial age of a triangle is its $size(\mathbf{t})$ divided by the average triangle size $\overline{s_{\mathscr{T}}}$. Thus, the average triangle starts with an age of one, while small triangles start “younger” and big triangles “older”. The deletion process now explicitly distinguishes distance driven and topologically driven deletions.

6.2.2.6 Finalization

One of the assets of the GCS algorithm is the fact that $\mathscr{S}$ is an approximation of $\mathscr{S}_{phy}$ at any time during the process – the algorithm can be stopped and resumed at any given time. With the novel surface distance approximation error a potential stopping point for the algorithm can be chosen more sensibly, since it can be set to an actual sample-to-surface error value.

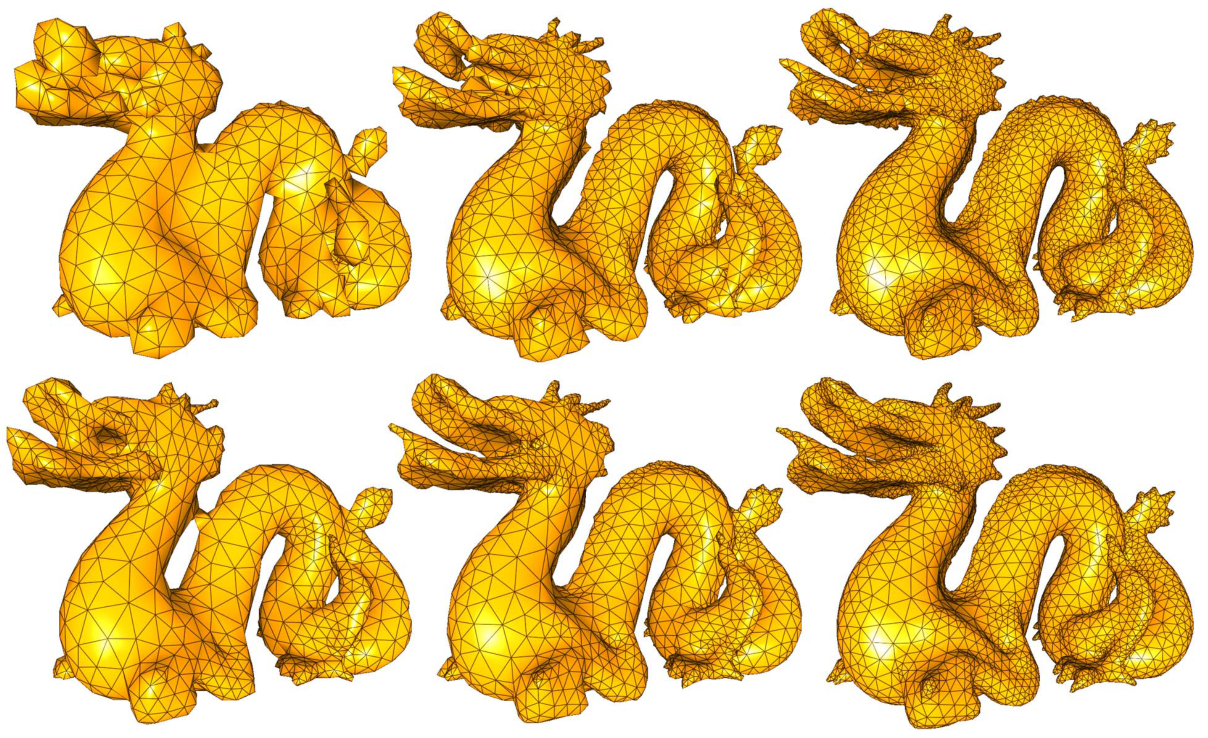


Figure 6.7: A progression series of the Dragon model from left to right with 2500, 5000 and 10000 triangles with the GCS algorithm (top) and with the GSS algorithm (bottom). With the new algorithm the surface converges faster toward the final topology.

6.3 Additional Usage

6.3.1 Unifying Sample Density

If the approximation of the basic GCS algorithm is set to distance minimization – using sample-to-vertex distance errors instead of signal counters – it can also be used as a sample distance unifying process. Reconstruction processes that rely on evenly distributed sample distances could pre-process point clouds with non-uniform sample densities.

6.3.2 Remeshing and Mesh Optimization

The GCS algorithm can be used as a remeshing algorithm. With the presented changes, the process can additionally use the available triangles more efficiently to create evenly distributed approximation errors. Potentially the GSS approach could be used to create a pure *mesh optimization* process where solely the approximation quality matters. However, this would mean to suspend all operations that increase the triangle quality. This idea is further discussed in the conclusion section.

6.4 Results

Different tests have been performed with the Stanford Dragon model as a good example for a point cloud that is relatively challenging by its shape and sample distribution, the Hand model exposes sharp features, the Asian Dragon and the Thai Statue expose a lot of curved areas, the Heating Pipes model includes some extremely noisy areas, non-uniform sample densities and open surface areas, the Happy Buddha has regions of surfaces lying close together. As a

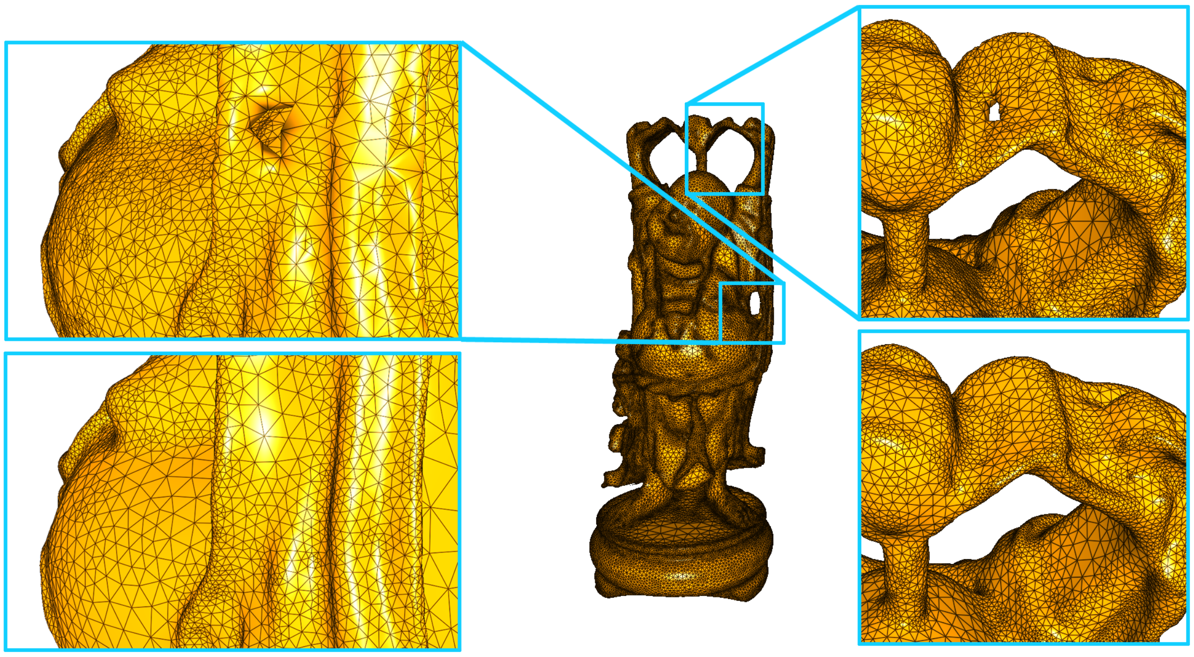


Figure 6.8: Some thin areas of the Happy Buddha model, reconstructed with 200K triangles with GCS (top) and GSS (bottom) algorithm. The new algorithm is able to build a correct topology in thin areas in an earlier algorithm stage.

comparison an extended GCS approach is used which incorporates the SGC topology adaptation (see section 3.1.4), but vertices are not adapting to curvature or sharp features. For the GSS algorithm the following parameter settings have been used:

Parameter Settings:

Symbol	Setting	Meaning
λ	9	Half-life of a distance error
γ	7	Allowed misses before deletion theshold is reached
max_a	10	Maximum age for triangles
$max_{\nabla n}$	0.9	Threshold before edge collapse
lim_{skip}	0.9	Distance threshold before mesh change
lim_{single}	1.2	Distance threshold before multiple vertex change
n_v	3	Number of investigated vertices in structure search

Theoretically, any surface would be correctly reconstructed with the SGC algorithm if infinite samples, memory, and time were available. A reasonable parameter to judge the efficiency of a refinement based surface approximation is the time that is needed to reach a certain accuracy. With the new approach, for instance, the topology of the Dragon model was approximated a lot faster, shown in Fig. 6.7.
Due to the advanced closest structure search, incorrectly assigned samples are less likely. This makes twists, caused by surface pulled inside out, less likely. Also, the approximation of surfaces close to each other profits from this correct assignment. With the presented method the number of required iterations to avoid permeating Voronoi regions is drastically reduced, as shown in Fig. 6.8.
Although the GCS algorithm is already quite robust when dealing with noise, it can be shown that *spikes* and rough surface gradients can be greatly reduced with the presented GSS method. The moving of entire *structures* proved to have a smoothing effect on the surface (see Fig. 6.9).

Model (# triangles)	GCS			GSS error=d_p			GSS error=d_p^2		
	time	dist	dist^2	time	dist	dist^2	time	dist	dist^2
Hand (20K)	0:0:6	3.19	4.38	0:0:7	4.29	4.11	0:0:6	4.70	5.00
Dragon (100K)	0:1:01	2.29	3.42	0:1:05	2.55	2.97	0:1:16	3.05	4.34
Asian Dragon (100K)	0:0:55	2.36	3.62	0:1:01	2.71	2.53	0:1:01	2.83	2.78
Statue (200K)	0:2:26	3.21	15.4	0:2:27	3.00	4.02	0:2:28	4.05	7.17
Buddha (200K)	0:2:30	1.48	14.1	0:2:38	1.89	11.7	0:2:40	4.02	63.0

Model (# triangles)	$n_v = 1$			$n_v = 5$			$n_v = 10$		
	time	dist	dist^2	time	dist	dist^2	time	dist	dist^2
Dragon (100K)	0:1:02	2.35	3.01	0:1:11	2.55	3.05	0:2:01	2.57	2.95

Table 6.1: Results for different models with the standard and the novel GSS algorithm. Also, different settings for n_v have been tested.

The SGC approach considered distance errors only indirectly by incorporating surface curvature into the process, whereas the GSS subdivision process is directly connected to actual distance errors. The advantage of this more direct approach is that subdivisions are more independent from a current surface development stage. This leads to a superior surface fitting, visible especially at the jaw of the Asian Dragon model. When comparing the behavior at sharp features, the explicit sharp features adaptation of the SGC approach builds a more continuous feature line. But the sharp feature is still properly represented in the GSS reconstruction, although no explicit feature line is created (see Fig. 6.10).
In Table 6.1, the new and the old algorithm are compared. Generally the old algorithm creates a lower average sample-to-surface distance, since it evenly distributes its subdivisions over $\mathscr{S}$, whereas the new algorithm focuses its subdivisions on areas of high approximation error. This is visible through a 25% decrease of the mean squared error. Especially for curved models such as the Asian Dragon and the Thai Statue this effect is very salient.
Although the search process is more complex, the extra time costs are nearly leveled by the discarded operations. During the reconstruction of the Dragon model 43.3% of the adaptations were discarded and a rate of 26.3% of surface movements were downgraded to vertex movements. If the square distance was used as approximation error, the results for both the average distance error as well as the square distance error were worse than the results of the GCS algorithm. The reason is that most triangles are used up to model tiny but steep curvature (see Fig. 6.11). In addition, triangles tended to clump even for low error half-lifes λ. For the Happy Buddha model

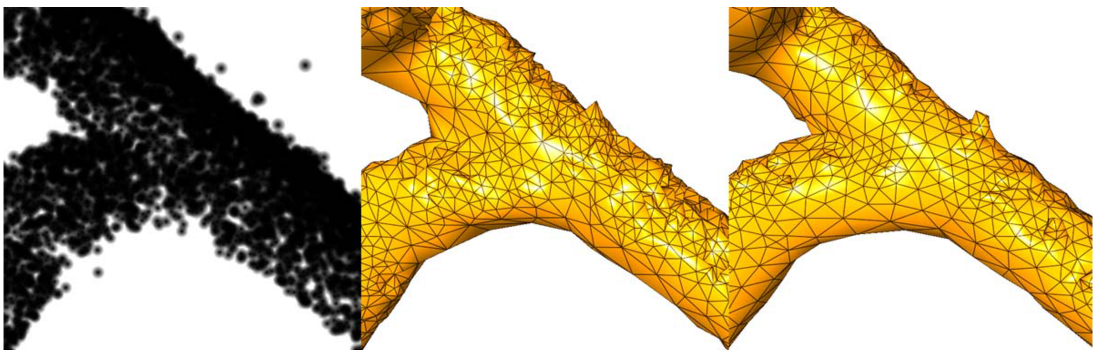


Figure 6.9: Very noisy section of the Heating Pipes model (left); Pointy vertices or spikes on the surface of the GCS algorithm (middle) and a smoother surface with the GSS approach (right).

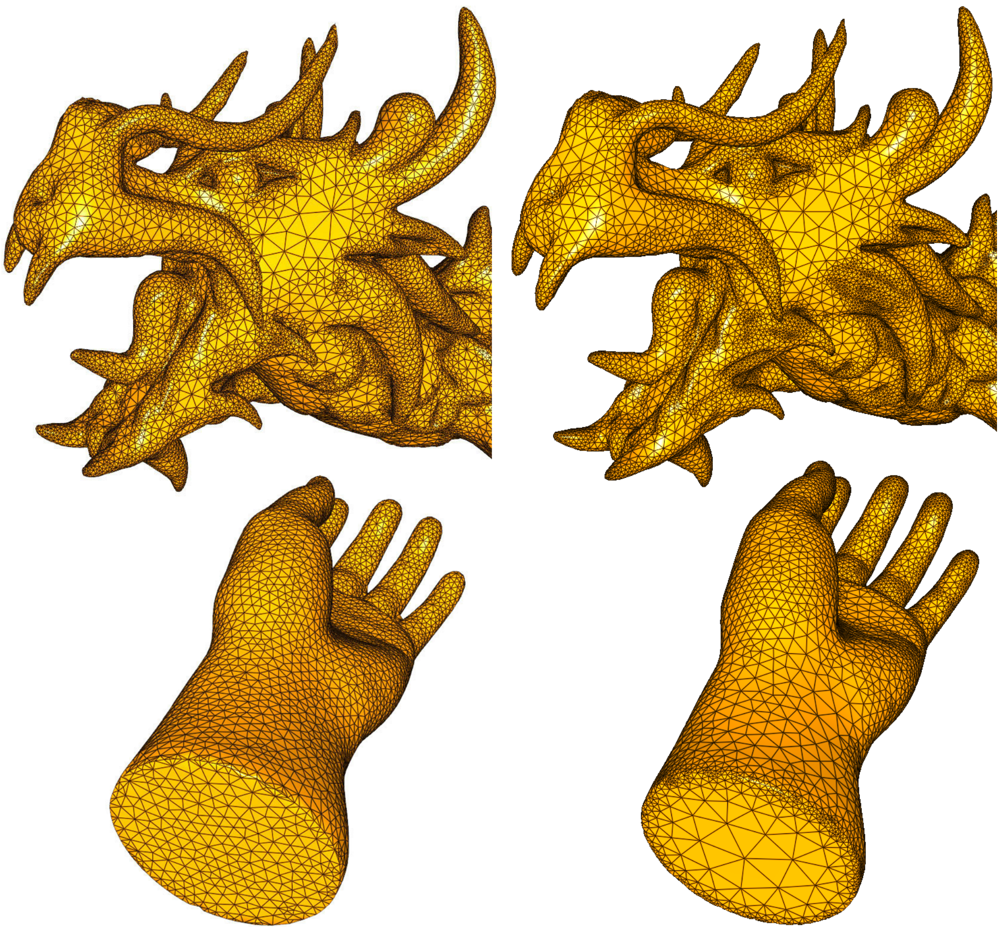


Figure 6.10: Reconstructions with SGC (left) and GSS (right). The Asian Dragon exposes curvature (top), while the Hand has a sharp feature (bottom). Note that the more direct connection between curvature and subdivisions of the GSS process leads to more detail at the jaw of the Dragon, while using the same number of triangles. The feature line of the Hand is more pronounced with the SGC algorithm, while it also exposes some low quality triangles.

this led to many clump-like artifacts. Due to this results using the square distance error can be generally considered impractical.

6.5 Conclusion

The presented GSS approach was derived from the analysis of different approximation settings for the GCS algorithm. Basically three goals were presented: creating a likelihood distribution with the vertices, *i.e.*, their placement resembles a generalization of the samples; creating a distance minimization with the vertices, where the approximated subspace ($\mathscr{S}_{phy}$ for surface reconstruction) is evenly sampled by the vertices; and creating a topology as close as possible to the one of the approximated subspace.
While the first two goals focus on measurements toward vertices, the latter novel one focuses on the actual created surface and its topology. The basic idea of the presented approach is to incorporate the constructed topology into the GCS learning scheme and to use sample-to-surface relations to provide more sensible information to recognize $\mathscr{S}_{phy}$. A vital part of this idea is to use sample-to-surface distances. The minimization of these distances is considered to lead to a faster recognition of a correct topology. Experimental results confirm this notion.

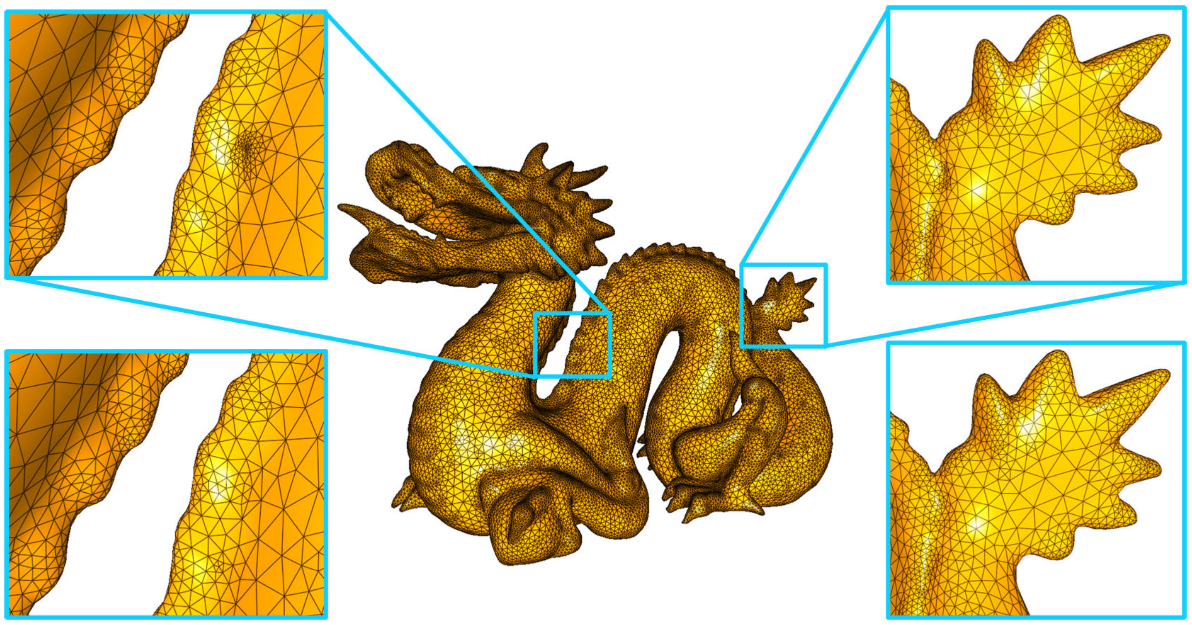


Figure 6.11: Two magnifications of the Dragon model reconstructed with100K triangles with the squared sample-to-surface distance as approximation error (top) and with just the distance error (bottom). In the former case, the triangle resolutions at the back and between tail and body are very high and overrepresent these areas.

When looking upon surface reconstruction as a problem of statistics, it is part of the *regression analysis*. However, most common problems in that area focus on optimizing *sample-to-subspace* distances only. For instance, when placing a line into a set of samples in a 2D space, as in *linear regression*. But this solely resembles the (surface) *fitting* problem. Here, the topology of the subspace is already determined, making the surface a *function*. In surface reconstruction with a refinement strategy, however, this function is determined while being optimized.

Surface fitting and topology optimization are two clearly separated problems when judged upon the surface properties to be satisfied. However, when setting 3D samples in relation to a current surface estimate to determine information about the investigated surface in a refinement approach, these problems merge.

The differentiation of approximation errors in the form of sample-to-surface distances and topological misconstructions is the single most important ability and also the most important task when implementing a GSS based reconstruction algorithm. The separate deletion processes for geometric redundancy and for topological misconstructions are examples of considering this dichotomy.

Although the presented GSS implementation works well, the concept by itself does not solve the general differentiation problem, but rather offers the necessary information to enable a differentiated handling. Given that the topology of a surface is correct, using sample-to-surface distances to determine locations for subdivisions is absolutely plausible. However, when introducing topological misconstructions, this methodology becomes problematic.

A large distance of a sample to the surface can mean the surface needs to be subdivided to get closer to the sample or that there is an entirely missing surface region and topological changes are needed. Since entirely new surface regions still have to be drawn out of existing surfaces, there is a transition between surface fitting and the actual creation of new topological structures (see Fig. 6.12). In this case, due to huge distances, huge amounts of subdivisions create surface, which is then adapted to represent the missing surface region. Although the correct surface is created eventually, the error values temporarily do not actually resemble sample-to-surface distances. These distances result from a misrepresented subspace, which in this case falsely sets a surface in relation to a sample, while the surface related to that sample does not yet exist.

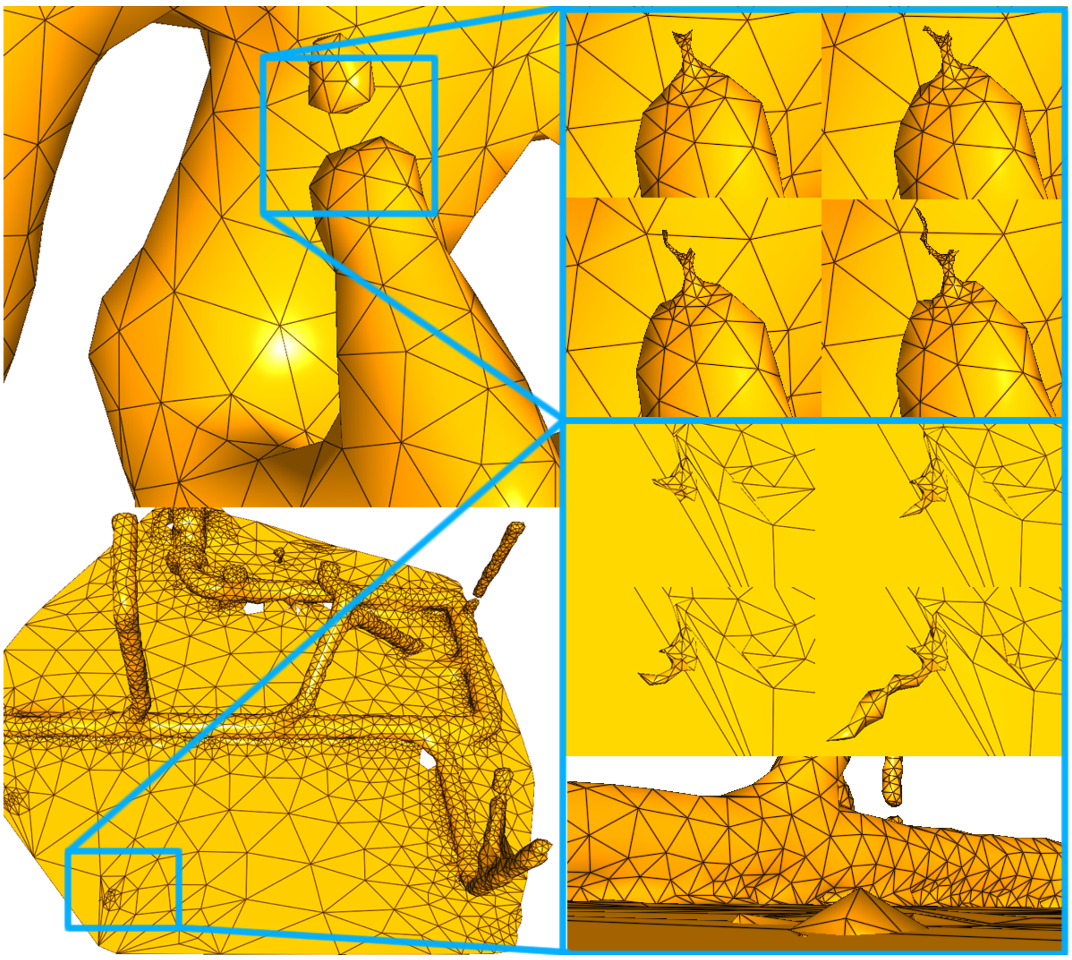


Figure 6.12: Creation of an entirely missing surface area at the leg of the Dragon model (top). Creation of a clump-like artifact, due to a falsely recognized surface topology at the noisy floor of the Heating Pipes model (bottom). Note that these pictures have been created purposefully by using an out of proportion crumb filter for the Dragon model and by setting error values equal to single sample-to-surface distances ($err_{new} = d_x$).

When alternating between one single and multiple surfaces, the problem becomes even worse. If the topology of a flat surface has been correctly recognized, sample-to-surface distances are very low, correctly leading to very few triangles. However, as soon as topologically incorrect structure is temporarily built in such an area, the sample-to-surface distances based on the incorrect subspace are huge. So, such a structure is very likely to grow through many subdivisions and build clump-like artifacts (see Fig. 6.12). The initial creation of such incorrect temporary structures is often caused by noise in the samples.
It is important to notice that the sample-to-surface distances do not cause this problem, but it only become clearly apparent due to this advanced information available in the GSS concept.
With the GSS approach the process could theoretically abandon all operations to optimize triangle shapes, such as the Laplacian smoothing process and the outbalancing of valences, and only modify the surface to minimize sample-to-surface distances. However, this is very likely to destabilize the process, since its robustness against noise and other distortions is mainly based on evenly distributed vertices, which result from triangles tending to be equaliteral.
The shown results for the presented implementation of the novel learning scheme are a proof of concept and give rise to many improvements for the algorithm in future work.

Chapter 7

Results

This thesis presented a detailed analysis of unorganized point based surface reconstruction. A broad overview of all major surface reconstruction methods and their performances was given. The origin of problems, their solutions as well as their dependencies were discussed. This enabled a more detailed perspective on the capabilities of the GCS algorithm and on its newly added features.
With the presented approaches or only marginally changed versions of them an extensive mesh and point cloud processing *tool-kit* can easily be created. It includes filters for point clouds for noise and non-uniform sample density removal, or suchlike to clean and edit meshes. It also includes algorithms for mesh unwrapping, LOD, remeshing, mesh optimization, and normal estimation tasks.
The search for an efficient and integrable solution on twisted surface regions led to a semi-local processing strategy. This strategy and the data structures used alongside it have great potential. Processed surfaces are independent of the mesh size. Assuming the front line to be an actual 1D contour, the memory consumption is proportional to the square root of the processed surface area. The runtime complexity is $O(n \log n)$, where n is the number of processed surface structures. Using a directed sealed edge-front exploits the properties of a 2D surface to gain efficiency and additionally allows for detecting and localizing front collisions.
The MDF was not used in the final reconstruction algorithm, but served as a proof of the potential lying in the edge-front based semi-local processing strategy. Making on-surface distances accessible for mesh analyzing, searching and editing is a valuable asset for computer graphics applications in general. The use of on-surface distances makes the MDF more independent of the actual underlying mesh structure.
The comprehensive overview of the GCS approach not only included additional uses of the algorithm, but also the successive buildup of the GCS algorithm with a focus on surface reconstruction. This systematic algorithm buildup allowed for localizing every algorithm property, optimization and problem solution, even as heritage from algorithm stages prior to the GCS approach.
GCS based reconstruction algorithms inherit the stochastic sample access from the k-means clustering approach, allowing virtually the processing of an infinite number of input samples. Placing vertices in the means of samples leads to robustness against noise and outliers. The SOM approach – introducing topology – enables surface reconstruction applications. As the surface is dragged over the samples, the process is robust against non-uniform sample densities and holes. As a reconstruction algorithm, the most important difference of the actual GCS approach to its algorithmic predecessors is its strategy to derive implications about the unknown surface $\mathscr{S}_{phy}$ from $\mathscr{P}$. The GCS algorithm starts with a simple mesh and uses an iterative refinement strategy. Every current estimate $\mathscr{S}$ always represents a result for the approximation of $\mathscr{S}_{phy}$.

This makes the GCS approach very powerful in avoiding local failures caused by ambiguous point constellations. It also adds flexibility, since it enables stopping and resuming the process at any time. With the tumble-tree and the k-d tree the algorithm has an average case runtime complexity of $O(n \log n)$.

Comparing GCS with GNG proved the former to be the more fitting for surface reconstruction. Both share many advantages, such as an independence of the point cloud size and a resistance against noise and non-uniform sample densities. The simpler GNG graph has fewer constraints when changing its connectivity, making it more suitable for parallelization. The GCS concept on the other hand needs fewer points to sustain a surface once constructed. More important, its surface model actually represents a triangular mesh and is therefore more sophisticated. If samples are set in relation to such a surface model, more meaningful information can be derived. For example, the Laplacian smoothing operation, the inertia of the surface evolving process, the presented filter processes, the twist solving process, and the new topology focused approximation concept are all based on an existing surface model.

An enhancement in the differentiation and variety of rules, which vertices follow during their adaptation process, were presented in the SGC approach. Here, the local behavior of vertices can be precisely modeled to deal with specific mesh aspects such as curvature, sharp features and boundaries. The cutting and coalescing mechanisms establish independence of the initial mesh topology and solve the problem of surfaces getting stuck in local minima. However, many surface progression related problems the SGC concept deals with were shown to be symptoms of a more general problem, as discussed in the chapter on the GSS concept.

The filter chain was introduced to enable easy editability for the implementation of individually desired behaviors, which then account for different reconstruction scenarios. Filter chains achieved this by allowing easy adding and removal of filters for different chains. Besides the cleaning of the mesh after cutting operations, this also allows artifacts such as huge triangles caused by outliers or undesired small crumb alike mesh segments, to be filtered out.

The presented valence optimization filter proved to be a valuable contribution to the mesh quality in general. The filter chains make the algorithm behavior very flexible and easy to edit, as experimentally demonstrated. However, the possibility of mutually dependent filters causing infinite loops is an important drawback of the concept.

To fix twisted surface regions in a current surface estimate included finding a solution to a *global* problem, which had to be integrated into a process only using *local* refinement operations. The solution to this complex problem lay in a semi-local processing strategy using the MEF, which enables a compact solution efficient in runtime and memory consumption. The presented twist resolving mechanism proved to be reliable and efficient on a variety of tested models. It had practically no significant impact on the runtime of the algorithm, since it was only seldomly triggered and no falsely detected twists were noted.

The surface creation in the GCS algorithm is based on iterative refinement operations, which constantly improve a current surface estimate. Within these operations, however, the current surface estimate is only represented implicitly. A direct representation of the created topology within the process allows for creating precisely modeled algorithm operations involving actual sample-to-surface distances. This shifts the focus on topology rather than the distribution of vertices. This novel GSS concept represents a major conceptual algorithm change, which enables redirecting of the GCS approximation behavior.

This concept was implementated and tested on a variety of challenging models. Tests showed that the new approximation behavior is faster in topology adaptation, superior in handling thin structures, produces fewer twists, handles sharp features, and creates smoother surfaces in the presence of noise. Although the closest surface structure search process is more complex, the extra time costs are compensated by discarded operations. However, using sample-to-surface

distances as the only criterion for subdivision can cause artifacts, due to the dichotomy of the surface fitting and topology finding problem.
In the following considerations the filter chain concept, the twist resolving mechanism, and GSS algorithm are all incorporated in one final algorithm for evaluation purposes. Besides the GSS algorithm, all algorithm components are used as presented in their corresponding chapter (see Alg. 13).
For the final algorithm the GSS concept is changed in two aspects. The sample-to-surface distance is not added as an approximation error, but is divided by the average sample-to-surface distance and then added. This makes it a hybrid between signal counter and distance error (see line 16 in Alg. 13). Still, huge distances create bigger signal counts, thus leading to more subdivisions. These subdivisions are, however, more evenly distributed avoiding artifacts. As the distribution of the vertices becomes more uniform, the distance error counters can be re-located from the triangles to the vertices, making the process more efficient both in runtime and memory consumption. Also, an age is assigned to the vertices, which resembles the one presented in the GNG algorithm. Most benefits of the concept are preserved by these changes, such as the search process being superior in topological correctness, the time efficient of the process due to skipping samples, the smoother surface through the movement of whole triangles and edges, and most importantly the differentiated deletion process.

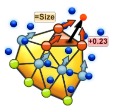
The algorithm is evaluated by reconstructing surfaces from a series of reference data sets that represent a variety of problem cases. It then is compared to some well-known reconstruction algorithms. The described algorithm is referred to as the “final GSS algorithm”.

7.1 Comparison with Classical Reconstruction Approaches

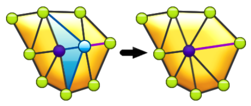
For the comparison four models are selected. The Hand (38K samples) being a relatively easy model to reconstruct, but which exposes a sharp feature, the Stanford Dragon (438K samples) with its challenging shape, the Happy Buddha (544K samples) model with thin surface areas and a challenging topology and the Heating Pipes (918K samples) which expose noise, outliers and open surfaces.
Four algorithms are tested including the final GSS algorithm. All used implementations originate from the open source mesh processing tool *MeshLab* [IST13]. The region growing algorithm *ball pivoting* [BMR+99] from section 2.4.1.2 is an example for a surface oriented interpolation method. Volume oriented interpolation methods such as *α-Shapes* [EM92] and the *crust* [ABK98] are not included in the comparisons, since they did not produce reasonable results concerning the underlying test geometries. Using those approaches, surfaces were represented by strongly interlaced triangles exposing neither a sensible orientation nor a continuous 2D triangle surface. As an example for a distance function based approximation method *robust implicit moving least squares* (RIMLS) [OGG09] (see section 2.4.3.2) is used, which represents a MLS approach. As a volume oriented distance function, instead of the surface oriented MLS functions, the *Poisson surface reconstruction* is used [KBH06] (see section 2.4.3.4).
The comparison is on the one hand designed to resemble a realistic reconstruction scenario and on the other hand to take the different requirements of the algorithms into consideration. In order for the interpolation methods to perform properly, evenly distributed samples are required. Therefore, all point clouds are down-sampled with a clustering algorithm which also creates evenly distributed samples as described in section 2.2.2. The Heating Pipes point cloud is down-sampled to 150K points, but only for the interpolation method.
The distance function based approaches rely on samples augmented with surface normals. These normals are estimated with a standard approach described in [HDD+92]. However, in

Algorithm 13 Final GSS Algorithm
(combines the filter chain Alg. 7, twist solving Alg. 11, and GSS Alg. 12)

```
1:  Initialization
2:  repeat
3:      repeat
4:          repeat
                Basic Step
```



```
5:              Select random sample $\mathbf{p}_x$ of $\mathscr{P}$
6:              Find the vertex $\mathbf{v}_x$ in $[\mathbf{s}]$ that exposes the smallest on-surface distance $d_x$ to $\mathbf{p}_x$
7:              Set the age $a_x$ of $\mathbf{v}_x$ to zero: $a = 0$
8:              Increase the age of all other vertices: $\forall a_i (i \in \mathscr{M})\ \Delta a_i = 1$
9:              if $d_x$ in relation to $\overline{d_{\mathscr{P}}}$ indicates an adaption is needed: $d_x / \overline{d_{\mathscr{P}}} > lim_{skip}$ then
10:                 if Only a slight adaption is needed: $d_x / \overline{d_{\mathscr{P}}} > lim_{single}$ then
11:                     Set $[\mathbf{s}]$ to be the closest vertex in $[\mathbf{s}]$ to $\mathbf{p}_x$
12:                 end if
13:                 Move all vertices in $[\mathbf{s}]$ as much toward $\mathbf{p}_x$ as determined by the learning rate $lr$
14:                 Perform Laplacian smoothing on all neighbors $\mathscr{N}_{[\mathbf{s}]}$ of $[\mathbf{s}]$ as much as determined by the
                    neighbor learning rate $lrn$
15:             end if
16:             Increase signal counter $sc_x$ of vertex $\mathbf{v}_x$ by one: $\Delta sc_x = d_x / \overline{d_{\mathscr{P}}}$
17:             Decrease signal counters of all other vertices by a fraction: $\forall sc_i (i \in \mathscr{M})\ \Delta sc_i = -\beta \cdot sc_i$
18:             if $\mathbf{v}_x$ is boundary vertex AND coalescing partner $\mathbf{v}_{opp}$ was found then
19:                 Coalescing
20:             else
21:                 Solve Twist
22:             end if
23:             Filter Chain
24:             Increment the iteration counter: $\Delta t = 1$
25:         until The basic step has been performed $c_{add}$ times: $t$ mod $c_{add} = 0$
26:         Vertex Split
27:     until The basic step has been performed $c_{del}$ times: $t$ mod $c_{del} = 0$
        Remove
```



```
28:     Select vertex $\mathbf{v}_x$ with the lowest signal counter value $\forall sc_i (i \in \mathscr{M}) >= sc_x$
29:     if $\mathbf{v}_x$ exposes an edge $\mathbf{e}_{xy}$, which is geometrically redundant: $\mathbf{n}_x \cdot \mathbf{n}_y < max_{\nabla n}$ then
30:         Perform an edge collapse operation on $\mathbf{e}_{xy}$
31:     end if
32:     Select vertex $\mathbf{v}_y$ with the highest age $a_y$: $\forall a_i (i \in \mathscr{M})\ a_i <= a_y$
33:     if The age $a_y$ of $\mathbf{v}_y$ is too high: $a_y > max_a$ then
34:         Cut $\mathbf{v}_y$ out including its surrounding faces and edges
35:     end if
36:     Filter Chain
37: until A certain number of vertices is reached: $|\mathscr{M}| >= n_{final}$
```

Point Cloud (# samples)	time	# triangles	dist	dist^2	equilaterality	valence$[5;6;7]$
	Ball Pivoting					
Hand (19K)	0:0:4	37K	1.534	1486	59.0%	87.6%
Dragon (150K)	0:2:36	239K	2.596	1.750	53.3%	63.3%
Buddha (157K)	0:2:50	295K	2.027	0.948	56.3%	77.2%
Pipes (150K)	0:2:42	277K	3.107	2.001	53.3%	54.5%
	RIMLS					
Hand (19K)	0:0:3	48K	1.235	116.1	47.8%	92.2%
Dragon (150K)	0:0:25	312K	1.098	0.300	47.1%	89.2%
Buddha (157K)	0:0:35	447K	1.048	0.169	47.9%	91.5%
Pipes (918K)	0:5:31	1,322K	1.006	0.134	48.1%	70.1%
	Poisson Reconstruction					
Hand (19K)	0:0:2	31K	15.68	13132	50.5%	73.6%
Dragon (150K)	0:0:18	257K	6.787	6.135	50.1%	72.5%
Buddha (157K)	0:0:21	353K	6.993	2.586	49.9%	73.5%
Pipes (918K)	0:1:02	1,166K	45.23	39189	51.9%	63.8%
	Final GSS					
Hand (19K)	0:0:6	35K	1.818	151.7	82.1%	98.7%
Dragon (150K)	0:2:21	290K	1.496	0.373	81.1%	97.9%
Buddha (157K)	0:2:31	300K	1.171	0.071	81.4%	98.2%
Pipes (918K)	0:0:18	50K	6.994	38.84	80.4%	98.0%
Farm (10M)	5:09:02	14M	0.719	0.113	86.6%	98.1%
Bridge (26M)	6:58:10	18M	0.296	3.534	85.1%	97.0%
Office (451M)	6:03:21	22M	0.636	0.371	84.1%	98.2%

Table 7.1: Performance comparison of different reconstruction algorithms.

case of the Happy Buddha model no reasonable normal estimation could be achieved. To avoid the intentional creation of bad reconstructions results for this point cloud, the original normals from the ground truth model are used. The Heating Pipes point cloud exposes the same problem. The normals show contradicting orientations for pipes and floor. Since using the normal estimation built upon the twist solving GCS algorithm (see section 6.3) would distort an algorithm comparison, the known to be incorrect normals are used for the reconstruction. For comparison purposes, the algorithms are set to produce roughly the same number of vertices for the resulting models. The Heating Pipes model is an exception, since its high noise level detaches its sample number from the number of vertices actually required.
All *MeshLab* algorithms are performed multiple times with different parameter settings especially if results appear insufficient. For the comparison the best results are chosen.
The results of the comparisons are presented in Table 7.1 and Fig. 7.1, Fig. 7.2, Fig. 7.3, and Fig. 7.4. The distance measurements are performed on the full resolution point clouds and not the down-sampled ones used for the reconstruction.
When comparing the speed, the final GSS algorithm is rather slow. Only the ball pivoting algorithm is slower. This is actually very surprising when considering its working principle (see section 2.4.1.2). With an optimized implementation, it should be one of the fastest algorithms.
At sample-to-surface distances the RIMLS algorithm and the final GSS algorithm reach the best results especially when comparing the square distances. This happens due to a mechanism

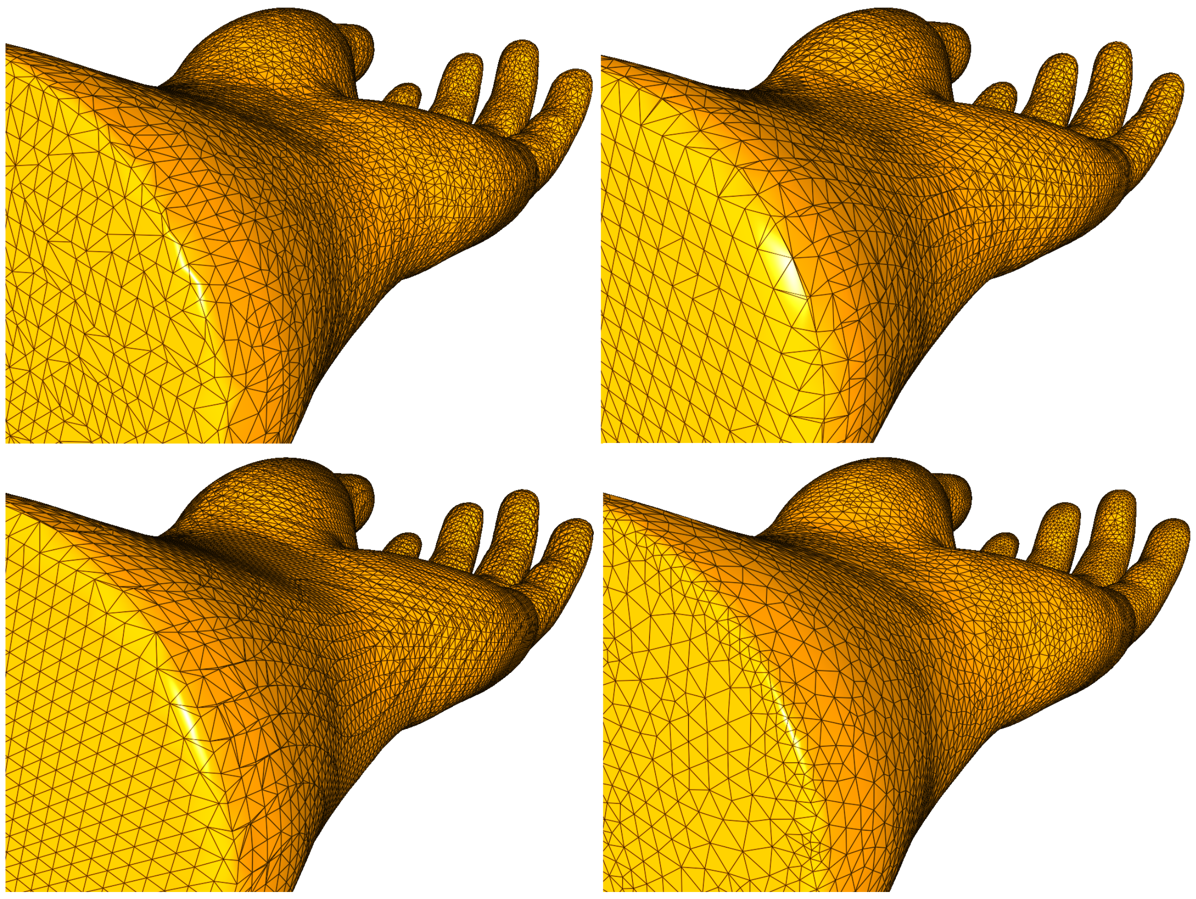


Figure 7.1: Reconstructions of Hand model: Ball pivoting (top left), RIMLS (top right), Poisson reconstruction (bottom left), and the final GSS algorithm (bottom right).

to specially preserve curved areas in the RIMLS algorithm and the topology led subdivision process in the final GSS algorithm. When examining the produced triangle quality, the final GSS algorithm produces by far the best shaped triangles. It can also be observed that the closeness of triangles to being equilateral is a more sensible measurement for triangle quality than the valence distribution, since the former values have proven to be steadier for the different approaches.
The sharp feature at the Hand model is decently reconstructed by all algorithms, however, the marching cubes based triangulations form the most differentiated feature line. The Poisson surface reconstruction constantly produced closed meshes, even if an open surface was intended for the Heating Pipes model. The final GSS algorithm also produces sealed models although providing no theoretical guaranty for this. Additionally, it reconstructs the open surface correctly.
The final GSS algorithm and Poisson reconstruction are the best performing approaches in the presence of noise when ignoring the effects caused by the incorrect normals. While the Poisson reconstruction always has a smoothing effect on the resulting surface, the final GSS algorithm can still match the accuracy of the MLS based approach. Besides its runtime the final GSS algorithm is always close or at the top of the ranking when investigating different reconstruction problems. This proves the algorithm as an efficient all-round high quality approach.

7.2 Reconstruction of Challenging Models

In the comparison to other algorithms, the final GSS algorithm proved its strong all-round abilities. These abilities to robustly deal with very different and demanding problems at the same time is tested in the following with three objects scanned by terrestrial laser scanners.

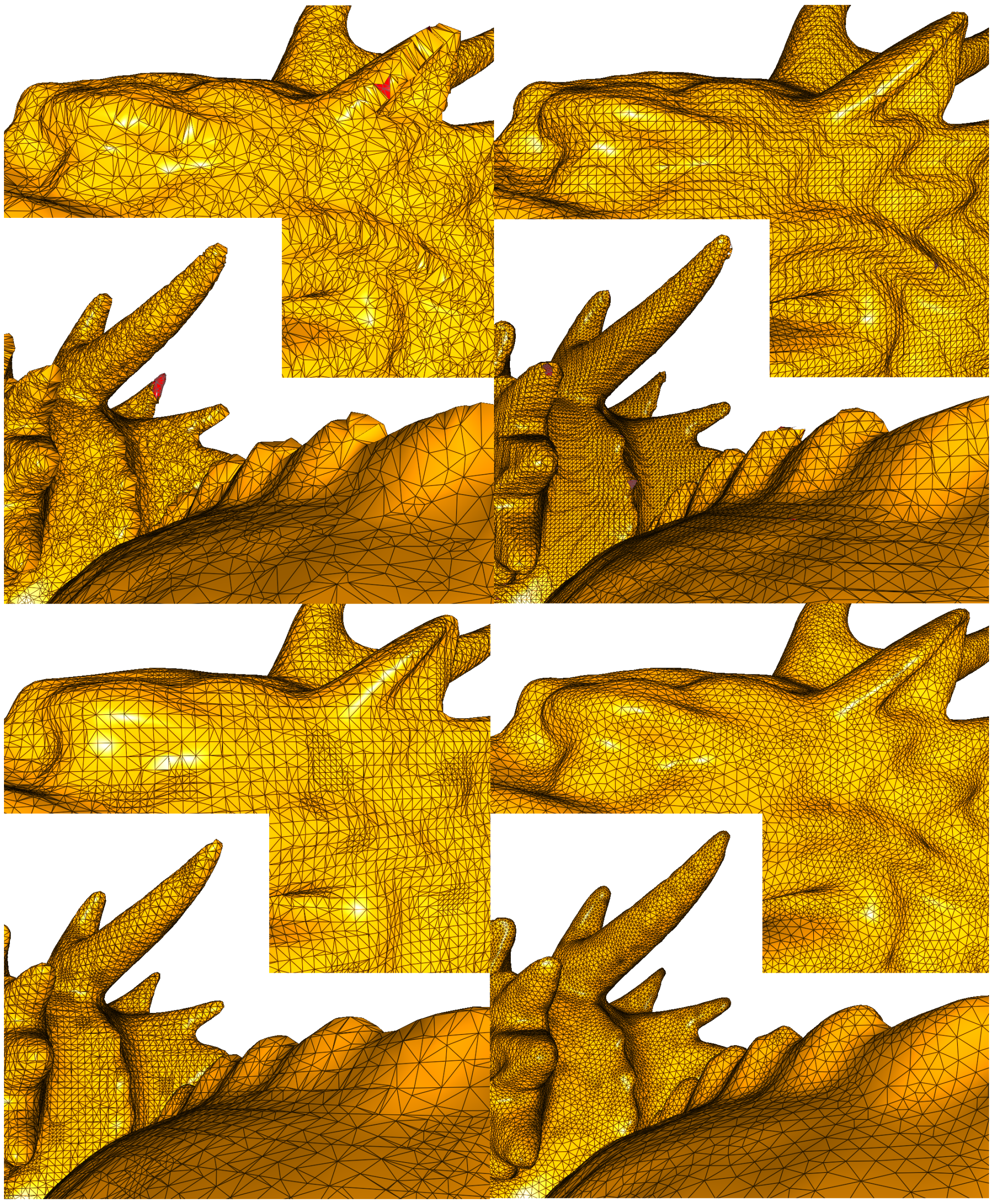


Figure 7.2: Reconstructions of Dragon model: Ball pivoting (top left), RIMLS (top right), Poisson reconstruction (bottom left), and the final GSS algorithm (bottom right).

These objects expose most challenges addressed in the sections on the analysis of the surface reconstruction problem.
The first data set is the “Farm Building in Ethiopia” (10M samples). It shows an open landscape with vegetation, noise, outliers, non-uniform sample densities, large holes and a lot of ghost geometry. The “Kornhaus Bridge” (26M samples) exposes thin metal structures, noise, outliers and strongly non-uniform sample densities. The final scan comes from the laser scanning department for crime scene investigation in Hamburg (Germany) and shows their “Office” (451M samples). This data set has many outliers, sharp features and many thin and tiny structures. In Table 7.1, the results of the reconstructions are presented and they are shown in Fig. 7.5, Fig. 7.6, and Fig. 7.7.
Although the final GSS algorithm was rather slow in the presented comparison, it has a low average case runtime complexity. When reconstructing these huge point clouds, the process takes several hours. This is still in the range of a “one night” process, which is convenient when considering the high quality reconstruction results. With the filter chain the Farm Building in

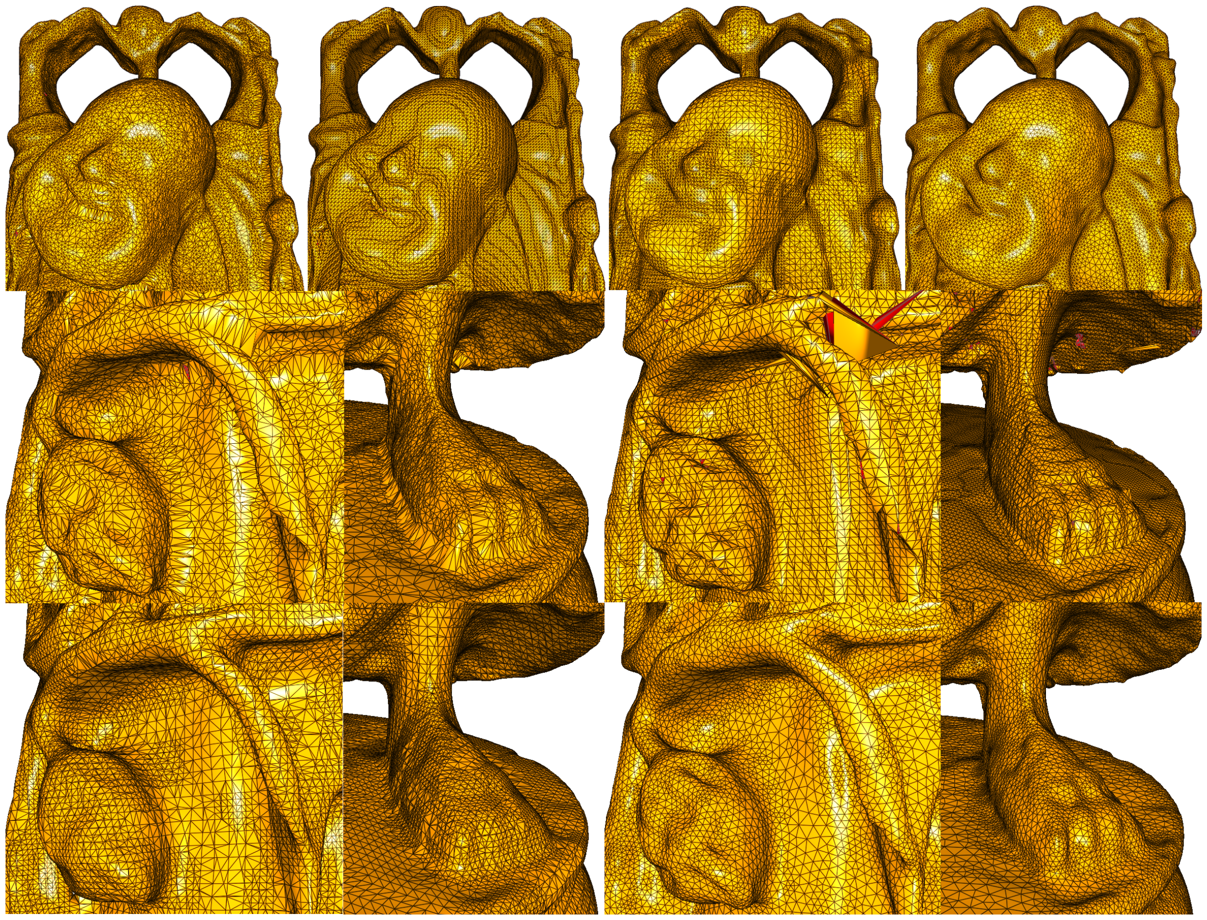


Figure 7.3: Reconstructions of the Happy Buddha model: Ball pivoting, RIMLS, Poisson reconstruction, and the final GSS algorithm (top row). Ball pivoting (middle left), RIMLS (middle right), Poisson (bottom left), and the final GSS algorithm (bottom right).

Ethiopia could be completely freed of its ghost geometries and other disturbances within the point cloud. Even the thin metal structures of the Kornhaus Bridge are correctly reconstructed despite of its challenging sample distribution. The office environment exposes the algorithm to a variety of problems at the same time. Here, the general robustness of the approach becomes most apparent. All model surfaces are accurate and expose high triangle quality and a consistent global surface orientation. Although, the Kornhaus Bridge is reconstructed with fewer triangles, its reconstruction takes more time than the one of the Office. The reason for the difference lies in the sample distribution of the Office, which exposes many flat surface regions, that are very suitable to be organized in the presented k-d tree.

If flaws occur during the reconstruction process, they only lead to minor artifacts in the resulting mesh. Some imperfections are bars at the bridge railing being falsely coalesced and some twisted metal structures staying undetected, since they are yet too low in resolution for the twist resolving process. The Office exposes some surface distortions due to noise at the jackets. Solutions for these cases are discussed in section 8.3.

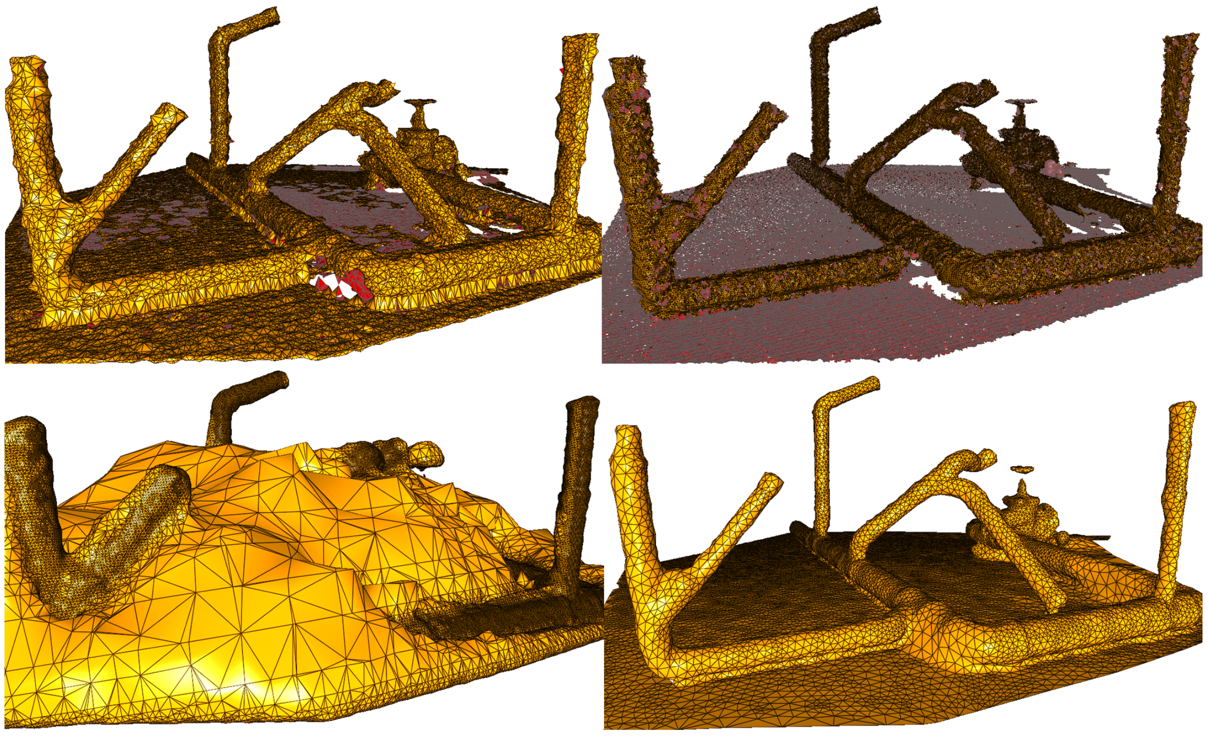


Figure 7.4: Reconstructions of Heating Pipes model: Ball pivoting (top left), RIMLS (top right), Poisson reconstruction (bottom left), and the final GSS algorithm (bottom right).

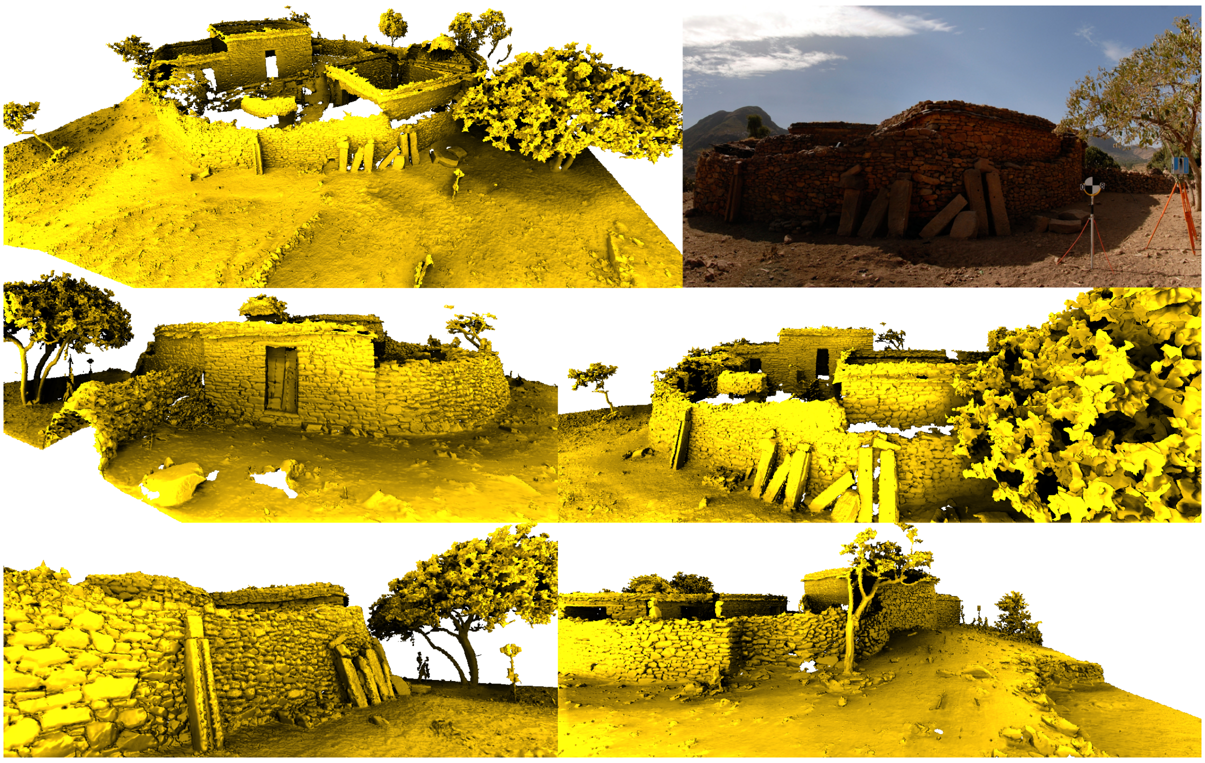


Figure 7.5: Image of the Farm Building in Ethiopia and its reconstruction with the final GSS algorithm.

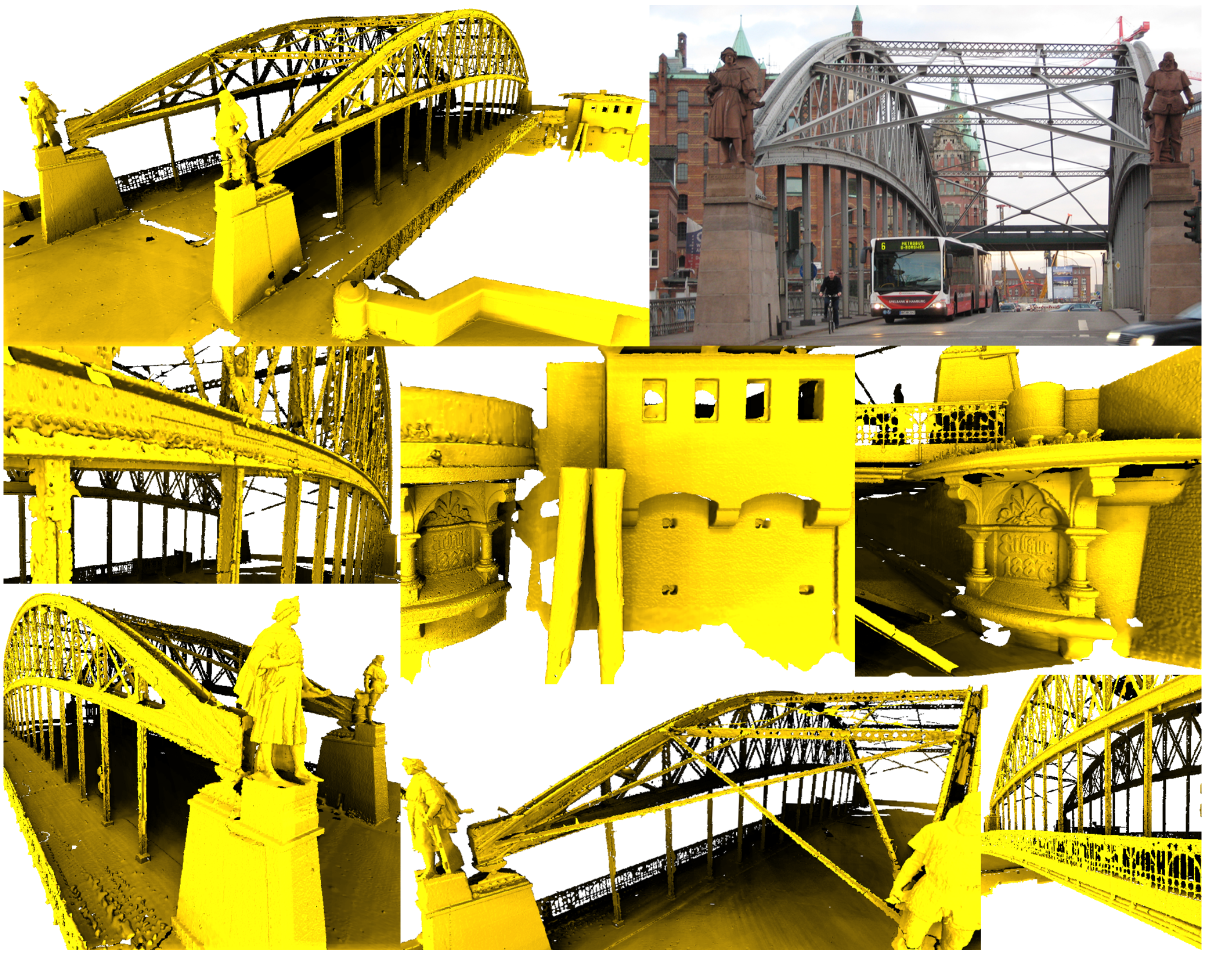


Figure 7.6: Image of the Kornhaus Bridge and its reconstruction with the final GSS algorithm.

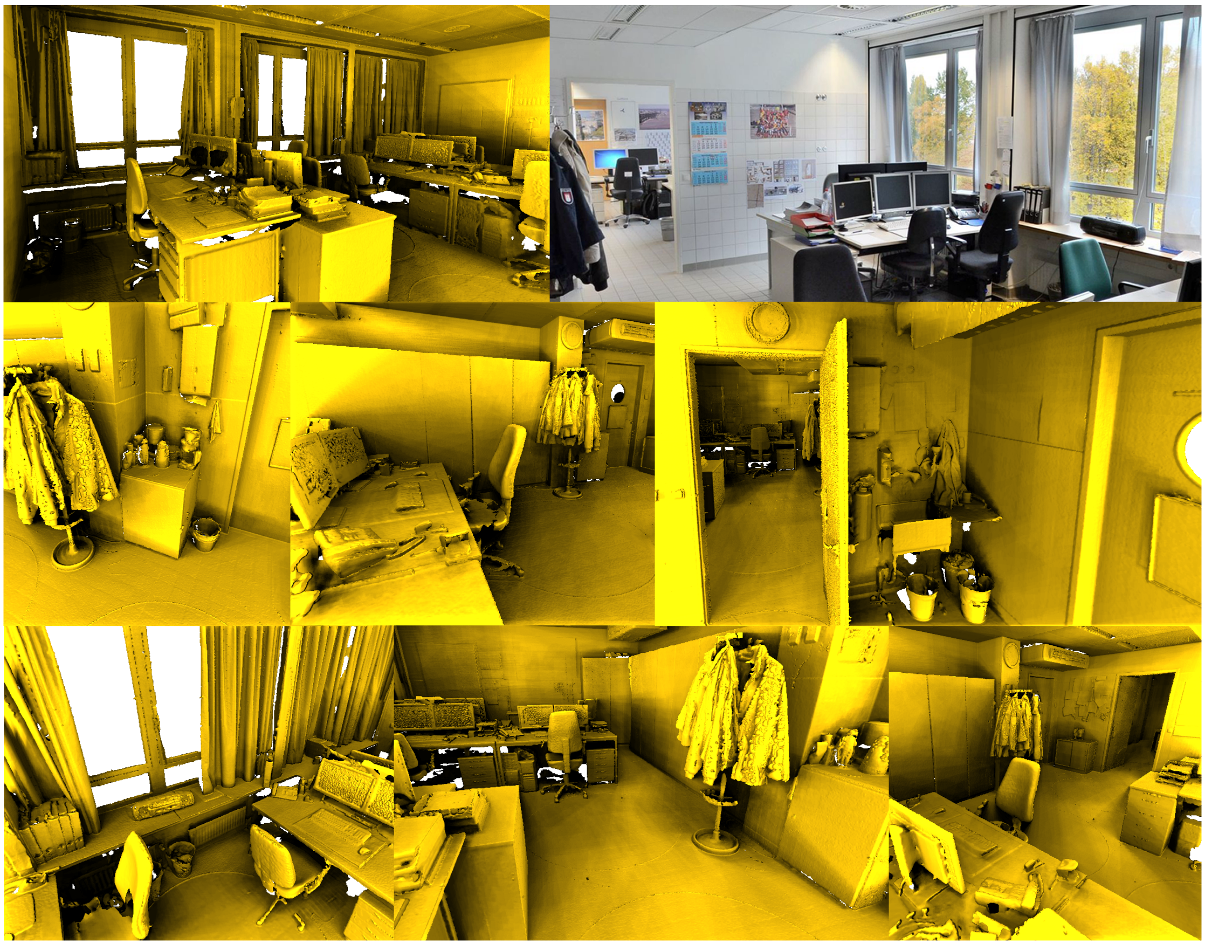


Figure 7.7: Image of the crime scene investigation Office and its reconstruction with the final GSS algorithm.

Chapter 8

Conclusion

In this chapter, the findings of this thesis are summarized and the core contributions are discussed. This is followed by considerations on the challenges future work might hold and a finalizing epilog.

8.1 Summary

This thesis presented a detailed overview of the subject of surface reconstruction. The GCS algorithm was presented with all its inherited, inherent and added properties concerning the surface reconstruction task. Furthermore, its additional uses were presented and it was differentiated from the GNG algorithm.
Novel algorithm developments and concepts were introduced to create a more efficient, robust, reliable and high quality surface reconstruction algorithm. It was given more flexibility with the filter chain concept, enabled to deal with twisted surface regions and conceptually improved to account for topology focused approximation behavior.
The resulting algorithm was compared with classical reconstruction algorithms and tested by reconstructing surfaces from extremely challenging point clouds. The presented algorithm proved to be an efficient all-round high quality approach.

8.2 Discussion

Growing Cell Structures: GCS as an algorithm concept is a very versatile computer graphics tool, as proven by the many presented point cloud filters and mesh-processing algorithms based on it.
A comprehensive overview of the GCS algorithm was given including algorithm stages prior to GCS. This detailed presentation allowed for perceiving GCS based reconstruction algorithms, not as one indistinguishable algorithm, but as a systematically composed collection of distinct and localized properties. This overview enabled precise discussion and the introduction of conceptual changes, such as the GSS.
Growing Cell Structures vs. Growing Neural Gas: The most important differences between GCS and GNG lie in their mechanisms to build and incorporate topology in their processing scheme. Whereas GCS produce an explicit 2D surface solely constructed of triangles, GNG produces an arbitrary graph. For surface reconstruction this is not only interesting at the end of the process, where the GNG graph has to be transformed into a valid 2D surface, but also within the main processing phase. GCS can put samples in relation to an actual surface estimate and

thereby deduce additional information. The pinnacle of the use of this strategic advantage lies in the presented GSS concept.

Filter Chain Concept: The filter chain concept enables great editability of the algorithm behavior, since new behavior can easily be added and removed. The capabilities of the concept were experimentally demonstrated by modifying the algorithm toward different problem cases. The problem of mutually dependent filters causing infinite loops is a conceptual weakness that needs consideration when implementing and combining filters. With a composition of filters complex algorithm behavior can be implemented systematically and is easy to edit.

When representing the entire algorithm functionality as filters, the local rule based SGC concept could be elevated to a new level, where rules could be automatically exchanged and tested. Possibly, rule sets could even automatically adjust to the given input data.

Twist Solving: Finding a suitable solution to resolve twisted surface regions, which properly integrates into the iterative refinement process of the GCS approach, was theoretically as well as in its implementation a complex task. The presented *semi-local* processing strategy is able to cope with the *global* characteristics of falsely oriented surface regions and integrates into the otherwise *local* optimization process.

With the presented twist solving mechanism one of the last vital limitation for using the GCS algorithm as a universal surface reconstruction approach was removed. The presented mechanism is a very general solution to the problem, since its surface separating process is based solely on mesh connectivity aspects and thereby does not rely on geometry based heuristics. The process was experimentally proven to be reliable and efficient, nevertheless it might fail in cases where inconsistent orientations are not recognized before the process finishes.

Edge-Front: The presented solution is very compact, due to the use of a highly efficient and universal data structure – the edge-front. Edge-fronts hold great potential for computer graphics applications, which was proven by the results of MDF in geodesic distance calculations. Many other extensions and improvements of edge-front based computations, not needed for the twist solving process, are left for further investigation, such as *static* front line elements, *curvature* driven expansion processes, and front elements which *remain* on collision.

The edge-front data structure itself represents an important contribution to this thesis.

Growing Surface Structures: The presented GSS concept introduces a topology focused approximation behavior to the GCS approach. Instead of focusing on the vertex distribution, this approach aims to create a topology as close as possible to one of the approximated subspaces. The presented approach incorporates the constructed topology into the GCS learning scheme. Sample-to-surface distances and closest surface structures become available within the algorithm operations. The novel GSS concept enables a better differentiation between approximation errors in form of sample-to-surface distances and topological misconstructions. This ability is vital for creating topologically correct reconstructions. The presented implementation of the novel learning scheme provided convincing practical results on test data sets. However, many alternative implementations, which might use the additional information in different ways, are left for further investigation. While the common GCS learning scheme focuses on creating vertex distributions, a possible implementation of the GSS learning scheme could focus on creating a “surface distribution”. Such a distribution would solely aim to minimize sample-to-surface distances and would discard all operations with the sole task to increase triangle quality. Although this increases the risk of artifact creation it would also open up a novel interesting algorithm behavior for further investigations.

Final Combined Reconstruction Algorithm: In this thesis, a universal surface reconstruction algorithm was presented. It processes unorganized points as input data, which may be distorted by noise, outliers, non-uniform sample densities and which may expose holes in the sampling. Compared to classical reconstruction approaches, it can virtually cope with any number of input samples and has a low average case runtime complexity of $O(n \log n)$. The algorithm constantly optimizes a surface estimate of a currently investigated surface. This refinement strategy makes it very robust when dealing with ambiguous point constellations. Equipped with cutting and coalescing operations the algorithm can handle open surfaces and create arbitrary topologies. It can easily be modified to account for individually desired reconstruction behavior. The resulting surfaces are soundly oriented on a local as well as on a global level even for extremely challenging point clouds. The current surface estimate quickly adapts to the demanded topology even for thin surface areas.

Concerning mesh quality, flexibility, universality, and robustness, it clearly outperforms classical reconstruction approaches.

8.3 Future Work

The presented approach has a low average case runtime complexity, but is still relatively slow when, for instance, compared to a reconstruction approach using linear base functions. To decrease the runtime of the process, the iterative refinement could be exchanged with a recursive refinement process as presented in [HK06]. However, this might change the essence of the algorithm entirely. The efficiency could also be improved using the parallelized GNG algorithm. The most important GNG disadvantage is the absence of an actual surface model. This could be solved by using an implicit surface type as suggested in [Fri96].
The presented approach uses local optimization operations and therefore cannot set surface structures into a global context as, for instance, a model based approach does, where information of completely missing surface regions can be regained. The presented approach gathers information about the surface under investigation by setting samples in relation to a current surface estimate. A current surface estimate is a global structure, since it is an estimate of the entire surface. However, the refinement operations put samples only in relation to locally limited areas of the surface. To remove this limitation, again a semi-local processing strategy could be applied. The degree of *locality* could vary when putting samples in relation to the surface. If a sample lies on a flat surface, for instance, the given information is likely to be sufficient. If it lies on a curved distorted surface, the investigated surface area could be expanded. This would increase the degree to which the current surface estimate is “interpreted” when information is gathered.
The SGC approach with its differentiated vertex behaviors has proven to increase the reconstruction quality. The more specific types of behavior are created, the more specific cases can be treated. With the additional information available in the GSS concept, the algorithm behavior could be differentiated in many novel ways. However, creating such heuristics based behaviors is very time consuming and might quickly lead to a “trial and error” type of approach. Such an approach is likely to become unsystematic and thereby hard to test and even harder to verify theoretically. An approach based on extremely diversified algorithm behaviors would need to be created and tested automatically to ensure systematic results instead of “ad hoc” solutions. The presented filter chain concept already holds the potential for the automatic testing of different algorithm configurations. A mechanism to automatically create different filter operations would be needed to diversify the algorithm behavior.
An approach able to adapt its operations specifically toward the surface reconstruction task would need point clouds with ground truth data. It would also require having a *supervised* training phase where behaviors are learned, while the actual reconstruction process would be performed as an *unsupervised* learning approach.
If an algorithm would specifically mimic the behavior of a scanning device including aspects such as noise and varying sample densities, challenging point clouds could be created from a virtual surface. Such synthetic point clouds could be used for the training phase where the virtual surface then would represent the ground truth data.
Where in classic approaches, handling noise generally compromises surface detail, such an approach could offer individually fitted algorithm behaviors. Thus, noisy regions could be treated differently from the rest of the point cloud.

8.4 Epilog

When investigating a subspace with an *iterative refinement* approach, the subspace is molded from a structure of interconnected base units. The task of fitting the structure into the possibly complex shaped subspace is solved as a result of *emergence*, where the structure's base units themselves just follow simple *rules*, but create complex *behavior* as a result of their interaction.

The *plasticity* of this *behavior* is determined by the *plasticity* of the *rules* applied.

Smart growing cells allow for greater *rule* variety, filter chains enable automated *rule* adjustment and exchange, semi-local processing makes it possible to seamlessly scale the impact of *rules* from local to global, and growing surface structures introduce a learning scheme to adjust *rules* specifically to an investigated subspace.

In surface reconstruction, a 2D subspace is investigated and a mesh is fitted as a structural basis. While the abilities of the presented reconstruction method result from the novel *rule plasticity* suggested in this thesis, the complex *behavior* this *plasticity* gives rise to does not result from the reconstruction task.

Thus, the investigated subspace in this thesis might only be one of many to come.

Notation

Mathematical Symbols

Basics

b	A scalar value $b \in \mathbb{R}$. **[**x**]** The suggested parameter setting for b is x.
$\overline{b}$	The average of all b-values.
$\mathbf{b}$	A vector $\mathbf{b} = (x, y, z)$ and $\mathbf{b} \in \mathbb{R}^3$.
$\mathbf{b}_x$	A specific vector with the index x from a set of vectors.
$\mathscr{B}$	A set of elements $\mathscr{B} = \{a, b, c\}$.
$\lvert\mathscr{B}\rvert$	The number of elements in $\mathscr{B}$.
$\mathbf{b}\triangleright$	Singly linked list element.
$\triangleleft\mathbf{b}\triangleright$	Doubly linked list element.
$\mathbf{B}\triangleright$	Connected singly linked list elements.
$\triangleleft\mathbf{B}\triangleright$	Connected doubly linked list elements.

Surface Reconstruction

$\mathscr{P}$	Input point cloud of the surface reconstruction process. A finite collection of scattered surface samples $\mathscr{P} = \{\mathbf{p}_1 ... \mathbf{p}_n\}$ in 3D space $\mathbb{R}^3$.
$\mathscr{P}_{ori}$	A point cloud $\mathscr{P}$ where every sample additionally possesses a normal vector, which indicates its orientation.
$\mathscr{S}_{phy}$	A real world physical surface that has been scanned and is proposed to be digitalized in a reconstruction process.
$\mathscr{S}$	The resulting surface model produced by a reconstruction.

Mesh

$\mathbf{v}$	A mesh vertex $\mathbf{v} \in \mathscr{M}$.
$\mathbf{e}$	An edge connecting two vertices $\mathbf{e} = (\mathbf{v}_x, \mathbf{v}_y),\ \mathbf{v}_x \neq \mathbf{v}_y$ and $\mathbf{v}_x, \mathbf{v}_y \in \mathscr{M}$.
$\mathbf{t}$	A mesh triangle $\mathbf{t} = (\mathbf{v}_x, \mathbf{v}_y, \mathbf{v}_z),\ \mathbf{v}_x \neq \mathbf{v}_y \neq \mathbf{v}_z$ and $\mathbf{v}_x, \mathbf{v}_y, \mathbf{v}_z \in \mathscr{M}$.
$\mathscr{M}$	A mesh consisting of vertices $\mathscr{M} = \{\mathbf{v}_1 ... \mathbf{v}_n\}$. Between those vertices connections to facets exist, such as triangles or quads.
$\mathscr{T}$	All triangles in a mesh $\mathscr{T} = \{\mathbf{t}_1 ... \mathbf{t}_n\}$.

Growing Cell Structures

n_{final}	The number of vertices in $\mathscr{M}$ that are the abort criterion for a process.
t	The current iteration count of a process.
$\mathbf{p}_x$	The currently selected and processed random sample from $\mathscr{P}$.
$\mathbf{v}_x$	A specific vertex from $\mathscr{M}$ with the index x.
$\mathbf{n}_x$	The surface normal of $\mathbf{v}_x$, which indicates the surface orientation at position $\mathbf{v}_x$.
$\mathbf{bn}_x$	The boundary normal of $\mathbf{v}_x$, which indicates how the surface at position $\mathbf{v}_x$ would progress at the boundaries cutting edge.
$\mathbf{v}_{opp}$	A specific vertex from $\mathscr{M}$ that is in some way "opposite" to $\mathbf{v}_x$, *e.g.*, as a coalescing partner for $\mathbf{v}_x$.
$\mathscr{N}_x$	The neighboring vertices of vertex $\mathbf{v}_x$.
sc_x	The signal counter of vertex $\mathbf{v}_x$.
min_{sc}	If a signal counter comes below this threshold, it is deleted. [0.3]
lr	The learning rate determines how much a winning vertex is moved toward a sample. [0.1]
lrn	The neighbor learning rate determines how much the neighbors of a winning vertex are moved. [0.08]
c_{add}	Number that determines how many times the basic algorithm step is performed, before a new vertex is added. [100]
c_{del}	Number that determines how many times the basic algorithm step is performed, before a vertex is removed. [500]
β	Value that determines how much all signal counters are constantly decreased.
γ	The number of times a vertex can be missed before reaching min_{sc}. [7]
$curv_x$	The curvature of $\mathbf{v}_x$ calculated as one minus the average scalar product of $\mathbf{n}_x$ and the normals of its surrounding vertices $\mathscr{N}_x$.
$\overline{curv_{\mathscr{M}}}$	The average curvature of all vertices in $\mathscr{M}$.
σ_{curv}	The standard deviation of the curvature.

The Filter Chain Concept

$\mathscr{V}_{filt}$	A set of vertices that have been exposed to changes and are about to be tested for certain undesired or improvable structures.
$\mathbf{F}\triangleright_{filt}$	A list of filters vertices have to pass after they have been edited.
$valc(\mathbf{v})$	The valence, *i.e.*, the number of edges connected to vertex $\mathbf{v}$.

Minimal Edge Front

n_{exp}	Number of expansions to be performed.
$\mathscr{V}_{front}$	Collection of vertices representing the current front.
$\mathscr{V}_{old}$	Collection of vertices surpassed by a VF.
$\mathscr{V}_{new}$	Collection of vertices currently added during the expansion process of a VF.
$\mathscr{E}_{front}$	Collection of edges representing the current edge front.
$\mathscr{E}_{new}$	Collection of edges representing the new edge front in progress.
$\triangleleft\mathbf{e}\triangleright_x$	Specific element of an edge path in a MEF with index x.
$next(\triangleleft\mathbf{e}\triangleright_x)$	The edge path element subsequent to $\triangleleft\mathbf{e}\triangleright_x$.
$previous(\triangleleft\mathbf{e}\triangleright_x)$	The edge path element previous to $\triangleleft\mathbf{e}\triangleright_x$.
$\triangleleft\mathbf{E}\triangleright_x$	The edge path running in front of $previous(\triangleleft\mathbf{e}\triangleright_x)$ and $\triangleleft\mathbf{e}\triangleright_x$.

Minimal Distance Front

d_{max}	The maximum allowed expansion distance.
$\mathscr{D}_{exp}$	All currently possible expansions ordered by their anticipated distance to the starting point.
d_{start}	The on-surface distance back to the starting point from a front vertex.
$\mathbf{d}_{start}$	The normalized on-surface direction back to the starting point from a front vertex.
$\mathbf{v}_{sub}$	A vertex associated with a subdivision in between two edge associated subdivisions.
$\mathbf{e}_{sub}$	An edge associated with a certain subdivision.
$\mathbf{t}_{sub}$	A triangle connected at the backside of a subdivision $\mathbf{e}_{sub}$.
$\mathbf{v}_{left}$	The vertex to the left of a subdivision edge $\mathbf{e}_{sub}$.
$\mathbf{v}_{right}$	The vertex to the right of a subdivision edge $\mathbf{e}_{sub}$.
$\mathbf{p}_{connect}$	A point on an edge $\mathbf{e}_{sub}$ connecting a vertex $\mathbf{v}_x$ with the front.

Growing Surface Structures

$[\mathbf{s}]$	A mesh structure, such as either a vertex, an edge, or a triangle.
$\mathscr{N}_{[\mathbf{s}]}$	All adjacent vertices to structure $[\mathbf{s}]$.
$\mathbf{t}_x$	A specific triangle from the triangles $\mathscr{T}$ with the index x.
err_x	The approximation error of triangle $\mathbf{t}_x$.
$size(\mathbf{t})$	The surface area of triangle $\mathbf{t}$.
$\overline{s_{\mathscr{T}}}$	The average surface area of all triangles in $\mathscr{T}$.
d_x	The Euclidian distance of a sample $\mathbf{p}_x$ to the closest surface structure $[\mathbf{s}]$.
$\overline{d_{\mathscr{P}}}$	The average sample-to-surface distance.
∇n	The dot product of vertex normals of an edge about to be collapsed to reduce geometric redundancy.
$max_{\nabla n}$	A threshold for ∇n to surpass to allow for an edge collapse operation. [0.9]
a	The age of an edge in GNG or a triangle in GSS.
max_a	Maximum age for a triangle to reach before it is considered misplaced. [10]
β	The aging rate for not winning triangles.
γ	The number of times a triangle can be missed before reaching max_a. [7]
λ	The half-life of a distance error actualization. It takes λ updates until a distance value d_x only has half of its impact on err. [9]
lim_{skip}	If a distance value d_x divided by the average distance $\overline{d_{\mathscr{P}}}$ drops below this threshold, the geometric properties of $\mathscr{M}$ remain unchanged. [0.9]
lim_{single}	If a distance value d_x divided by the average distance $\overline{d_{\mathscr{P}}}$ drops below this threshold, the general surface is considered correct and only a vertex instead of an edge or triangle is adapted. [1.2]
n_v	The number of vertices used in the search process of the GSS algorithm to find the closest surface structure $[\mathbf{s}]$. [3]

Abbreviations

ACO	aggressive cut out
AI	artificial intelligence
ANN	artificial neural network
CAD	computer-aided design
CAM	computer-aided manufacturing
CT	computed tomography
EMST	Euclidian minimal spanning tree
FMM	fast marching method
GCS	growing cell structures
GNG	growing neural gas
GSS	growing surface structures
LBG	Linde-Buzo-Gray
LOD	level of detail
MA	medial axis
MAT	medial axis transform
MDF	minimal distance front
MEF	minimal edge front
MLS	moving least squares
MRI	magnetic resonance imaging
MS	medial scaffold
MVS	multi-view stereo
NG	neural gas
NURBS	non-uniform rational B-splines
PCA	principal component analysis
RANSAC	random sample consensus
RBF	radial base function
RIMLS	robust implicit moving least squares
SGC	smart growing cells
SOM	self-organizing map
SVM	support vector machine
TOF	time of flight
VF	vertex front

Illustration Key

Problem Analysis

●	Sample
—	Actual Surface
- -	Conceivable Surface
➡	Normal
—	Created Surface
●	Outside
●	Inside
●	Search Center
●	3D Search
●	Vertex
—	Edge
- - -	Boundary Edge

Growing Cell Structures

●	Vertex on Sample
●	Selected Sample
➡	Vertex Movement
●	Winnig Vertex
⇨	Neighbor Movement
●	Neighbor Vertex
+1	Increase Signal Counter
➡	Operation Direction
●	Vertex of Max Signal Count
—	Split Edge
○	New/Deleted Vertex
▲	New/Deleted Triangle
—	New/Deleted Edge

Minimal Edge Front

○	Processed Vertex
○	Processed Surface
◒	Current Front Vertex
—	Current Front Edge
—	*previous*($\triangleleft e \triangleright_x$)
—	$\triangleleft e \triangleright_x$
●—●	$\triangleleft E \triangleright_x$ Not Minimized
●—●	$\triangleleft E \triangleright_x$ Minimized
●	Skipped Vertex
X	Deleted from Data Structures
●	Added Front Vertex
—	New Front Edges

Twist Solving

●	Surface Outside
●	Surface Inside
●	v_x
●	v_{opp}

Growing Surface Structures

→	Distance to Closest Vertex
→	Distance to Closest Surface

Bibliography

[AB10a] Hendrik Annuth and Christian-A. Bohn. Surface reconstruction with smart growing cells. In *Intelligent Computer Graphics 2010*, volume 321 of *Studies in Computational Intelligence*, pages 47–66. Springer Verlag, 2010.

[AB10b] Hendrik Annuth and Christian A. Bohn. Tumble tree: reducing complexity of the growing cells approach. In *Proceedings of the 20th international conference on Artificial neural networks: Part III*, ICANN'10, pages 228–236, Berlin, Heidelberg, 2010. Springer-Verlag.

[AB11] Hendrik Annuth and Christian-A. Bohn. Reconstruction for Virtual Reality Scenes. In *Proceedings of Virtuelle und Erweiterte Realität*, volume 8, pages 121–133. SHAKER Verlag, 2011.

[AB12a] Hendrik Annuth and Christian-A. Bohn. Approximation of geometric structures with growing cell structures and growing neural gas - a performance comparison. In *Proceedings of International Conference on Neural Computing Theory and Applications (NCTA 2012)*, pages 552–557. SciTePress, 2012.

[AB12b] Hendrik Annuth and Christian-A. Bohn. Resolving Twisted Surfaces within an Iterative Refinement Surface Reconstruction Approach. In *Proceedings of Vision, Modeling, and Visualization (VMV 2012)*, pages 175–182, 2012.

[AB12c] Hendrik Annuth and Christian-A. Bohn. Smart growing cells: Supervising unsupervised learning. In *Computational Intelligence*, volume 399 of *Studies in Computational Intelligence*, pages 405–420. Springer Berlin / Heidelberg, 2012.

[AB13] Hendrik Annuth and Christian-A. Bohn. Growing surface structures. In *Proceedings of International Conference on Neural Computing Theory and Applications (NCTA 2013)*, page 7. SciTePress, 2013.

[AB14a] Hendrik Annuth and Christian-A. Bohn. Geodesic mesh processing with edge-front based data structures. In *Proceedings of International Conference on Computer Graphics Theory and Applications (GRAPP 2014)*, page 43. SciTePress, 2014.

[AB14b] Hendrik Annuth and Christian-A. Bohn. Growing surface structures: A topology focused learning scheme. In *Computational Intelligence*, Studies in Computational Intelligence. Springer Berlin / Heidelberg, 2014.

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

[ABE98] Nina Amenta, Marshall Bern, and David Eppstein. The crust and the beta-skeleton: Combinatorial curve reconstruction. *Graphic Models and Image Processing*, 60(2 of 2):125–135, 1998.

[ABK98] Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 415–421, New York, NY, USA, 1998. ACM.

[ACDL00] N. Amenta, S. Choi, T. K. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. In *Proceedings of the sixteenth annual symposium on Computational geometry*, SCG '00, pages 213–222, New York, NY, USA, 2000. ACM.

[ACK01] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, SMA '01, pages 249–266, New York, NY, USA, 2001. ACM.

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

[AKM$^+$06] M. Attene, S. Katz, M. Mortara, G. Patane, M. Spagnuolo, and A. Tal. Mesh segmentation - a comparative study. In *Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006*, SMI '06, page 7, Washington, DC, USA, 2006. IEEE Computer Society.

[AS96] Maria-Elena Algorri and Francis Schmitt. Surface Reconstruction from Unstructured 3D Data. *Computer Graphics Forum*, 15(1):47–60, 1996.

[BF02] Jacob Barhak and Anath Fischer. Adaptive reconstruction of freeform objects with 3d som neural network grids. *Computers & Graphics*, 26(5):745–751, 2002.

[BG67] G. Backus and F. Gilbert. Numerical applications of a formalism for geophysical inverse problems. *Geophys.J.R oy.Astr on Soc.*, Vol. 13:247–276, 1967.

[BH93] A. Baader and G. Hirzinger. Three-dimensional surface reconstruction based on a self-organizing feature map. In *Proc. 6th Int. Conf. Advan. Robotics*, pages 273–278, 1993.

[BK04] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(9):1124–1137, September 2004.

[BK07] David Bommes and Leif Kobbelt. Accurate computation of geodesic distance fields for polygonal curves on triangle meshes. In Hendrik P. A. Lensch, Bodo Rosenhahn, Hans-Peter Seidel, Philipp Slusallek, and Joachim Weickert, editors, *Proceedings of Vision, Modeling, and Visualization (VMV 2007)*, pages 151–160. Aka GmbH, 2007.

[BKBH09] Matthew Bolitho, Michael Kazhdan, Randal Burns, and Hugues Hoppe. Parallel poisson surface reconstruction. In *Proceedings of the 5th International Symposium on Advances in Visual Computing: Part I*, ISVC '09, pages 678–689, Berlin, Heidelberg, 2009. Springer-Verlag.

[Bli82] James F. Blinn. A generalization of algebraic surface drawing. *ACM Trans. Graph.*, 1(3):235–256, July 1982.

[BLP$^+$12] D. Bommes, B. Lévy, N. Pietroni, E. Puppo, C. Silv a, M. Tarini, and D. Zorin. State of the art in quad meshing. In *Eurographics STARS*, 2012.

[BMR$^+$99] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Cláudio Silva, and Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, October 1999.

[BMSW11] Prosenjit Bose, Anil Maheshwari, Chang Shu, and Stefanie Wuhrer. A survey of geodesic paths on 3d surfaces. *Comput. Geom. Theory Appl.*, 44(9):486–498, November 2011.

[Boi84] Jean-Daniel Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Trans. Graph.*, 3(4):266–286, October 1984.

[BZ00] C. Baillard and A. Zisserman. A plane-sweep strategy for the 3D reconstruction of buildings from multiple images. In *ISPRS Congress and Exhibition*, Amsterdam, 2000.

[CBC$^+$01] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 67–76, New York, NY, USA, 2001. ACM.

[CBM$^+$03] J. C. Carr, R. K. Beatson, B. C. McCallum, W. R. Fright, T. J. McLennan, and T. J. Mitchell. Smooth surface reconstruction from noisy range data. In *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, GRAPHITE '03, page 119, New York, NY, USA, 2003. ACM.

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

[CG04] Frederic Cazals and Joachim Giesen. Delaunay Triangulation Based Surface Reconstruction: Ideas and Algorithms. Technical Report RR-5393, INRIA, November 2004.

[CH90] Jindong Chen and Yijie Han. Shortest paths on a polyhedron. In *SCG 90: Proceedings of the Sixth Annual Symposium on Computational geometry*, pages 360–369. ACM Press, 1990.

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

[CLK09] Ming-Ching Chang, Frederic Fol Leymarie, and Benjamin B. Kimia. Surface reconstruction from point clouds by transforming the medial scaffold. *Comput. Vis. Image Underst.*, 113(11):1130–1146, November 2009.

[CM95] Yang Chen and Gérard Medioni. Description of complex objects from multiple range images using an inflating balloon model. *Comput. Vis. Image Underst.*, 61:325–334, May 1995.

[CSD04] David Cohen-Steiner and Frank Da. A greedy delaunay-based surface reconstruction algorithm. *Vis. Comput.*, 20(1):4–16, April 2004.

[CWW12] Keenan Crane, Clarisse Weischedel, and Max Wardetzky. Geodesics in heat. *CoRR*, abs/1204.6216, 2012.

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

[DG06] Tamal K. Dey and Samrat Goswami. Provable surface reconstruction from noisy samples. *Comput. Geom. Theory Appl.*, 35:124–141, August 2006.

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

[DHOS07] Joel II Daniels, Linh K. Ha, Tilo Ochotta, and Claudio T. Silva. Robust smooth feature extraction from point clouds. In *Proceedings of the IEEE International Conference on Shape Modeling and Applications 2007*, SMI '07, pages 123–136, Washington, DC, USA, 2007. IEEE Computer Society.

[dRAdLN07] Renata L. M. do Rego, Aluizio F. R. Araújo, and Fernando Buarque de Lima Neto. Growing self-organizing maps for surface reconstruction from unstructured point clouds. In *IJCNN*, pages 1900–1905. IEEE, 2007.

[DRADLN10] Renata Lúcia Mendonça Ernesto Do Rêgo, Aluizio Fausto Ribeiro Araújo, and Fernando Buarque De Lima Neto. Growing self-reconstruction maps. *Trans. Neur. Netw.*, 21(2):211–223, February 2010.

[EBV05] Jordi Esteve, Pere Brunet, and Alvar Vinacua. Approximation of a variable density cloud of points by shrinking a discrete membrane. *Comput. Graph. Forum*, pages 791–807, 2005.

[EH96] Matthias Eck and Hugues Hoppe. Automatic reconstruction of b-spline surfaces of arbitrary topological type. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 325–334, New York, NY, USA, 1996. ACM.

[EM92] Herbert Edelsbrunner and Ernst P. Mücke. Three-dimensional alpha shapes. In *Volume Visualization*, pages 75–82, 1992.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[FCOS05] Shachar Fleishman, Daniel Cohen-Or, and Cláudio T. Silva. Robust moving least-squares fitting with sharp features. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 544–552, New York, NY, USA, 2005. ACM.

[Fre14] Free Software Foundation. Gnu free documentation license. `https://gnu.org/licenses/fdl.html/`, 2014.

[Fri93] Bernd Fritzke. Growing cell structures - a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7:1441–1460, 1993.

[Fri95] Bernd Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, Cambridge MA, 1995.

[Fri96] Bernd Fritzke. Automatic construction of radial basis function networks with the growing neural gas model and its relevance for fuzzy logic. In *Proceedings of the 1996 ACM symposium on Applied Computing*, SAC '96, pages 624–627, New York, NY, USA, 1996. ACM.

[FW91] Bernd Fritzke and Peter Wilke. FLEXMAP—A neural network with linear time and space complexity for the traveling salesman problem. In *Proceedings of the International Joint Conference on Neural Networks* (Singapore), pages 929–934. Piscataway, NJ: IEEE, 1991.

[GK02] M. Gopi and Shankar Krishnan. A fast and efficient projection-based approach for surface reconstruction. In *Proceedings of the 15th Brazilian Symposium on Computer Graphics and Image Processing*, SIBGRAPI '02, pages 179–186, Washington, DC, USA, 2002. IEEE Computer Society.

[GSH$^+$07] Ran Gal, Ariel Shamir, Tal Hassner, Mark Pauly, and Daniel Cohen-Or. Surface reconstruction using local shape priors. In *Proceedings of the fifth Eurographics symposium on Geometry processing*, SGP '07, pages 253–262, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[HDD$^+$92] Hugues Hoppe, Tony DeRose, Tom Duchamp, John Alan McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In James J. Thomas, editor, *SIGGRAPH*, pages 71–78. ACM, 1992.

[HDD$^+$93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 19–26, New York, NY, USA, 1993. ACM.

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

[Heb49] D. O. Hebb. *The organization of behavior: a neuropsychological theory*. John Wiley & Sons, New York, 1949.

[HF08] Y. Holdstein and A. Fischer. Three-dimensional surface reconstruction using meshing growing neural gas (mgng). *Vis. Comput.*, 24:295–302, March 2008.

[HG01] Andreas Hubeli and Markus Gross. Multiresolution feature extraction for unstructured meshes. In *Proceedings of the conference on Visualization '01*, VIS '01, pages 287–294, Washington, DC, USA, 2001. IEEE Computer Society.

[HK06] Alexander Hornung and Leif Kobbelt. Robust reconstruction of watertight 3d models from non-uniformly sampled point clouds without normal information. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, SGP '06, pages 41–50, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[HLZ$^+$09] Hui Huang, Dan Li, Hao Zhang, Uri Ascher, and Daniel Cohen-Or. Consolidation of unorganized point clouds for surface reconstruction. *ACM Trans. Graph.*, 28(5):176:1–176:7, December 2009.

[HM02] Jianbing Huang and Chia-Hsiang Menq. Combinatorial manifold mesh reconstruction and optimization from unorganized points with arbitrary topology. *Computer-Aided Design*, pages 149–165, 2002.

[HSKK01] Masaki Hilaga, Yoshihisa Shinagawa, Taku Kohmura, and Tosiyasu L. Kunii. Topology matching for fully automatic similarity estimation of 3d shapes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 203–212, New York, NY, USA, 2001. ACM.

[HV98] Miklós Hoffmann and Lajos Várady. Free-form surfaces for scattered data by neural networks. *Journal for Geometry and Graphics*, 2:1–6, 1998.

[IJL$^+$04] I.P. Ivrissimtzis, W.-K. Jeong, S. Lee, Y. Lee, and H.-P. Seidel. Neural meshes: surface reconstruction with a learning algorithm. Research Report MPI-I-2004-4-005, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, October 2004.

[IJS03a] I. P. Ivrissimtzis, W-K. Jeong, and H-P. Seidel. Using growing cell structures for surface reconstruction. In *SMI '03: Proceedings of the Shape Modeling International 2003*, page 78, Washington, DC, USA, 2003. IEEE Computer Society.

[IJS03b] Ioannis Ivrissimtzis, Won-Ki Jeong, and Hans-Peter Seidel. Neural meshes: Statistical learning methods in surface reconstruction. Technical Report MPI-I-2003-4-007, Max-Planck-Institut f́ur Informatik, Saarbrücken, April 2003.

[ILL$^+$04] Ioannis Ivrissimtzis, Yunjin Lee, Seungyong Lee, Won-Ki Jeong, and Hans-Peter Seidel. Neural mesh ensembles. In *3DPVT '04: Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium*, pages 308–315, Washington, DC, USA, 2004. IEEE Computer Society.

[IST13] ISTI - CNR Research Center. Meshlab homepage. `http://meshlab.sourceforge.net/`, 2013.

[JBS$^+$06] P. Jenke, M. Bokeloh, A. Schilling, W. Straßer, and M. Wand. Bayesian point cloud reconstruction. In *EUROGRAPHICS 2006*, 2006.

[JHB95] Daniel crevier. ai: The tumultuous history of the search for artificial intelligence. ny: Basic books, 1993. 432 pp. (reviewed by charles fair). *Journal of the History of the Behavioral Sciences*, 31(3):273–278, 1995.

[JIS03] W.-K. Jeong, I.P. Ivrissimtzis, and H.-P. Seidel. Neural meshes: Statistical learning based on normals. *Computer Graphics and Applications, Pacific Conference on*, 0:404, 2003.

[KB04] Leif Kobbelt and Mario Botsch. A survey of point-based techniques in computer graphics. *Comput. Graph.*, 28(6):801–814, December 2004.

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

[KL96] Venkat Krishnamurthy and Marc Levoy. Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH*, pages 313–324, 1996.

[KO00] Biliana Kaneva and Joseph O'Rourke. An implementation of chen & han's shortest paths algorithm. In *CCCG*, 2000.

[Koh82] Teuvo Kohonen. Self-Organized Formation of Topologically Correct Feature Maps. *Biological Cybernetics*, 43:59–69, 1982.

[Kol82] Gina Kolata. How can computers get common sense? *Science*, 217(4566):1237–1238, 1982.

[KS98] R. Kimmel and J. A. Sethian. Computing geodesic paths on manifolds. In *Proc. Natl. Acad. Sci. USA*, pages 8431–8435, 1998.

[KS00] Takashi Kanai and Hiromasa Suzuki. Approximate shortest path on polyhedral surface based on selective refinement of the discrete graph and its applications. In *Proceedings of the Geometric Modeling and Processing 2000*, GMP '00, page 241, Washington, DC, USA, 2000. IEEE Computer Society.

[KT03] Sagi Katz and Ayellet Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 954–961, New York, NY, USA, 2003. ACM.

[KY05] Chuan-Chu Kuo and Hong-Tzong Yau. A delaunay-based region-growing approach to surface reconstruction from unorganized points. *Comput. Aided Des.*, 37(8):825–835, July 2005.

[LBG80] Y. Linde, A. Buzo, and R. M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28:84–95, 1980.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, August 1987.

[LCOL07] Yaron Lipman, Daniel Cohen-Or, and David Levin. Data-dependent mls for faithful surface approximation. In *Proceedings of the fifth Eurographics symposium on Geometry processing*, SGP '07, pages 59–67, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[LCOLTE07] Yaron Lipman, Daniel Cohen-Or, David Levin, and Hillel Tal-Ezer. Parameterization-free projection for geometry reconstruction. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

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

[Lev99] M. Levoy. The digital michelangelo project. In *Proceedings. Second International Conference on 3-D Digital Imaging and Modeling*, pages 2 –11, 1999.

[Lev03] D. Levin. Mesh-independent surface interpolation. *Geometric Modeling for Scientific Visualization*, pages 37–49, 2003.

[LHL$^+$12] Kuan-Yuan Lin, Chung-Yi Huang, Jiing-Yih Lai, Yao-Chen Tsai, and Wen-Der Ueng. Automatic reconstruction of b-spline surfaces with constrained boundaries. *Comput. Ind. Eng.*, 62(1):226–244, February 2012.

[LMS97] Mark Lanthier, Anil Maheshwari, and Jörg-Rüdiger Sack. Approximating weighted shortest paths on polyhedral surfaces. In *Proceedings of the thirteenth annual symposium on Computational geometry*, SCG '97, pages 485–486, New York, NY, USA, 1997. ACM.

[LS81] P. Lancaster and K. Salkauskas. Surfaces Generated by Moving Least Squares Methods. *Mathematics of Computation*, 37(155):141–158, 1981.

[LTJW07] Ligang Liu, Chiew-Lan Tai, Zhongping Ji, and Guojin Wang. Non-iterative approach for global mesh optimization. *Comput. Aided Des.*, 39(9):772–782, September 2007.

[LYO$^+$10] Yotam Livny, Feilong Yan, Matt Olson, Baoquan Chen, Hao Zhang, and Jihad El-Sana. Automatic reconstruction of tree skeletal structures from point clouds. *ACM Trans. Graph.*, 29(6):151:1–151:8, December 2010.

[Mac67] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[MAVdF05] Boris Mederos, Nina Amenta, Luiz Velho, and Luiz Henrique de Figueiredo. Surface reconstruction from noisy point clouds. In *Proceedings of the third Eurographics symposium on Geometry processing*, SGP '05, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

[MBL$^+$91] James V. Miller, David E. Breen, William E. Lorensen, Robert M. O'Bara, and Michael J. Wozny. Geometrically deformed models: a method for extracting closed geometric models form volume data. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '91, pages 217–226, New York, NY, USA, 1991. ACM.

[Men01] Robert Mencl. Reconstruction of surfaces from unorganized three-dimensional point clouds. phd, Universität Dortmund, 2001.

[MHH08] Markus Melato, Barbara Hammer, and Kai Hormann. Neural gas for surface reconstruction. *IfI Technical Report Series*, 2008.

[MKN$^+$04] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa. Point based animation of elastic, plastic and melting objects. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '04, pages 141–151, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[MMP87] Joseph S. B. Mitchell, David M. Mount, and Christos H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16(4):647–668, August 1987.

[MP88] Marvin Minsky and Seymour Papert. *Perceptrons : an introduction to computational geometry*. The MIT Press, Cambridge (Mass.), London, 1988.

[MS91] T. Martinetz and K. Schulten. A ”Neural-Gas” Network Learns Topologies. *Artificial Neural Networks*, I:397–402, 1991.

[MS94] T. Martinetz and K. Schulten. Topology representing networks. *Neural Networks*, 7(3):507–522, 1994.

[MSP+08] F. Mari, Jo José Hiroki Saito, Gustavo Poli, Marcelo R. Zorzan, and Alexandre L. M. Levada. Improving the neural meshes algorithm for 3d surface reconstruction with edge swap operations. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 1236–1240, New York, NY, USA, 2008. ACM.

[Mur91] Shigeru Muraki. Volumetric shape description of range data using "blobby model". In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 227–235, New York, NY, USA, 1991. ACM.

[NY06] T. S. Newman and H. Yi. A survey of the marching cubes algorithm. *Computers & Graphics*, 30(5), 2006.

[OBA+03] Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. Multi-level partition of unity implicits. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 463–470, New York, NY, USA, 2003. ACM.

[OGG09] A. C. Öztireli, G. Guennebaud, and M. Gross. Feature preserving point set surfaces based on non-linear kernel regression. *Computer Graphics Forum*, 28(2), 2009.

[OTC+10] Guilherme N. Oliveira, Rafael P. Torchelsen, Joao L. D. Comba, Marcelo Walter, and Rui Bastos. Geotextures: A multi-source geodesic distance field approach for procedural texturing of complex meshes. In *Proceedings of the 2010 23rd SIBGRAPI Conference on Graphics, Patterns and Images*, SIBGRAPI '10, pages 126–133, Washington, DC, USA, 2010. IEEE Computer Society.

[PBP02] Hartmut Prautzsch, Wolfgang Boehm, and Marco Paluszny. *Bezier and B-Spline Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[PLK05] Si Hyung Park, Seoung Soo Lee, and Jong Hwa Kim. A surface reconstruction algorithm using weighted alpha shapes. In *Proceedings of the Second international conference on Fuzzy Systems and Knowledge Discovery - Volume Part I*, FSKD'05, pages 1141–1150, Berlin, Heidelberg, 2005. Springer-Verlag.

[PLL12] Min Ki Park, Seung Joo Lee, and Kwan H. Lee. Multi-scale tensor voting for feature extraction from unstructured point clouds. *Graph. Models*, 74(4):197–208, July 2012.

[PMW$^+$08] Mark Pauly, Niloy J. Mitra, Johannes Wallner, Helmut Pottmann, and Leonidas J. Guibas. Discovering structural regularity in 3d geometry. *ACM Trans. Graph.*, 27(3):43:1–43:11, August 2008.

[PT97] Les Piegl and Wayne Tiller. *The NURBS book (2nd ed.)*. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[RMB$^+$08] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Emanuel Dolha, and Michael Beetz. Towards 3D point cloud based object maps for household environments. *Robot. Auton. Syst., Special Issue on Semantic Knowledge in Robotics*, 56(11):927–941, 2008.

[Rot03] Franz Rottensteiner. Automatic generation of high-quality building models from lidar data. *IEEE Comput. Graph. Appl.*, 23(6):42–50, November 2003.

[SACO04] Andrei Sharf, Marc Alexa, and Daniel Cohen-Or. Context-based surface completion. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 878–887, New York, NY, USA, 2004. ACM.

[SCD$^+$06] Steven M. Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*, CVPR '06, pages 519–528, Washington, DC, USA, 2006. IEEE Computer Society.

[SDK09] Ruwen Schnabel, Patrick Degener, and Reinhard Klein. Completion and reconstruction with primitive shapes. *Computer Graphics Forum (Proc. of Eurographics)*, 28(2):503–512, March 2009.

[Set95] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. In *Proc. Nat. Acad. Sci*, pages 1591–1595, 1995.

[SLS$^+$06] Andrei Sharf, Thomas Lewiner, Ariel Shamir, Leif Kobbelt, and Daniel Cohen-Or. Competing fronts for coarse-to-fine surface reconstruction. In *Eurographics 2006 (Computer Graphics Forum)*, volume 25, pages 389–398, Vienna, october 2006. Eurographics.

[SOS05] Chen Shen, James F. O'Brien, and Jonathan R. Shewchuk. Interpolating and approximating implicit surfaces from polygon soup. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

[SS02] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision*, 47(1-3):7–42, April 2002.

[SSB05] Florian Steinke, Bernhard Schölkopf, and Volker Blanz. Support vector machines for 3d shape processing. *Comput. Graph. Forum*, pages 285–294, 2005.

[SSK$^+$05] Vitaly Surazhsky, Tatiana Surazhsky, Danil Kirsanov, Steven J. Gortler, and Hugues Hoppe. Fast exact and approximate geodesics on meshes. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 553–560, New York, NY, USA, 2005. ACM.

[SSZCO10] Shy Shalom, Ariel Shamir, Hao Zhang, and Daniel Cohen-Or. Cone carving for surface reconstruction. *ACM Trans. Graph.*, 29(6):150:1–150:10, December 2010.

[SWK07] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient ransac for point-cloud shape detection. *Computer Graphics Forum*, 26(2):214–226, June 2007.

[SWWK08] Ruwen Schnabel, Raoul Wessel, Roland Wahl, and Reinhard Klein. Shape recognition in 3d point-clouds. In V. Skala, editor, *The 16-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2008*. UNION Agency-Science Press, February 2008.

[SZBN03] Thomas W. Sederberg, Jianmin Zheng, Almaz Bakenov, and Ahmad Nasri. T-splines and t-nurccs. *ACM Trans. Graph.*, 22(3):477–484, July 2003.

[Tau95] Gabriel Taubin. A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 351–358, New York, NY, USA, 1995. ACM.

[TV91] D. Terzopoulos and M. Vasilescu. Sampling and Reconstruction with Adaptive Meshes. In *Proceedings of the 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 70–75, Lahaina, HI, 1991.

[TWAD09] Jane Tournois, Camille Wormser, Pierre Alliez, and Mathieu Desbrun. Interleaving delaunay refinement and optimization for practical isotropic tetrahedron mesh generation. *ACM Trans. Graph.*, 28(3):75:1–75:9, July 2009.

[VF05] Tamás Várady and Michael A. Facello. New trends in digital shape reconstruction. In *Proceedings of the 11th IMA international conference on Mathematics of Surfaces*, IMA'05, pages 395–412, Berlin, Heidelberg, 2005. Springer-Verlag.

[VHK99] Lajos Várady, Miklós Hoffmann, and Emőd Kovács. Improved free-form modelling of scattered data by dynamic neural networks. *Journal for Geometry and Graphics*, 3:177–183, 1999.

[VT92] M. Vasilescu and Demetri Terzopoulos. Adaptive meshes and shells: Irregular triangulation, discontinuities, and hierarchical subdivision. In *Proceedings of Computer Vision and Pattern Recognition conference*, pages 829–832. IEEE Computer Society Press, 1992.

[WALH13] Thomas Wiemann, Hendrik Annuth, Kai Lingemann, and Joachim Hertzberg. An evaluation of open source surface reconstruction software for robotic applications. In *16th International Conference on Advanced Robotics (ICAR 2013)*, pages 1–7. IEEE, 2013.

[XMQ04] Hui Xie, Kevin T. McDonnell, and Hong Qin. Surface reconstruction of noisy and defective data sets. In *Proceedings of the conference on Visualization '04*, VIS '04, pages 259–266, Washington, DC, USA, 2004. IEEE Computer Society.

[YLL$^+$09] Dong-Ming Yan, Bruno Lévy, Yang Liu, Feng Sun, and Wenping Wang. Isotropic remeshing with fast and exact computation of restricted voronoi diagram. In *Proceedings of the Symposium on Geometry Processing*, SGP '09, pages 1445–1454, Aire-la-Ville, Switzerland, Switzerland, 2009. Eurographics Association.

[Yu99] Yizhou Yu. Surface reconstruction from unorganized points using self-organizing neural networks. In *IEEE Visualization 99, Conference Proceedings*, pages 61–64, 1999.

[ZKK02] G. Zigelman, R. Kimmel, and N. Kiryati. Texture mapping using surface flattening via multidimensional scaling. *IEEE Transactions on Visualization and Computer Graphics*, 8(2):198–207, April 2002.

[ZOF01] Hong-Kai Zhao, Stanley Oshery, and Ronald Fedkiwz. Fast surface reconstruction using the level set method. In *VLSM '01: Proceedings of the IEEE Workshop on Variational and Level Set Methods*, 2001.