Diss. ETH No. 15134

Point Primitives for Interactive Modeling and Processing of 3D Geometry

A dissertation submitted to the
Federal Institute of Technology (ETH) of Zurich

for the Degree of
Doctor of Sciences

presented by
Mark Pauly
Diplom Informatiker University of Kaiserslautern
born 18. Februar 1974
citizen of Germany

accepted on the recommendation of
Prof. Dr. Markus Gross, examiner

Prof. Dr. Leif Kobbelt, co-examiner

Prof. Dr. Peter Schröder, co-examiner

2003

ABSTRACT

3D geometry has become increasingly popular as a new form of digital media. Similar to other types of media data, i.e., sound, images, and video, this requires tools to acquire, store, process, edit, and transmit 3D geometry. With increasing complexity of 3D geometric models and growing demand for advanced modeling functionality, significant effort is being devoted to the design of efficient, reliable, and scalable algorithms for digital geometry processing.

This thesis investigates the use of point primitives for 3D geometry processing and interactive modeling. Representing surfaces by point clouds allows direct processing of 3D scanner data, which avoids the need for surface reconstruction methods. The structural simplicity of point-based representations supports efficient re-sampling for extreme geometric deformations and topology changes. It also leads to concise algorithms that are well suited for hardware implementation.

The main focus of this thesis is on algorithms for shape and appearance modeling of surfaces represented by point clouds. This requires methods for local surface analysis and reconstruction, filtering, re-sampling, and estimation of the signed distance function.

Local surface analysis is based on a statistical operator applied to local neighborhoods of point samples. This enables efficient estimation of the tangent space of the underlying surface, as well as providing different approximations for surface curvature. To compute local approximations of a surface represented by point samples, an extension of the moving least squares surface model is presented that adapts to the local sampling density.

An important geometric processing tool is surface simplification. To reduce the complexity of point-sampled surfaces, various simplification methods are presented, including clustering, iterative point-pair contraction, and particle simulation. These methods are used to build hierarchies of surface approximations for efficient multi-level computations.

A multi-scale surface representation is introduced that enables sophisticated editing functionality at different approximation levels. Representing a point-sampled surface at different levels of geometric detail supports advanced filtering methods such as enhancement filters. Multi-scale methods are also applied to implement a feature extraction pipeline for point-sampled surfaces with particular emphasis on robustness and efficiency.

The above methods are integrated into a unified framework for point-based shape and appearance modeling. Shape modeling functionality includes boolean operations and free-form deformation, appearance editing comprises painting, texturing, sculpting, and filtering methods. Based on a dynamic sampling paradigm, this system defines a complete and versatile modeling environment for 3D content creation.

ZUSAMMENFASSUNG

3D Geometrie hat in den letzten Jahren eine vermehrte Verbreitung als neues, digitales Medium erfahren. Dies erfordert, ähnlich zu anderen Typen digitaler Medien wie Audio-, Bild-, oder Video-daten, Werkzeuge zur Akquisition, Speicherung, Verarbeitung, Editierung und Übertragung von 3D Geometrie. Mit wachsender Komplexität geometrischer Modelle und steigender Nachfrage nach fortgeschrittener Modellierungs-Funktionalität wird vermehrte Aufmerksamkeit der Entwicklung effizienter, zuverlässiger, und skalierbarer Algorithmen für die Verarbeitung geometrischer Modelle gewidmet.

Diese Dissertation untersucht die Verwendung von Punkt-Primitiven für die Verarbeitung geometrischer Daten. Die Darstellung von Flächen mittels Punktwolken erlaubt die direkte Verarbeitung von 3D Scanner-Daten, was den Einsatz von Oberflächenrekonstruktions-Verfahren erspart. Die strukturelle Einfachheit punkt-basierter Repräsentationen unterstützt effizientes Abtasten bei extremen geometrischen Deformationen oder Änderungen der Topologie, und führt zu Algorithmen, die sich gut für eine Implementierung in Hardware eignen.

Der Schwerpunkt dieser Dissertation liegt auf Algorithmen zur Modellierung von Flächen, die durch Punktwolken repräsentiert sind. Dazu werden Methoden zur lokalen Flächen-Analyse und -Rekonstruktion, Filterung, Abtastung, und Schätzung der Flächen-Distanzfunktion entwickelt.

Die lokale Flächen-Analyse basiert auf einem statistischen Operator, der auf lokale Punkt-Nachbarschaften angewendet wird. Dies erlaubt die effiziente Schätzung des Tangential-Raumes der unterliegenden Fläche und liefert unterschiedliche Näherungen der Krümmung. Eine Erweiterung der moving least squares Methode wird präsentiert, die eine lokale Approximationen einer Fläche, die durch eine Punktwolke beschrieben ist, ermöglicht.

Ein wichtiges Werkzeug zur Verarbeitung geometrischer Daten ist Oberflächen-Vereinfachung. Verschiedene Verfahren zur Reduzierung der Komplexität punkt-basierter Flächen werden präsentiert. Diese Methoden werden eingesetzt um Flächen-Hierarchien für effiziente multi-level Berechnungen zu erzeugen.

Eine multi-skalen Flächen-Repräsentation wird eingeführt, die komplexe Editierungen auf verschiedenen Näherungsstufen unterstützt. Dies erlaubt die Implementierung komplexer Filter-Methoden wie Band-pass filter. Multi-Skalen Analyse wird auch eingesetzt zur Konstruktion eines robusten und effizienten Verfahrens zur Merkmals-Extraktion für punkt-basierte Modelle.

Die obengenannten Verfahren werden in ein einheitliches System zur punkt-basierten Modellierung der Gestalt und Erscheinung geometrischer Modelle integriert. Modellierungs-Operationen beinhalten Boolesche Operationen und Frei-Form Deformation, sowie Verfahren zum Bemalen, Texturieren und Filtern von diskreten Oberflächen. Basierend auf einer Methode zur dynamischen Abtastung bietet dieses System eine vollständig und vielseitige Modellierungs-Umgebung zur Bearbeitung von 3D Geometrie.

ACKNOWLEDGEMENTS

My sincere thanks to my advisor Markus Gross. His advice and guidance have been invaluable, his enthusiasm and scientific knowledge both motivating and inspiring. I had a great time at the Computer Graphics Laboratory and I hope that our cooperation will continue in the future.

Many thanks also to my co-advisors Leif Kobbelt and Peter Schröder. It has been a great experience to work with them and share their knowledge in many interesting discussions.

I am very grateful to the people at ETH Zurich for providing such an enjoyable and stimulating working environment. My thanks to Dirk Bauer, Daniel Bielser, Milena Brendle, Manuela Cavegn, Joachim Giesen, Bruno Heidelberger, Andreas Hubeli, Matthias John, Richard Keiser, Nicky Kern, Oliver Knoll, Rolf Koch, Sathya Krishnarmurthy, Edouard Lamboray, Reto Lütolf, Matthias Müller, Martin Näf, Christoph Niederberger, Ronald Peikert, Martin Roth, Bernt Schiele, Christian Sigg, Thomas Sprenger, Oliver Staadt, Matthias Teschner, Julia Vogel, Emo Welzl, Tim Weyrich, Stephan Wuermlin.

A very special thanks to Matthias Zwicker for enduring endless discussions on sampling theory, women, and the meaning of life.

This work has been supported by the joint Berlin/Zurich graduate program Combinatorics, Geometry, and Computation, financed by ETH Zurich and the German Science Foundation (DFG).

CONTENTS

Contents

FIGURES

1

INTRODUCTION

Lately, 3D geometric models have appeared as a new form of digital media. While sound, images and video have been the dominant digital media so far, 3D geometry more and more frequently plays an important role in many application fields, such as e-commerce, entertainment, industrial design, physical simulation, medicine, and education. This creates the need to acquire, store, transmit, process and modify geometric models in an efficient and scalable way. Digital geometry processing is a new research area that is concerned with these issues [106]. Its goal is to define a mathematical foundation for geometric processing applications and to develop tools and algorithms for efficient manipulation of geometric data.

A fundamental question in digital geometry processing is the choice of surface representation: What mathematical description should be chosen to represent the surface of a 3D object on a digital computer? The diversity of the application fields of 3D geometry is reflected in a wide variety of surface representations that have been proposed in the past. For example, industrial design applications for car and airplane construction are mostly based on non-uniform rational B-Splines (NURBS) [32]. Medical applications make frequent use of implicit representations such as level sets or radial basis functions [113], while the game and movie industry has focused on polygonal representations such as triangle meshes [22].

This thesis investigates the fairly recent idea to use point primitives for geometric modeling applications. Initially proposed in [76] as a display primitive, points have been studied mainly in the context of rendering [44, 98, 92, 119, 59], while their use in digital geometry processing is largely unexplored. The goal of this thesis is to close this gap and define algorithms for point-based surface analysis, re-sampling, filtering, and shape and appearance modeling.

The remainder of this chapter is organized as follows: Section 1.1 sets the context of this work by describing the 3D content creation process. Section 1.2 provides some motivation for the approach taken in this thesis, while Section 1.3 briefly discusses related work. The major contributions will be listed in Section 1.4, before Section 1.5 concludes the chapter with an outline of the dissertation.

1.1 3D CONTENT CREATION

There are two common ways to create digital 3D geometry: Either 3D models are designed from scratch, e.g., using some interactive modeling software, or an algorithmic construction process [86]. Alternatively, models can be digitized from a physical object using a 3D scanning device. This latter approach has gained increasing attention in recent years, where significant technological progress has led to the development of 3d scanners that meet the quality and efficiency requirements of advanced 3D content creation applications. Also, with growing complexity of 3D models, ab-initio design becomes more and more time-consuming and thus expensive.

Figure 1.1 illustrates the process of 3D content creation based on 3D acquisition. First some physical object is digitized using a 3D scanning device. The raw data from the scanner is then converted into a surface representation suitable for subsequent processing and modeling. Finally, the model needs to be displayed using some rendering method. The following sections discuss the individual stages of the 3D content creation pipeline in more detail, focusing on aspects relevant for this thesis.

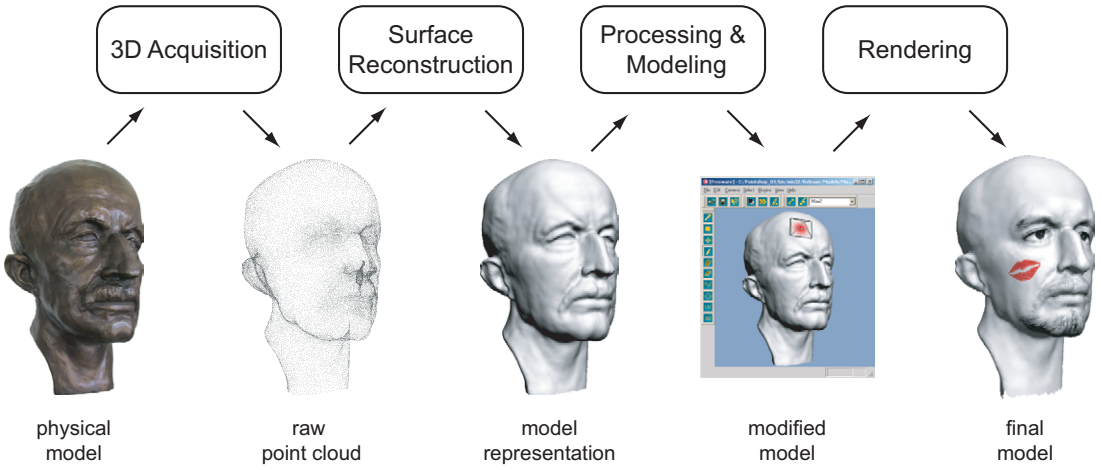


Figure 1.1 3D content creation pipeline.

1.1.1 Acquisition

3D photography is a widely used technique to obtain a digital representation of some physical model. This area has matured significantly in recent years with a great variety of commercial hard- and software solutions available for 3D geometry and appearance acquisition. Popular scanning techniques include laser-range scanners [75], structured light scanners [45], shape from shading [54], and shape from stereo [100]. While the accuracy and spatial resolution of these devices steadily increases, their cost has been reduced substantially. This makes 3D acquisition attractive for high-end applications in industrial design, arts, or archeology, as well as for low cost 3D content creation for entertainment or e-commerce products.

In the most general setting, 3D acquisition devices produce a discrete sample of a physical object, i.e., a collection of point samples. Depending on the acquisition technique, each point sample also carries a number of attributes, such as color, material properties, or measurement confidence.

Since most objects cannot be observed completely from a single viewpoint, several scans can be combined using some registration method [12]. Still, for highly convoluted or concave surfaces, occlusion can cause holes in the reconstruction, leading to an incomplete surface model. Also, scanned data is always corrupted by noise, due to inherent limitations in the physical measurement process. Noise either occurs as outliers, or in the form of small deviations of the sample points from the true surface. Some preprocessing is thus required to obtain a valid surface model from a set of physical measurements. For example, low-pass filtering has been used for noise removal [23], while volumetric diffusion methods have been applied for hole-filling [21].

1.1.2 Surface Reconstruction

In the next stage of the content creation pipeline, the pre-processed set of point samples is converted into a certain surface representation, e.g., a polygonal mesh or a collection of spline patches. This process of surface reconstruction has been a major focus in geometric modeling and computational geometry in recent years [53, 7, 20, 29]. The goal of surface reconstruction is to build a continuous *surface model* from the cloud of point samples, typically assuming that the underlying surface is a smooth 2-manifold. A surface model allows to describe a manifold surface, which is an infinite subset of $\mathbb{R}^3$, with a finite number of parameters. These parameters define the *surface representation* and are computed during the surface reconstruction process. For example, parametric surface representations, such as NURBS or triangle meshes, are defined by a set of control points with an appropriate interpolation or approximation scheme. The control points can either be a subset of the input point cloud or computed using some fitting technique [67].

1.1.3 Processing and Modeling

Given a 3D model in a specific surface representation, the processing and modeling stage is concerned with analyzing and modifying the model surface. Processing operations include surface analysis, filtering, re-sampling, compression, segmentation, shape matching, feature extraction, and others. Modeling comprises shape and appearance editing functionality including boolean operations, free-form deformation, painting and texturing, sculpting, and others.

As compared to traditional multimedia data such as sound, images or video, geometric models require fundamentally different techniques for processing and modeling. The former can be described as a discrete, regular sample of a continuous function defined in 1D (sound), 2D (images), or 3D (video). Digital signal processing provides powerful tools for such data sets, including an extensive sampling theory and sophisticated filtering techniques. These methods are generally not applicable for 3D geometric models, however. The reason for this is two-fold:

- A geometric model is commonly described as a 2-manifold surface embedded in 3-space. In general, there is no canonical functional representation (global parameterization) of such a surface.
- Geometric models are typically sampled irregularly, i.e., samples can be arbitrarily distributed on the surface. Many signal processing methods, including the discrete Fourier transform, rely on a regular grid structure of the underlying data, however.

Non-functional setting and irregular sampling distribution pose great challenges for the design of geometric processing algorithms. Additionally, even though the definition of a manifold surface is based on a set of local parameterizations that observe some continuity constraints [25], global properties of the surface, e.g., surface topology, need to be considered.

Recent efforts have tried to generalize signal processing concepts to discrete surfaces [108, 23, 47, 91]. In particular, significant progress has been made for semi-regular meshes, i.e., piecewise linear surfaces with subdivision connectivity [106]. However, many questions in digital geometry processing are still unanswered. For example, a rigorous mathematical sampling theory, analogous to Fourier theory, has not been defined yet for discrete surfaces.

1.1.4 Rendering

To display a geometric model on a screen or printing device, numerous 3D rendering algorithms have been proposed in the past (see [35] for an overview). While high-quality rendering methods, such as ray-tracing and radiosity, are mainly used for off-line production of animated sequences, forward warping rendering algorithms based on the z-buffer are dominant in interactive applications. The performance of such rendering methods is crucial for achieving sufficiently high frame-rates required for interactive modeling.

Since modern graphics hardware is highly optimized for polygon rendering, many rendering methods first convert a model given in a certain surface representation into a set of triangles. For example, implicit surfaces are often rendered as a triangle mesh that is extracted using the marching cubes algorithm [83]. Similarly, higher-order spline surfaces are often approximated by a polygonal mesh for rendering.

A recent trend in computer graphics is point-based rendering. Pioneered by Levoy and Whitted in [76], various researchers have presented new rendering methods using point primitives [44, 98, 92, 119, 59]. These methods proved particularly successful for rendering densely sampled shapes stemming from 3D acquisition (see also [75]).

1.2 MOTIVATION

This thesis is concerned with the processing and modeling stage of the 3D content creation pipeline. In particular, it focuses on interactive methods for shape and appearance modeling of 3D geometry. A fundamental question in the design of such algorithms is the choice of the underlying surface representation.

On the most abstract level, these representations fall into two major categories: *Implicit* and *parametric* representations. Both categories have their specific advantages and drawbacks. Implicit representations [113] such as level sets [84] and radial basis functions [16] have a particularly simple algebraic structure. Surfaces are defined as zero-sets of 3D scalar fields, which are generated by a weighted superposition of basis functions. In this setting, surfaces with highly complex topology can be represented easily and the global consistency of the surface is guaranteed by construction. Extreme geometric deformations and even topology changes can be achieved by simply modifying the weight coefficients of the respective basis functions. However, efficient rendering and the precise modeling of sharp features is usually a difficult task, since

individual points on implicitly represented surfaces can only be accessed by some ray-casting technique.

Parametric representations like splines [32], subdivision surfaces [115], or triangle meshes are based on a mapping from a (piecewise) planar domain into 3-dimensional space. Here the algebraic structure is usually more complicated and tightly correlated with the complexity of the surface. Since point samples on the surface can be obtained by evaluating the parametric surface function, parametric surfaces can be rendered quite efficiently. Moreover, sharp creases along a curve on the surface can be modeled by adjusting the surface function in parameter space. On the other hand, extreme deformations and topology changes usually require changes to the domain in order to avoid strong distortions and inconsistencies. The shape of a triangle mesh, for example, can be modified to a certain extend by only changing the vertex positions, while keeping the connectivity fixed. However, if the local distortion becomes too strong or if the surface topology is to be changed, mesh connectivity has to be updated while maintaining strict consistency conditions to preserve a manifold surface structure. In practice, the necessity to re-establish the global consistency of the surface function makes this kind of operations on parametric representations rather inefficient.

In this thesis the use of a hybrid geometry representation based on point primitives is proposed that is designed to combine the advantages of implicit and parametric surface models. The geometric shape of a 3D model is represented by an unstructured point cloud. Since point clouds have minimal consistency constraints, re-structuring the surface under deformation is simple and efficient. The point cloud is not treated as a fixed, static representation of the underlying manifold surface, but rather as a dynamic set of temporary samples that evolves over time as the surface deforms. As a second component, an implicit surface model is added that provides access to important surface attributes such as the signed distance function. This supports advanced modeling functionality such as boolean operations or collision detection.

The choice of surface representation is also guided by a trade-off between the number of individual primitives and their descriptive power or approximation order. For example, NURBS can represent complex shapes with relatively few primitives, since each individual spline patch has a high approximation order. On the other hand, combining different patches to a consistent surface model can be difficult, since strict global continuity constraints need to be observed. Polygonal meshes are easier to handle as they are defined by a set of vertices with a consistent adjacency graph, but require a higher number of primitives to achieve the same geometric accuracy. Even more primitives are necessary for point-based representations, not only to provide a certain geometric accuracy, but also to ensure topologically exact reconstruction [8]. On the other hand, point-based surface representations do not require a global combinatorial structure. Even though more primitives need to considered, the hope is that the performance of geometric processing algorithms can be improved, since less computational effort has to be spent per primitive. Ideally, this should also lead to concise algorithms and less complex implementations.

As discussed above, 3D acquisition has become increasingly popular for creating 3D surface models. With increasing resolution of the scanning devices, geometric data sets are becoming more and more complex, lately reaching billions of sample points for a single model [75]. Since scanners typically produce a collection of point samples, 3D reconstruction methods are used to convert the point cloud into a certain surface representation (see Section 1.1.2). Unfortunately, most of these methods do not scale

well with model size, e.g., many techniques based on the Delaunay triangulation have a worst-case complexity of $O(n^2)$, where n is the number of sample points. Being able to directly process point cloud representations would avoid the need for surface reconstruction entirely. Also, point-based rendering methods have proven very successful for displaying complex surfaces. Thus the entire 3D content creation pipeline shown in Figure 1.1 can be implemented with points as a unified acquisition, processing and rendering primitive.

1.3 RELATED WORK

This section briefly reviews some related work that is concerned with surface modeling using point primitives. A more detailed discussion of specific papers will be given in the subsequent chapters.

Levoy and Whitted [76] proposed points as a universal meta-primitive for geometric modeling and rendering applications for 3D geometry. They define a point-based rendering pipeline and discuss the conversion process of geometric models into a set of point samples.

Szeliski and Tonnesen introduced oriented particles for surface modeling [107, 110]. In this approach, individual point samples are coupled through inter-particle potentials, which define their path of motion during an iterative simulation. Oriented particles have been implemented in a blending tool described in Section 6.1.4. The dynamic sampling strategy of Witkin and Heckbert [114] has been incorporated to handle non-uniform sampling distributions. Experiments with this method showed that, while suitable for surface smoothing and filling of small holes, full-scale particle simulation is computationally too expensive for interactive modeling of complex point-sampled surfaces.

Linsen presented a modeling and rendering framework for point-sampled surfaces using the concept of a *fan cloud* [80, 81, 82]. The idea is to represent a local neighborhood of point samples by a triangle fan that is computed using a specific neighborhood relation. Thus the method builds a triangle soup of one-ring neighborhoods that can be processed similarly to a triangle mesh. In particular, all operations that are not dependent on a consistent connectivity graph, e.g., rendering or surface fairing, can be performed with such a representation. A fan cloud does not provide a manifold surface model, however. Re-sampling operations in a dynamic setting are thus difficult to perform.

Central to the algorithms presented in this thesis is the moving least squares (MLS) approximation [74]. A number of researchers have used the MLS surface model to implement geometry processing methods for point clouds. Alexa et al. [4, 5] introduced algorithms for up- and down-sampling of point set surfaces and presented a high quality point rendering system. Fleishman et al. extended this work towards an efficient scheme for progressive encoding and compression [33]. A high-quality rendering method has been presented in [2] that implements ray-tracing on MLS surfaces.

1.4 CONTRIBUTIONS

The goal of this thesis is to investigate point primitives as a surface representation in the context of digital geometry processing. The focus is on new methods for shape and appearance modeling for surfaces represented by point samples.

The main contributions of this thesis are:

- A collection of operators for local surface analysis and classification.
- A set of point-based surface simplification algorithms.
- A multi-scale point-based surface representation.
- A feature extraction pipeline for point clouds.
- A framework and algorithms for point-based shape and appearance modeling.

1.5 OUTLINE

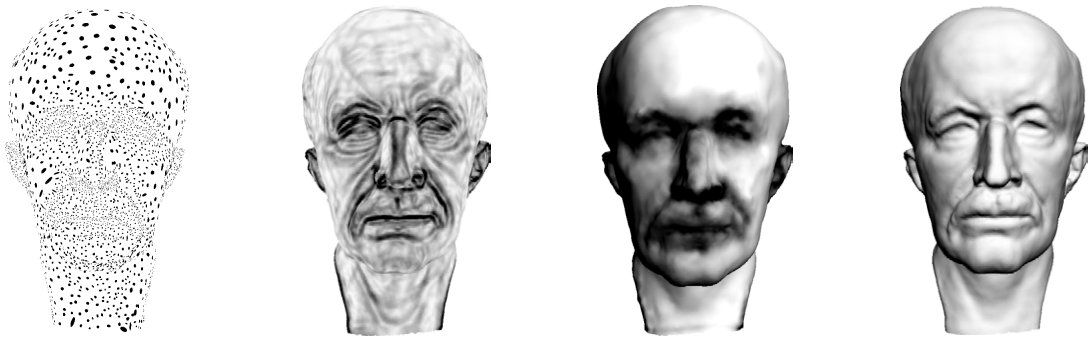
The thesis is organized as follows:

- **Chapter 2** presents various techniques for surface analysis based on local neighborhoods, including tangent plane and surface variation estimation. It also describes and analyses the moving least squares (MLS) surface model used to define a continuous surface from a set of point samples.
- **Chapter 3** introduces a number of surface simplification techniques for reducing the complexity of point-sampled surfaces. A comparative analysis of the different methods is given that considers aspects such as approximation error, resulting sampling distribution, and time and memory efficiency.
- **Chapter 4** extends the single-scale surface representation defined in Chapter 2 to a multi-scale representation that approximates a given model at different levels of geometric detail. For this purpose, low-pass filtering methods for surface smoothing and a decomposition operator for detail encoding are presented. The chapter also defines various spectral filters based on the multi-scale decomposition.
- **Chapter 5** presents a feature extraction pipeline for point-sampled surfaces. A multi-scale classification operator is used to determine a set of feature nodes that are connected using a minimum spanning tree and modeled as active contour models for subsequent smoothing. A non-photorealistic point renderer is introduced to create artistic line-drawings using the extracted feature curves.
- **Chapter 6** introduces boolean operations and free-form deformation for shape modeling of point-sampled surfaces. The representation, sampling, and rendering of sharp feature curves is discussed, as well as a dynamic sampling method that supports extreme geometric deformations.
- **Chapter 7** presents techniques for point-based appearance modeling. Interactive appearance editing operations are based on a new algorithm for computing a minimum distortion parameterization, and a sampling method using parameterized scattered data approximation. Editing operations include painting and texturing, sculpting, and filtering.

- **Chapter 8** shows some examples of point-based shape and appearance modeling operations. It introduces the software platform Pointshop3D, which serves as an implementation test-bed for the algorithms developed in this thesis.
- **Chapter 9** summarizes the thesis and concludes with some directions for future research.

2

POINT-SAMPLED SURFACES



The main objective of this thesis is the processing of manifold surfaces that are represented by finite sets of point samples. The goal is to perform operations directly on the point cloud and only compute (local) surface approximations when they are required. This provides more flexibility when designing new processing algorithms that do not depend on a global combinatorial structure (such as a consistent connectivity graph) of the underlying surface representation.

The input data is given as a point cloud $P = \{\mathbf{p}_i\}_{i \in [1, n]}$, with $\mathbf{p}_i \in \mathbb{R}^3$. The sample points can be produced by some 3D scanning device, e.g., a laser-range scanner, or a sampling process that creates samples from a particular surface representation, e.g., an implicit surface. An arbitrary number of attributes such as color or material properties can be assigned to each sample point, but in general no additional information regarding the definition of the surface is given, such as normal vectors or curvature estimates.

This chapter presents the fundamental techniques that are used to define more sophisticated processing algorithms in the subsequent parts of this thesis. In particular, efficient methods for estimating local surface properties, such as normal vector and curvature, are described. These computations are based on neighborhoods of point

samples, which are defined as subsets of the point cloud P that satisfy some local neighborhood relation. Also, the moving least squares (MLS) surface model is discussed, which defines a smooth manifold surface from the point cloud P and allows the evaluation of the signed distance function.

2.1 LOCAL SURFACE ANALYSIS

The concept of a 2-manifold surface embedded in 3-space is crucial for the algorithms presented in this thesis. The following definition of a manifold surface is taken from [25]:

Definition 2.1: A subset $S \subset \mathbb{R}^3$ is a *2-manifold surface*, if for each $\mathbf{x} \in S$, there exists a neighborhood V in $\mathbb{R}^3$ and a map $X: U \to V \cap S$ of an open set $U \subset \mathbb{R}^2$ onto $V \cap S \subset \mathbb{R}^3$ such that

- X is differentiable, i.e., if $X(u, v) = (x(u, v), y(u, v), z(u, v))$ for $(u, v) \in U$ then the functions x, y and z have continuous partial derivatives of all orders in U,
- X is a homeomorphism, i.e., the inverse $X^{-1}: V \cap S \to U$ exists and is continuous,
- for each $\mathbf{u} \in U$, the differential $dX_u \mathbb{R}^2 \to \mathbb{R}^3$ is one-to-one.

Note that the definition of a surface is based on local (possibly infinitesimal) neighborhoods. Intrinsic properties of the surface, such as tangent plane or Gaussian curvature, are defined with respect to such neighborhoods (see [25] for details).

2.1.1 Local Neighborhoods

In the discrete setting a local neighborhood can be defined through spatial relations of the sample points. Given a point $\mathbf{p} \in P$, a local neighborhood is defined as an index set N_p such that each $\mathbf{p}_i, i \in N_p$ satisfies a certain neighborhood condition. This condition should be set in such a way that the points of N_p adequately represent a small, local surface patch around the point $\mathbf{p}$ (see also Section 2.2). For the computations described below it is important that local neighborhoods only depend on the geometric locations of the point samples in space, not on some additional combinatorial structure associated with the point cloud. In particular, the neighborhood of a point $\mathbf{p}$ should have no dependence on the neighborhoods of its neighbors $\mathbf{p}_i, i \in N_p$.

K-nearest Neighbors

The definition of the k-nearest neighbors N_p^k is based on an ordering of all points in P according to Euclidean distance to the point $\mathbf{p}$. Let Π be a permutation such that $\|\mathbf{p}_{\Pi(1)} - \mathbf{p}\| > 0$ and $\|\mathbf{p}_{\Pi(i)} - \mathbf{p}\| \le \|\mathbf{p}_{\Pi(i+1)} - \mathbf{p}\|, i \in [1, n-1]$.

Then the index set N_p^k of the k-nearest neighbors of the sample $\mathbf{p}$ is given as

$$N_p^k = \{\Pi(1), \Pi(2), \ldots, \Pi(k)\}. \tag{2.1}$$

The set N_p^k defines a sphere s_p^k centered at $\mathbf{p}$ with radius $r_p^k = \|\mathbf{p}_{\Pi(k)} - \mathbf{p}\|$, such that $\mathbf{p}_i$ is inside s_p^k if and only if $i \in N_p^k$.

The following two neighborhood relations are based on the projection of the k-nearest neighbors onto the tangent plane T_p at $\mathbf{p}$ (see Section 2.1.3). Given N_p^k, let $\mathbf{q}_i$ be the projection of $\mathbf{p}_i \in N_p^k$ onto T_p.

BSP Neighbors

Let B_i be the sub-space defined by

$$\{\mathbf{x} \in B_i \,|\, (\mathbf{x} - \mathbf{q}_i) \cdot (\mathbf{p} - \mathbf{q}_i) \geq 0\}. \tag{2.2}$$

Then the BSP (binary space partition) neighbors of $\mathbf{p}$ are defined by the index set $N_p^B \subseteq N_p^k$ such that $i \in N_p^B$, if $i \in N_p^k$ and $\mathbf{q}_i \in \bigcap_{j \in N_p^k} B_j$.

Voronoi Neighbors

Let V be the Voronoi diagram (see [94]) of the projected points $\mathbf{q}_i, i \in N_p^k$. The Voronoi cell V_i of $\mathbf{q}_i$ is defined as

$$V_i = \{\mathbf{x} \in T_p \,|\, \|\mathbf{x} - \mathbf{q}_i\| \leq \|\mathbf{x} - \mathbf{q}_j\| \forall j \in N_p^k, j \neq i\} \tag{2.3}$$

Let V_p be the Voronoi cell that contains a point $\mathbf{p}$. The Voronoi neighbors of $\mathbf{p}$ are defined by the index set $N_p^V \subseteq N_p^k$ such that $i \in N_p^V$, if $i \in N_p^k$ and V_i is adjacent to V_p, i.e., $V_i \cap V_p \neq \emptyset$.

Figure 2.1 illustrates the three different neighborhood relations in 2D. BSP-neighbors and Voronoi neighbors are useful to select a subset of the k-nearest neighbors that still provide an adequate sampling of a local surface patch. Processing methods that critically depend on the number of points in a neighborhood can benefit from this reduction, in particular if a series of computations is performed on a fixed neighborhood. In such cases, the additional cost of evaluating the more sophisticated neighborhood relation of BSP- and Voronoi-neighbors is quickly amortized.

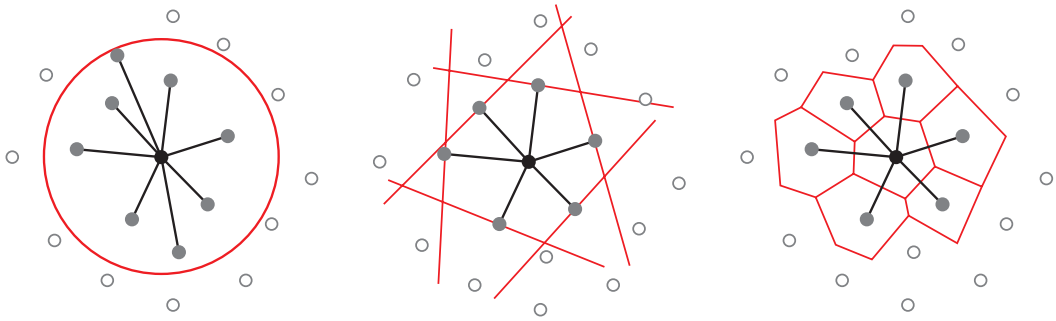


Figure 2.1 Local neighborhoods: (a) k-nearest neighbors, (b) BSP-neighbors, (c) Voronoi neighbors.

2.1.2 Local Sampling Density

An important measure for analyzing point-sampled surfaces is the local density of the point samples, i.e., the number of samples per unit area. The local sampling density ρ_i at a sample point $\mathbf{p}_i \in P$ can be estimated by computing the sphere with minimum

radius r_i centered at $\mathbf{p}_i$ that contains the k-nearest neighbors to $\mathbf{p}_i$. By approximating the intersection of the underlying surface and this sphere with a disc, ρ_i can be defined as

$$\rho_i = \frac{k}{\pi r_i^2}. \tag{2.4}$$

This discrete estimation of local sampling density can be extended to a continuous function $\rho : \mathbb{R}^3 \rightarrow \mathbb{R}_+$ using scattered data approximation as

$$\rho(\mathbf{x}) = \frac{1}{\Omega} \sum_{i=1}^{n} \omega_i(\mathbf{x}) \rho_i, \qquad \Omega = \sum_{i=1}^{n} \omega_i(\mathbf{x}), \tag{2.5}$$

where $\mathbf{x} \in \mathbb{R}^3$ and the weights ω_i can be computed using radial basis functions, i.e., $\omega_i(\mathbf{x}) = B(\|\mathbf{x} - \mathbf{p}_i\|)$. A continuous density function $\rho(\mathbf{x})$ is useful because it provides for each point $\mathbf{x}$ on the surface S an estimate of the sampling density of a small patch around $\mathbf{x}$. A suitable choice for the basis functions B are compactly supported piecewise polynomials. For example, if b is a Hermite polynomial of degree m satisfying

$$\begin{aligned} b(0) &= 1 \\ b(x) &= 0 \\ b'(0) &= b'(1) = 0 \\ &\cdots \\ b^{(m/2)}(0) &= b^{(m/2)}(1) = 0 \end{aligned} \tag{2.6}$$

then Equation 2.5 can be simplified to

$$\rho(\mathbf{x}) = \frac{1}{\Omega} \sum_{j \in N_x} \omega_j \rho_j, \qquad \Omega = \sum_{j \in N_x} \omega_j, \tag{2.7}$$

where N_x defines the k-closest points in P from $\mathbf{x}$ and

$$\omega_i = b\left(\frac{\|\mathbf{x} - \mathbf{p}_i\|}{r}\right), \tag{2.8}$$

with r being the radius of the enclosing sphere s_x. By construction, the function $\rho(\mathbf{x})$ of Equation 2.7 will be $C^{m/2}$ continuous. For example, the blending function

$$b(x) = (x^2 - 1)^2 \tag{2.9}$$

yields a C^2 continuous local sampling density function. Figure 2.12 (b) illustrates such a density map for the irregularly sampled point cloud of Figure 2.12 (a), where $k = 15$ in Equation 2.4.

Local Sample Spacing

From the definition of the sampling density function, an estimate for the local sample spacing can be derived as

$$\eta(\mathbf{x}) = \frac{1}{\sqrt{\rho(\mathbf{x})}}. \tag{2.10}$$

η measures the average distance of sample points within the sphere defined by the k-nearest neighbors.

2.1.3 Covariance Analysis

Based on the neighborhood relations introduced above, local surface properties at a point $\mathbf{p} \in P$ can be estimated using a statistical analysis of the neighboring samples. In particular, eigenanalysis of the covariance matrix of the point positions and normals of a local neighborhood yields efficient algorithms for estimating normal vectors and surface and normal variation (to be defined below).

Let $\bar{\mathbf{p}}$ be the centroid of the neighborhood of $\mathbf{p}$, i.e.,

$$\bar{\mathbf{p}} = \frac{1}{|N_p|} \sum_{i \in N_p} \mathbf{p}_i. \tag{2.11}$$

The 3×3 covariance matrix $\mathbf{C}$ for the sample point $\mathbf{p}$ is then given by

$$\mathbf{C} = \begin{bmatrix} \mathbf{p}_{i_1} - \bar{\mathbf{p}} \\ \cdots \\ \mathbf{p}_{i_k} - \bar{\mathbf{p}} \end{bmatrix}^{\mathrm{T}} \cdot \begin{bmatrix} \mathbf{p}_{i_1} - \bar{\mathbf{p}} \\ \cdots \\ \mathbf{p}_{i_k} - \bar{\mathbf{p}} \end{bmatrix}, i_j \in N_p, \tag{2.12}$$

$\mathbf{C}$ describes the statistical properties of the distribution of the sample points in the neighborhood of point $\mathbf{p}$ by accumulating the squared distances of these points from the centroid $\bar{\mathbf{p}}$. Consider the eigenvector problem

$$\mathbf{C} \cdot \mathbf{v}_l = \lambda_l \cdot \mathbf{v}_l, l \in \{0, 1, 2\}. \tag{2.13}$$

Since $\mathbf{C}$ is symmetric and positive semi-definite, all eigenvalues λ_l are real-valued and the eigenvectors $\mathbf{v}_l$ form an orthogonal frame, corresponding to the principal components of the point set defined by N_p [58]. The λ_l measure the *variation* of the $\mathbf{p}_i$, $i \in N_p$, along the direction of the corresponding eigenvectors. The total variation, i.e., the sum of squared distances of the $\mathbf{p}_i$ from the centroid is given by

$$\sum_{i \in N_p} \left| \mathbf{p}_i - \bar{\mathbf{p}} \right|^2 = \lambda_0 + \lambda_1 + \lambda_2. \tag{2.14}$$

Normal Estimation

Assuming $\lambda_0 \le \lambda_1 \le \lambda_2$, it follows that the plane

$$T(\mathbf{x}): (\mathbf{x} - \bar{\mathbf{p}}) \cdot \mathbf{v}_0 = 0 \tag{2.15}$$

through $\bar{\mathbf{p}}$ minimizes the sum of squared distances to the neighbors of $\mathbf{p}$ [58]. Thus $\mathbf{v}_0$ approximates the surface normal $\mathbf{n}_p$ at $\mathbf{p}$, or in other words, $\mathbf{v}_1$ and $\mathbf{v}_2$ span the tangent plane at $\mathbf{p}$ (see Figure 2.2 (a)).

Normal Orientation

A consistent orientation of the normal vectors can be computed using a method based on the Euclidean minimum spanning tree of the point cloud, as described in [53]. The algorithm starts with an extremal point, e.g., the sample with largest z-coordinate, and orients its normal to point away from the centroid of the point cloud. The normal vector of each adjacent point in the minimum spanning tree can then be oriented based on the assumption that the angle of the normal vectors of adjacent points is less than $\pi/2$ (see Figure 2.2 (b)). If the underlying surface is orientable (note that some surface are non-orientable, e.g., the Moebius strip) and the sampling distribution is sufficiently dense, then a consistent orientation of the normals will be obtained after all points of the point cloud have been visited.

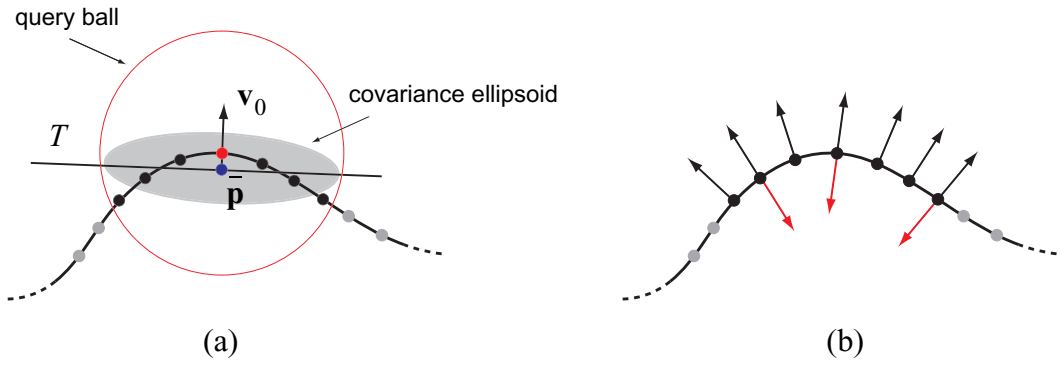


Figure 2.2 Normal estimation (2D for illustration). (a) Computing the tangent plane using covariance analysis, (b) normal orientation using the minimum spanning tree, where the red normals have been flipped, since the angle to the next adjacent oriented normal is larger then $\pi/2$.

Surface Variation

λ_0 quantitatively describes the variation along the surface normal, i.e., estimates how strongly the points deviate from the tangent plane. The *surface variation* at point $\mathbf{p}$ in a neighborhood of size n is defined as

$$\sigma_n(\mathbf{p}) = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2}. \tag{2.16}$$

The maximum surface variation $\sigma_n(\mathbf{p}) = 1/3$ is assumed for completely isotropically distributed points, while the minimum value $\sigma_n(\mathbf{p}) = 0$ indicates that all points lie in a plane. Note also that λ_1 and λ_2 describe the variation of the sampling distribution in the tangent plane and can thus be used to estimate local anisotropy.

Normal Variation

A similar method for local surface analysis considers the covariance matrix of the surface normals, i.e.,

$$\mathbf{C}' = \sum_{i \in N_p} \mathbf{n}_i^T \cdot \mathbf{n}_i. \tag{2.17}$$

Let $\lambda_0' \le \lambda_1' \le \lambda_2'$ be the eigenvalues of $\mathbf{C}'$ with corresponding eigenvectors $\mathbf{v}_0'$, $\mathbf{v}_1'$ and $\mathbf{v}_2'$. As Garland discusses in [38], λ_2' measure the variation of the surface normals in the direction of the mean normal, while λ_1' measures the maximum variation on the Gauss sphere. Thus the *normal variation* can be defined as

$$\sigma_n'(\mathbf{p}) = \lambda_1'. \tag{2.18}$$

Garland also analyses the covariance matrix of the normal vectors in the context of differential geometry. He shows that under mild conditions on the smoothness of the surface, the eigenvectors $\mathbf{v}_0'$, $\mathbf{v}_1'$ and $\mathbf{v}_2'$ converge to the direction of minimum curvature, maximum curvature, and mean normal, respectively, when sampling density goes to infinity.

Figure 2.3 compares surface variation and normal variation for the Max Planck model consisting of 139,486 points. With increasing neighborhood size, a smoothing of the variation estimates can be observed. This effect will be used in Chapter 5 to define a multi-scale feature classification operator.

2.2 SAMPLING REQUIREMENTS

The methods for local surface analysis introduced above are based on the assumption that the k-nearest neighbors of a sample point $\mathbf{p}$ adequately represent a small patch of the underlying surface S around $\mathbf{p}$. In particular, the intersection $S_{p,k}$ of the query ball of the k-nearest neighbors of $\mathbf{p}$ with S should be homeomorphic to a disc. (see Figure 2.4 (a)). Additionally, $S_{p,k}$ should be relatively flat to ensure accurate normal estimation (Figure 2.4 (b)). This means that the query ball must not be too large in regions of high curvature or where two geodesically distant parts of S come close together. Nevertheless, the density of the samples within the query ball needs to be high enough to enable accurate and stable estimation of local surface properties.

Appendix C presents a theoretical analysis of local surface estimation using k-nearest neighbors given a discrete sample P of a smooth, twice-differentiable manifold surface S. It is shown that, if P satisfies certain sampling criteria, there exists a k such that local surface properties, e.g., the tangent space, can be estimated accurately from the set of k-nearest neighbors. In particular, if the sampling density goes to infinity, the estimated normal vector at a point $\mathbf{p} \in P$ (see Equation 2.15) converges to the surface normal of S at $\mathbf{p}$.

Lemma C.4 gives quantitative bounds on the number k of nearest neighbors. Empirical evidence shows, however, that these bounds are far from being optimal. In fact, hardly any of the point-sampled models used in this thesis meets the sampling requirements stated in Lemma C.4. Still, experiments demonstrate that the methods presented in Section 2.1 give satisfactory results for these models. This suggests that there is still significant room for improvement on the bounds of Lemma C.4. Also note that in the proof a number of conservative bounds have already been incorporated.

Since the discrepancy between current theoretical results and practical experience is still substantial, the result of Lemma C.4 is not immediately relevant for practical applications. There is also a more inherent limitation. To determine if a given point cloud P meets the proposed sampling requirement, access to the medial axis of the underlying surface S is required. The medial axis is defined as the closure of the set of points which have more than one closest point on S (see also Definition C.1).

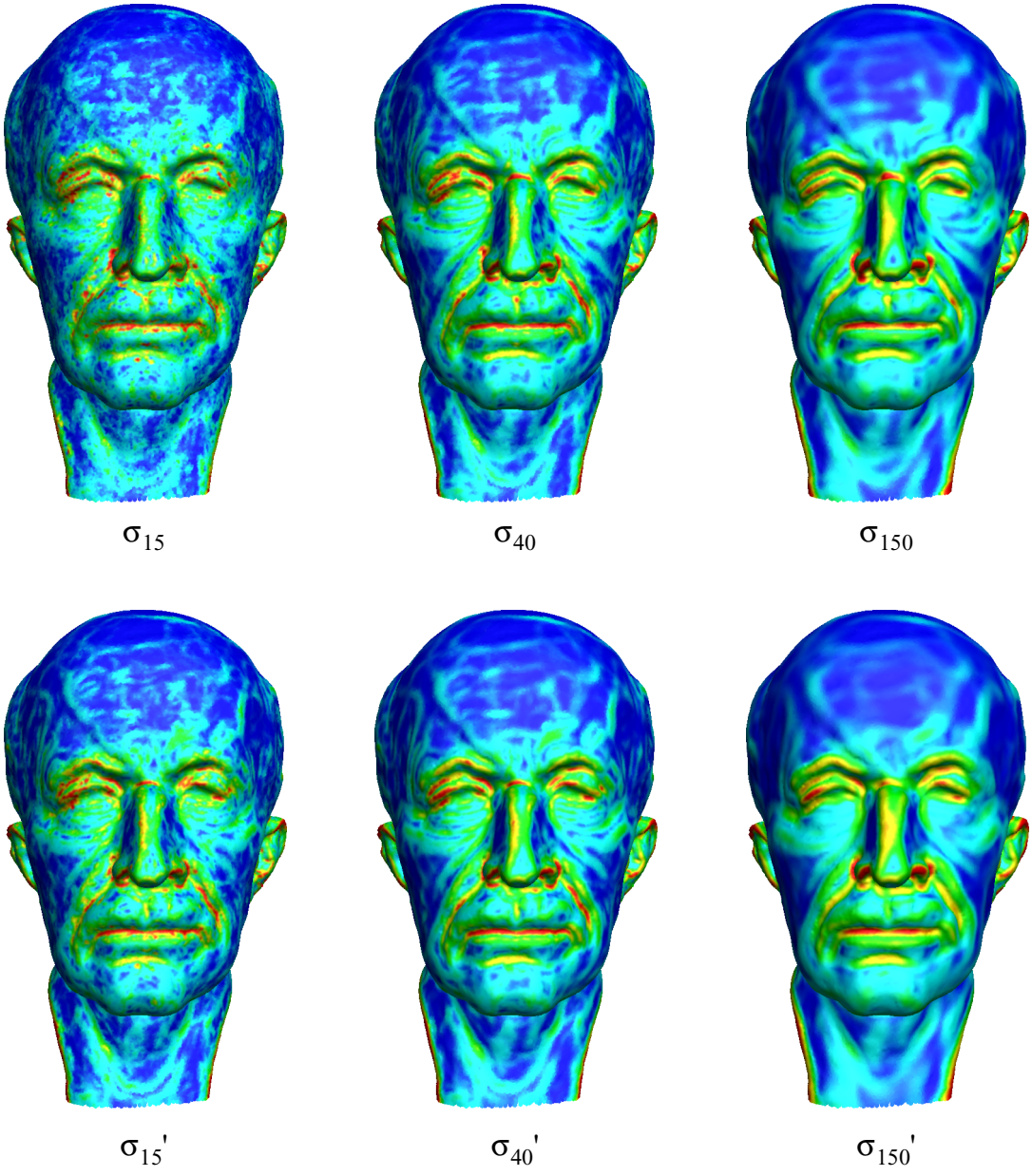


Figure 2.3 Comparison of surface variation (top row) and normal variation (bottom row) for increasing size of the local neighborhoods.

Unfortunately, the medial axis is not known for most point-sampled surfaces, e.g., scanned models. Thus given a point cloud P, it is generally not possible to decide whether P fulfills the sampling condition without additional information on the surface S. Nevertheless, the analysis given in Appendix C provides some theoretical justification on why point-based surface modeling based on k-nearest neighbors is practical. Also, in related fields such as surface reconstruction, a similar analysis on sampling requirements has led to the development of efficient algorithms with provable bounds (e.g., [7, 8]).

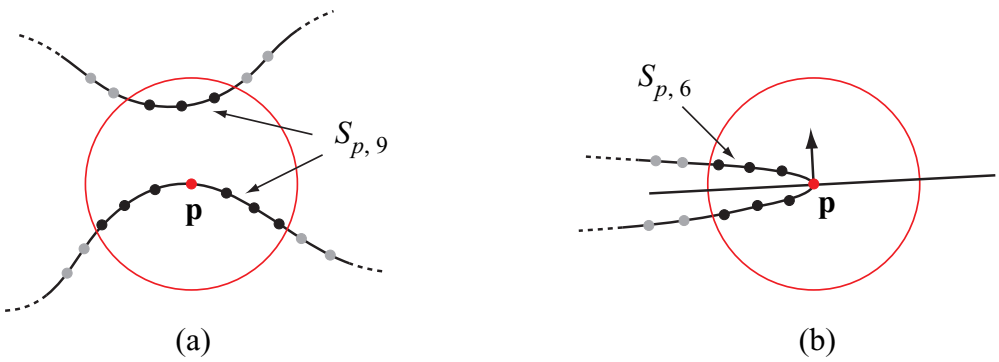


Figure 2.4 Undersampling leads to wrong local surface analysis using k-nearest neighbors: (a) the intersection of the query ball with the surface consists of two disjoint components, (b) wrong tangent plane estimation due to high curvature.

2.3 MOVING LEAST SQUARES SURFACES

Given a point cloud P the goal of a point-based surface model is to define a surface S that approximates or interpolates the sample points $\mathbf{p}_i \in P$. In the context of this thesis, the surface model should satisfy the following requirements:

- *Smoothness*: The surface S should be smooth and differentiable, preferably in a controllable manner. This means that there should be some mechanism to adjust the smoothness depending on the intended application.
- *Locality*: The evaluation of the surface model should be local, i.e., only points within a certain distance should influence the definition of the surface at a particular location. Locality is desirable to enable local modifications without affecting the global shape of the surface. It also increases the efficiency of the computations.
- *Stability*: The evaluation of the surface model should be stable and robust, even for non-uniform sampling distributions. This means that the surface model needs to be adaptive, i.e dependent on the local sampling density.

Methods for interpolating or approximating functions in $\mathbb{R}^d$ from a discrete set of scattered data values have been studied extensively in the past. Examples include reconstruction using finite elements [27], radial basis functions [26], and moving least squares (MLS) approximation [69]. The latter two are advantageous because they are "mesh-less", i.e., do not require a consistent tesselation of the function domain.

Recently, Levin has introduced an extension of the moving least squares approximation to surfaces [74]. This method will be used throughout this thesis for computing local approximations of the surface represented by a point cloud, and to evaluate the corresponding signed distance function. The idea is to locally approximate the surface by polynomials that minimize a weighted least squares error at the data points. Since the method is solely based on Euclidean distance between sample points, no additional combinatorial structure on the point cloud is required.

After an introduction to the MLS approximation in the functional setting, Section 2.3.2 discusses Levin's generalization to surfaces. The original definition of an MLS

surface will then be extended to an adaptive scheme that is more robust for non-uniformly sampled surfaces.

2.3.1 Functional MLS Approximation

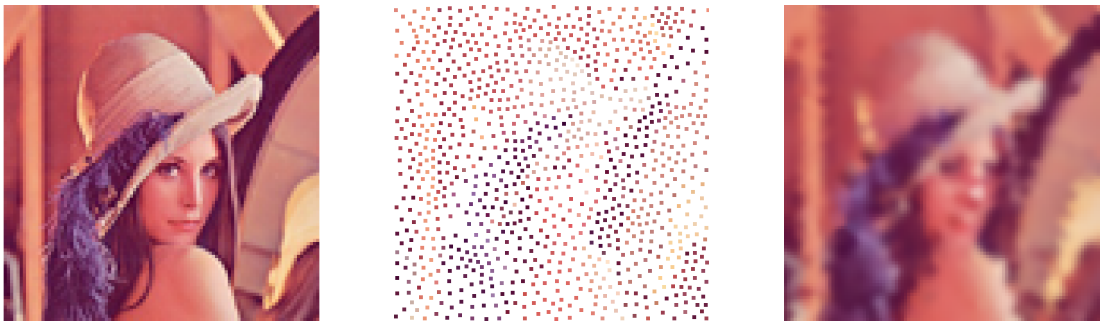
Let $X = \{\mathbf{x}_i, f_i\}$ be a finite set of distinct sample points $\mathbf{x}_i \in \mathbb{R}^d$ and corresponding function values $f_i \in \mathbb{R}$. Let Π_m^d be the space of polynomials of degree m in $\mathbb{R}^d$. For a given point $\mathbf{x} \in \mathbb{R}^d$, the MLS approximation $\tilde{f}(\mathbf{x})$ of degree m is given as the value $\tilde{p}(\mathbf{x})$, where $\tilde{p} \in \Pi_m^d$ minimizes

$$\sum_i (p(\mathbf{x}_i) - f_i)^2 \Phi(\mathbf{x}_i, \mathbf{x}) \tag{2.19}$$

among all $p \in \Pi_m^d$. Φ is a weight function that typically depends only on the Euclidean distance between the two sample points, i.e., can be written as $\Phi(\mathbf{x}_i, \mathbf{x}) = \phi(\|\mathbf{x}_i - \mathbf{x}\|)$, where $\phi: \mathbb{R}_+ \to [0, \infty]$ is a smooth, positive, monotonously decreasing function. Interpolation can be achieved by allowing a singularity at zero, i.e.,

$$\lim_{r \to 0} \phi(r) = \infty. \tag{2.20}$$

The existence and uniqueness of the minimization problem defined by Equation 2.19 can easily be proven. It can also be shown that the smoothness of the approximation depends on the smoothness of the weight function. More precisely, if $\Phi \in C^k$ then $\tilde{f} \in C^k$ as well (see [10] for more details). Figure 2.5 shows an example of an MLS approximation of 2D image data. The regularly sampled image on the left contains 100×100 pixels that have been sub-sampled randomly using injection in the middle image to 1,000 sample points. The right image shows the MLS approximation of this random set, sampled at the resolution of the original image.



original | random sub-sample | MLS reconstruction

Figure 2.5 Functional MLS approximation on 2D image data.

2.3.2 MLS Surface Model

The functional MLS approximation is defined for Euclidean domains and makes use of a global parameterization, which is generally not available for manifold surfaces. Therefore, Levin proposed an implicit MLS scheme to approximate 2D manifold surfaces embedded in 3D using a projection method [74].

Given a point set $P = \{\mathbf{p}_i\}$, the MLS surface $S(P)$ is defined as the stationary set of a projection operator Ψ_P, i.e.,

$$S(P) = \{\mathbf{x} \in \mathbb{R}^3 \,|\, \Psi_P(\mathbf{x}) = \mathbf{x}\}. \tag{2.21}$$

$\Psi_P(\mathbf{x})$ is defined by a two-step procedure:

- Compute a local reference plane

$$H = \{\mathbf{y} \in \mathbb{R}^3 \,|\, \mathbf{y} \cdot \mathbf{n} - D = 0\} \tag{2.22}$$

by minimizing the weighted sum of squared distances

$$\sum_{i \in I} (\mathbf{p}_i \cdot \mathbf{n} - D)^2 \phi(\|\mathbf{p}_i - \mathbf{q}\|), \tag{2.23}$$

where $\mathbf{q}$ is the orthogonal projection of $\mathbf{x}$ onto H. The reference plane defines a local coordinate system with $\mathbf{q}$ at the origin. Let (u_i, v_i, f_i) be the coordinates of the point $\mathbf{p}_i$ in this coordinate system, i.e., (u_i, v_i) are the parameter values in H and f_i is the height of $\mathbf{p}_i$ over H.

- Compute a bivariate polynomial $\tilde{p}(u, v)$ that minimizes

$$\sum_{i \in I} (p(u_i, v_i) - f_i)^2 \phi(\|\mathbf{p}_i - \mathbf{q}\|) \tag{2.24}$$

among all $p \in \Pi_m^2$.

The projection of $\mathbf{x}$ onto $S(P)$ is then given as

$$\Psi_P(\mathbf{x}) = \mathbf{q} + \tilde{p}(0, 0) \cdot \mathbf{n}. \tag{2.25}$$

Figure 2.6 illustrates the MLS projection for a curve example in 2D. The left image shows a single projection of a point $\mathbf{x}$ onto the MLS curve $S(P)$ depicted on the right.

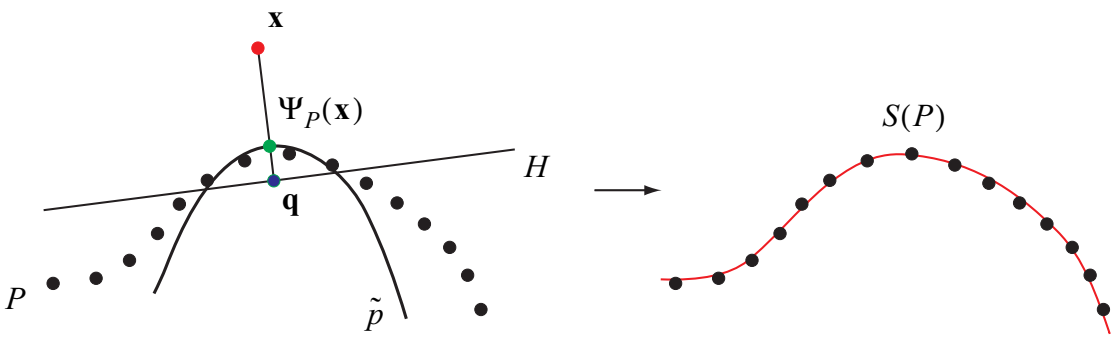


Figure 2.6 2D illustration of the MLS projection.

In [73] Levin analyses the smoothness and convergence rate, which leads him to the conjecture that the smoothness of $S(P)$ directly depends on the smoothness of ϕ, i.e., if $\phi \in C^k$ then $S(P) \in C^k$ (compare with Section 2.3.1). The kernel function ϕ thus controls the shape of the surface $S(P)$. A suitable choice for ϕ is the Gaussian, i.e.,

$$\phi(x) = e^{-\frac{x^2}{h^2}}, \tag{2.26}$$

where h is a global scale factor that determines the kernel width. The Gaussian kernel has been used successfully in different applications (see also [4, 33, 2]) and will be used throughout this thesis. For an analysis of different kernel functions see [10].

(a) h=1 (b) h=2 (c) h=4

Figure 2.7 Smoothing effect of the MLS projection. The Gaussian kernel width has been doubled for each image from left to right.

The scale factor h can be understood as the characteristic scale of $S(P)$, i.e., features that are smaller than h will be smoothed out. In this sense the MLS projection operator implements a low-pass filter, whose filter width can be controlled by the parameter h. Figure 2.7 shows an example an MLS surface computed with different scale factors.

Given the MLS surface representation, local surface properties such as tangent plane or curvature, can be estimated directly from the fitting polynomials. However, in a dynamic setting these computations are often too costly. Thus in time-critical applications, the methods presented in Section 2.1.3 are more appropriate.

2.3.3 Signed Distance Function

An important tool for local surface analysis is the signed distance function $d_+ : \mathbb{R}^3 \rightarrow \mathbb{R}$ that measures for each point $\mathbf{x} \in \mathbb{R}^3$ the signed distance to the surface S. Since the MLS method defines an orthogonal projection when using linear polynomials in Equation 2.24, a distance function can be defined as

$$d(\mathbf{x}) = \left\| \mathbf{x} - \Psi_P(\mathbf{x}) \right\|. \tag{2.27}$$

If S is closed and the normals on S are consistently oriented, e.g., always pointing outward of the surface S (see Section 2.1.3), then the signed distance function can be formulated as

$$d_+(\mathbf{x}) = (\mathbf{x} - \Psi_P(\mathbf{x})) \cdot \mathbf{n}, \tag{2.28}$$

where $\mathbf{n}$ is the surface normal at $\Psi_P(\mathbf{x})$ with $\|\mathbf{n}\| = 1$. Thus the MLS surface S can be defined implicitly as the zero-set of this 3D scalar field, i.e.,

$$S = \{\mathbf{x} \in \mathbb{R}^3 \,|\, d_+(\mathbf{x}) = 0\} = \{\mathbf{x} \in \mathbb{R}^3 \,|\, d(\mathbf{x}) = 0\}. \tag{2.29}$$

The signed distance function induced by the MLS projection operator will be used in subsequent parts of this thesis for error measurement (Section 3.5.1), multi-scale decomposition (Section 4.3.1) and inside/outside classification (Section 6.1.1).

2.3.4 Computing the MLS Projection

The MLS projection method defined above consists of two steps: Computation of the local reference plane H and computation of the least-squares polynomial $\tilde{p}$ with respect to that reference plane. The former requires a non-linear optimization, since the distances used in the kernel function depend on the projection $\mathbf{q}$ of the point of interest $\mathbf{x}$ onto the unknown reference plane. Alexa et al. [4] use Powell iteration to compute D and $\mathbf{n}$ in Equation (2.23).

A different approach is to try and estimate $\mathbf{q}$ directly, which determines $\mathbf{n} = (\mathbf{x} - \mathbf{q})/\|\mathbf{x} - \mathbf{q}\|$ and $D = \mathbf{q} \cdot \mathbf{n}$. This can be achieved using a Newton-type iteration based on a geometric gradient estimation. First an initial estimate $\mathbf{q}_0$ for $\mathbf{q}$ is computed by choosing the point of the point cloud P that lies closest to $\mathbf{x}$. Finding the best fitting plane H_0 to a local neighborhood of samples at $\mathbf{q}_0$ is an easy task since it only requires the solution of a linear system (see Section 2.1.3). The next estimate $\mathbf{q}_1$ for $\mathbf{q}$ is obtained by orthogonal projection of $\mathbf{x}$ onto H_0. This procedure can be iterated such that in each step another estimate $\mathbf{q}_{i+1}$ is computed by projecting $\mathbf{x}$ onto the plane H_i, which is the weighted least-squares plane fitted to a local neighborhood of samples of P around the previous estimate $\mathbf{q}_i$. This iterative procedure corresponds to a Newton-iteration scheme to find the point $\mathbf{q}$ that defines the reference plane with respect to $\mathbf{x}$. To avoid oscillatory behavior of the iterative scheme, an additional damping factor λ is introduced, i.e., instead of fully updating from $\mathbf{q}_i$ to $\mathbf{q}_{i+1}$ the damped update from $\mathbf{q}_i$ to $(1-\lambda)\mathbf{q}_i + \lambda\mathbf{q}_{i+1}$ is used, where $\lambda = 0.5$ typically gives satisfactory results.

Given the reference plane H, all points in P are projected into a local reference frame defined by H. The computation of the polynomial $\tilde{p}$ is now a linear weighted least squares approximation, which can be computed using normal equations. For example, a cubic polynomial leads to a linear system with 10 unknowns, which can be solved using a standard linear solver.

Both computation of the reference plane and the fitting polynomial require order $O(n)$ computations, since they involve the entire point cloud P (see Equations 2.23 and 2.24). However, since the value of the weight function drops quickly with distance, efficient approximation schemes can be applied to significantly reduce the complexity of the computations. For example, Alexa et al. [4] use a multi-pole scheme, where they cluster groups of points that are far away from the point of interest $\mathbf{r}$ into a single point and use this representative in the optimization. A different approach is to only collect points within a sphere s_r of radius r centered at $\mathbf{x}$, such that the weight for each point outside this sphere is so small that the influence in the least-squares optimization is negligible. For example, given an MLS scale factor h, r can be set to $3h$ to yield an MLS weight of less then 0.001 in Equations 2.23 and 2.24 for all points outside of s_r.

2.3.5 Approximation Error and Reconstruction

The MLS surface approximation has the projection property, i.e.,

$$\Psi_P(\Psi_P(\mathbf{x})) = \Psi_P(\mathbf{x}). \tag{2.30}$$

When using Gaussian kernels, the MLS projection is not interpolating, however. This means that, in general, $\Psi_P(\mathbf{p}_i) \neq \mathbf{p}_i$ for $\mathbf{p}_i \in P$. Thus the surface S_{Ψ_P} defined by the point set $\Psi_P(P) = \{\Psi_P(\mathbf{p}_i)\}$ is different than the surface S_P defined by P. To quantify this distance, Alexa et al. considered the approximation error of the MLS projection [5]. Assuming that both surfaces have been approximated with the same scale factor h, they found that

$$\left\| S_{\Psi_P} - S_P \right\| < M_{m+1} h^{m+1}, \tag{2.31}$$

where polynomials of degree m have been used in optimization of Equation 2.24 and M_{m+1} is a constant depending on the $(m+1)$-th derivatives of S_P. They also investigated the error of piecewise polynomial approximations of S_P, see [5] for details.

Another important aspect of the MLS surface model is its stability with respect to sampling density. Figure 2.8 shows an example of a point cloud that is not sampled densely enough for correct reconstruction by the MLS method. The bunny model on the left has been uniformly subsampled (see Chapter 3) from 34,834 to 534 sample points. On the right image, the MLS reconstruction of this subsampled point cloud is shown. Due to the loss of information during the simplification process, this surface exhibits less geometric detail, but still conveys the overall shape of the original model. At the tip of the ears, however, the surface has been contracted to a single sheet (see Figure 2.8 (e)). In fact, an important application of the MLS is thinning of noisy point clouds [71], where this effect is exploited explicitly. In the example of the bunny's ears, the sampling rate is too low in relation to the local feature size (see Section 2.2), so the volumetric extent of the ears is interpreted as deviations from a plane that are considered as noise.

A very fundamental question regarding the MLS surface model is the following: Given a surface S, what is the minimum sampling rate (and how need these samples be distributed on the surface) so that the MLS model provides a topologically correct reconstruction of S? This problem is closely related to the definition of sampling requirements described in Section 2.2. It is beyond the scope of this thesis to address this question in detail. Instead, the following section will present an extension of the MLS scheme that improves the approximation error and numerical stability of the computations.

2.3.6 Adaptive MLS Surfaces

So far the MLS surface approximation has been defined with a fixed Gaussian kernel width h. Finding a suitable h can be difficult for non-uniformly sampled point clouds, however. A large scale factor will cause excessive smoothing in regions of high sampling density. Even worse, if the filter kernel is too small, only very few points will contribute significantly to Equations (2.23) and (2.24) due to the exponential fall-off of the weight function. This can cause instabilities in the optimization because of limited precision of the computations, which lead to wrong surface approximations.

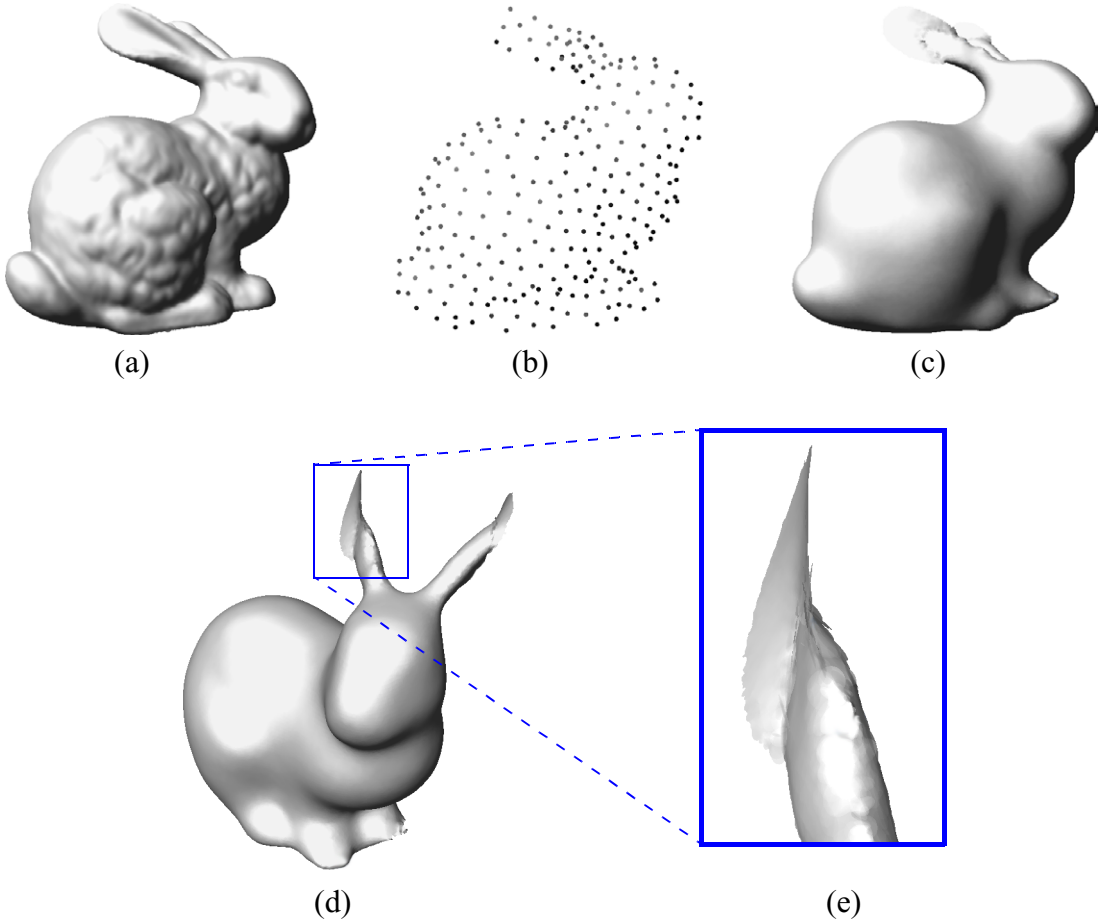


Figure 2.8 Wrong MLS surface reconstruction due to undersampling. (a) original model, (b) subsampled point cloud, (c) and (d) MLS reconstruction of (b), (e) zoom of the ear.

Figure 2.9 shows an example of a surface that cannot be approximated adequately using a global scale factor. The surface is defined using a concentric sine wave whose wavelength and amplitude gradually increases towards the rim of the surface (see Figure 2.9 (c)). Similarly, the sampling density decreases towards the boundary as illustrated in Figures 2.9 (b) and (d). Thus the surface detail is coupled with the sampling density, i.e., in regions of high sampling density the surface exhibits high-frequency detail, whereas low-frequency detail is present where the surface is sampled less densely.

Figure 2.10 shows reconstructions of the central section of this surface using the regular sampling grid shown in Figure 2.10 (b). The Gaussian used in Figure 2.10 (c) causes substantial smoothing and leads to a significant loss of geometric detail in the central area. In Figures 2.10 (d) and (e) the kernel width has been successively halved, which improves the approximation in the central region but leads to increasing instability towards the boundary.

To cope with this problem, the MLS approximation needs to adapt to the local sampling density. In regions of high sampling density a small Gaussian kernel should be applied to accurately reconstruct potential high geometric detail. If the sampling

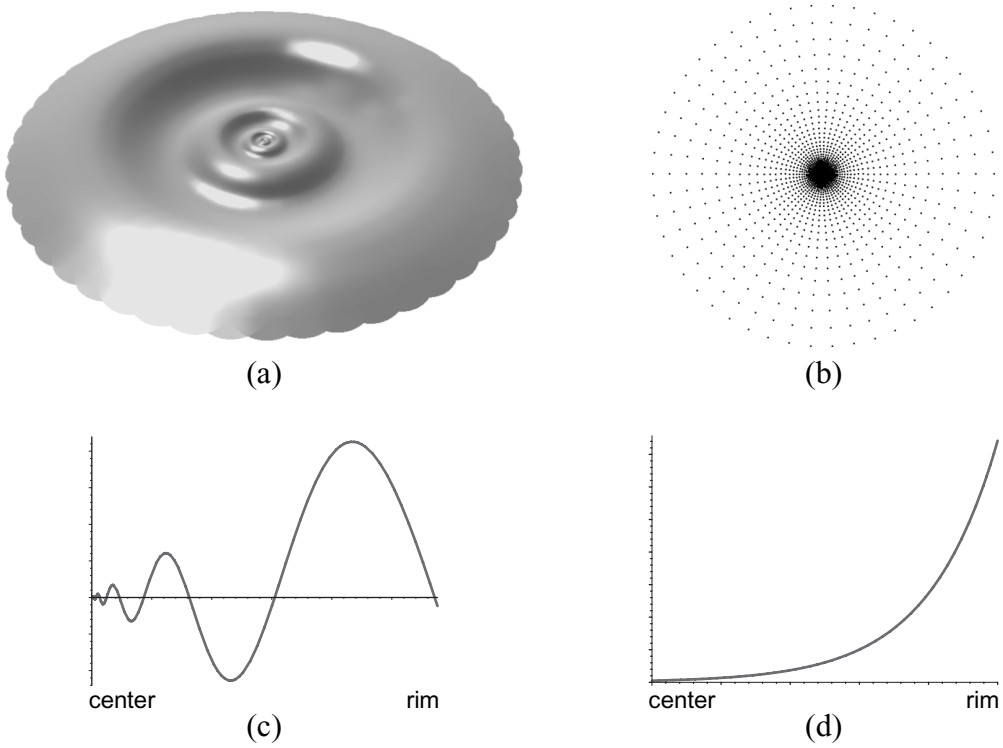


Figure 2.9 A surface that cannot be reconstructed using a fixed kernel width. (a) surface, (b) sampling pattern, (c) vertical cross-section of surface, (d), sample spacing

density is low, the kernel width needs to be bigger to improve the stability of the approximation.

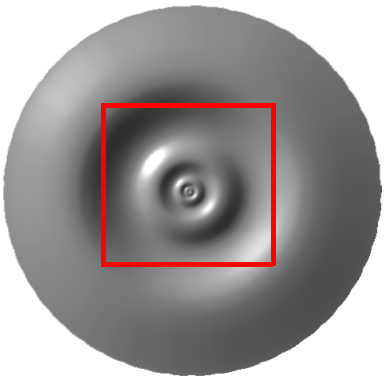
Given the continuous local sampling density estimate ρ of Section 2.1.2, the MLS approximation can be extended in the following way: Instead of using a fixed scale factor h for all approximation, a dynamically varying scale factor can be defined as

$$h_{\mathbf{x}} = \frac{h}{\rho(\mathbf{x})}, \tag{2.32}$$

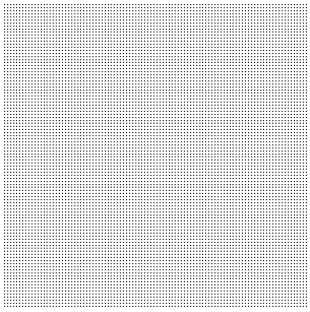
where $\mathbf{x}$ is the point that is to be projected onto the MLS surface. The idea of adaptive scale factors has been introduced by Baule for MLS approximations in the functional setting [10]. The focus here was on improving the accuracy and efficiency of the computations, while numerical stability was not considered. Therefore, Baule used an MLS approximation with fixed kernel ϕ_1 to estimate the sampling density and an subsequent adaptive MLS step with varying kernel ϕ_2 to compute the final function approximation. He proves that the smoothness of the functional approximation F depends on the smoothness of the two functions ϕ_1 and ϕ_2. More precisely, if $\phi_1 \in C^i$ and $\phi_2 \in C^j$ then $F \in C^{min(i,j)}$. This result, in connection with Levin's analysis of the MLS projection operator Ψ leads to the following conjecture:

Let $\rho \in C^m$ be a density estimate for a point cloud P and let

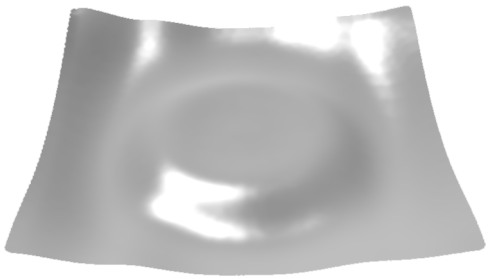
$$\phi(\mathbf{p}_i, \mathbf{x}) = e^{-\frac{x^2 \cdot \rho(\mathbf{x})^2}{h^2}} \tag{2.33}$$



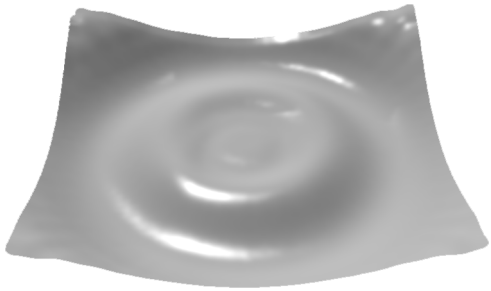
(a) section considered in the approximations of (c)-(e)



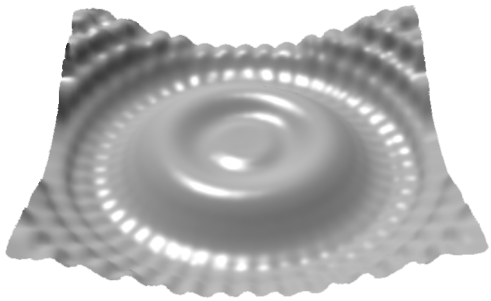
(b) sampling pattern of (c)-(e)



(c) h=1

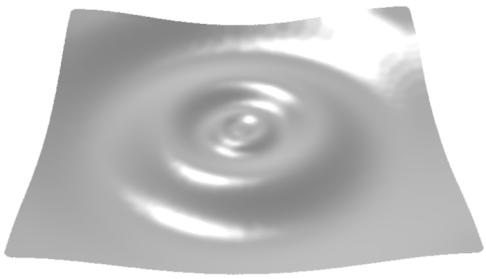


(d) h=1/2

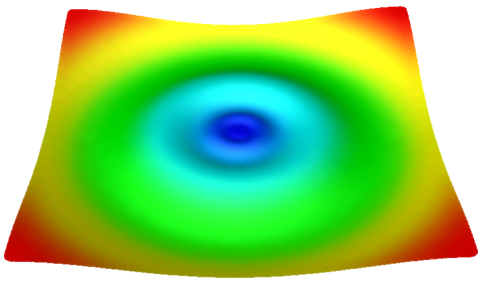


(e) h=1/4

Figure 2.10 Regular re-sampling of a non-uniformly sampled surface using fixed Gaussian kernels.



(a)



(b)

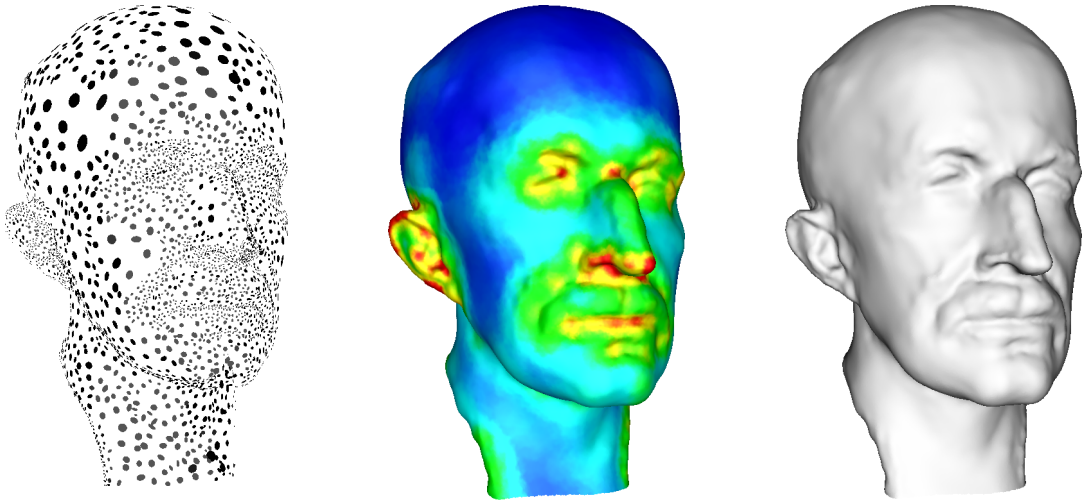
Figure 2.11 Adaptive MLS reconstruction of Figure 2.10: (a) reconstructed surface using the sampling pattern of Figure 2.10 (b), (b) sample spacing, where blue corresponds to low values, while red indicates high values.

be the MLS kernel function and S the MLS surface defined by P. Then $S \in C^m$.

Once a proof of Levin's conjecture [74] has been found (if it exists), it should be straightforward to extend this proof using Baule's approach for functional MLS approximations [10].

Figure 2.11 shows an example of an adaptive MLS surface using the same input point cloud and sampling pattern as in Figure 2.10. The sample spacing has been computed using a fourth-order B-Spline blending function as shown in Figure 2.11 (b). Observe that the high-frequency detail is faithfully recovered within the limits of the resolution of the sampling grid and that no instabilities occur at the surface boundary.

Figure 2.12 shows an example of an MLS reconstruction of a non-uniformly sampled point cloud that has been created by curvature adaptive simplification of a uniformly sampled surface (see Chapter 3). Throughout the rest of this thesis, the adaptive MLS scheme will be used for locally approximating a surface from a set of point samples.

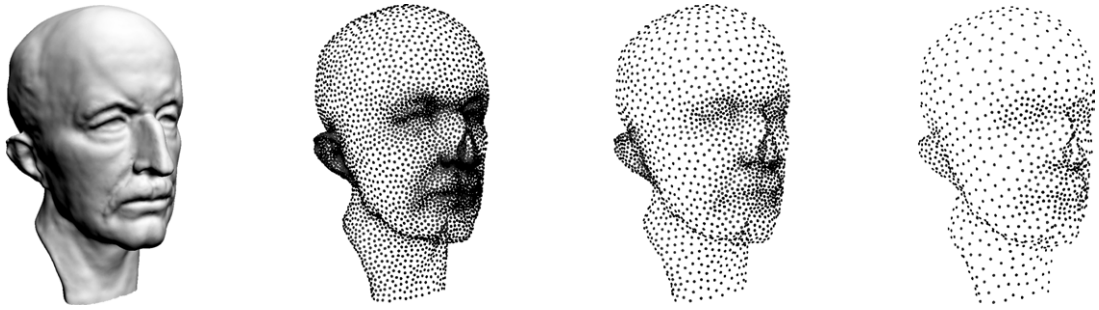


(a) (b) (c)

Figure 2.12 Adaptive MLS reconstruction of the Max Planck bust. (a) input point cloud with 5,413 points, (b) continuous sampling density map, (c) reconstructed MLS surface.

3

SURFACE SIMPLIFICATION



As indicated in the introduction, point-sampled surfaces often describe complex geometric objects using millions or even billions of sample points (see for example [75]). Reducing the complexity of such data sets is one of the key processing techniques for the design of scalable modeling and visualization algorithms. Surface simplification provides a means to generate the required approximations of a given surface that use fewer sample points than the original point model. These approximations should of course resemble the original surface as closely as possible.

Formally, the goal of point-based surface simplification can be stated as follows: Let S be a manifold surface defined by a point cloud P. Given a target sampling rate $n < |P|$, find a point cloud P' with $|P'| = n$ such that the distance $\varepsilon = d(S, S')$ of the corresponding surface S' to the original surface S is minimal. Alternatively, a target distance ε can be specified and the goal is to find the point cloud P' such that $d(S, S') \leq \varepsilon$ and $|P'|$ is minimal.

These objectives require the definition of a metric d that measures the geometric distance between the original and the simplified surface. As will be described in Section 3.5.1 a discrete surface distance metric can be defined using the MLS projection operator introduced in Section 2.3.2.

In practice, finding the global optimum to the above problems is intractable [38]. Most existing surface simplification techniques therefore use different heuristics based on local error measures. In this thesis, three different approaches have been implemented and analyzed [90]:

- *Clustering methods* split the point cloud into a number of disjoint subsets, each of which is replaced by one representative sample (see Section 3.2).
- *Iterative simplification* successively contracts point pairs in a point cloud according to a quadric error metric (Section 3.3).
- *Particle simulation* computes new sampling positions by moving particles on the point-sampled surface according to inter-particle repelling forces (Section 3.4).

These methods are extensions and generalizations of mesh simplification algorithms to point clouds, targeted towards densely-sampled organic shapes stemming from 3D acquisition, iso-surface extraction or sampling of implicit functions. They are less suited for surfaces that have been carefully designed in a particular surface representation, such as low-resolution polygonal CAD data.

Furthermore, the goal was to design algorithms that are general in the sense that they do not require any knowledge of the specific source of the data. For certain applications this additional knowledge could be exploited to design more effective simplification algorithms, but this would also limit the applicability of these methods.

The algorithms presented in this chapter differ in a number of aspects such as quality of the generated surfaces, computational efficiency and memory overhead. These features are discussed in a comparative analysis in Section 3.5. The purpose of this analysis is to give potential users of point-based surface simplification suitable guidance for choosing the right method for their specific application. Real-time applications, for instance, will put particular emphasis on efficiency and low memory footprint. Methods for creating surface hierarchies favor specific sampling patterns (e.g., [112]), while visualization applications require accurate preservation of appearance attributes, such as color or material properties.

3.1 RELATED WORK

Earlier methods for simplification of point-sampled models have been introduced by Alexa et al. [4] and Linsen [81]. These algorithms create a simplified point cloud that is a true subset of the original point set by ordering iterative point removal operations according to a surface error metric. While both papers report good results for reducing redundancy in point sets, pure subsampling unnecessarily restricts potential sampling positions, which can lead to aliasing artefacts and uneven sampling distributions. To alleviate these problems, the algorithms described in this chapter re-sample the input surface and implicitly apply a low-pass filter (e.g., clustering methods perform a local averaging step to compute the cluster's centroid).

In [91], Pauly and Gross introduced a re-sampling strategy based on Fourier theory. They split the model surface into a set of patches that are re-sampled individually using a spectral decomposition. This method directly applies signal processing theory to point-sampled geometry, yielding a fast and versatile point cloud decimation method. Potential problems arise due to the dependency on the specific patch layout and difficulties in controlling the target model size by specifying spectral error bounds.

3.2 CLUSTERING

Clustering methods have been used in many computer graphics applications to reduce the complexity of 3D objects. Rossignac and Borrel, for example, used vertex clustering to obtain multi-resolution approximations of complex polygonal models for fast rendering [97]. The standard strategy is to subdivide the model's bounding box into grid cells and replace all sample points that fall into the same cell by a common representative. This volumetric approach has some drawbacks, however. By using a grid of fixed size this method cannot adapt to non-uniformities in the sampling distribution. Furthermore, volumetric clustering easily joins unconnected parts of a surface, if the grid cells are too large. To alleviate these shortcomings, surface-based clustering techniques can be applied, where clusters are build by collecting neighboring samples while regarding local sampling density. Two general approaches for building clusters will be used in this thesis. An incremental method, where clusters are created by region-growing, and a hierarchical approach that splits the point cloud into smaller subsets in a top-down manner [103]. Both methods create a set of clusters $\{C_i\}$, such that for every point $\mathbf{p}_j \in P$ there exists a unique cluster C_i with $j \in C_i$. The simplified point cloud P' is then obtained by replacing each cluster C_i by a representative sample, typically its centroid given as

$$\overline{\mathbf{p}}_i = \frac{1}{|C_i|} \sum_{j \in C_i} \mathbf{p}_j . \tag{3.1}$$

3.2.1 Incremental Clustering

Starting from a random seed point $\mathbf{p}_j \in P$, a cluster C_0 is built by successively adding nearest neighbors. This incremental region-growing is terminated when the size of the cluster reaches a maximum bound. Additionally, the maximum allowed variation σ_n of each cluster can be restricted (see Section 2.1.3). This results in a curvature-adaptive clustering method, where more and smaller clusters are created in regions of high surface variation. The next cluster C_1 is then build by starting the incremental growth with a new seed chosen from the neighbors of C_0 and excluding all points of C_0 from the region-growing. This process is terminated when all sample points of P have been assigned to a cluster in $\{C_i\}$.

Due to fragmentation, this method creates many clusters that did not reach the maximum size or variation bound, but whose incremental growth was restricted by adjacent clusters. To obtain a more even distribution of clusters, sample points of all clusters that did not reach a minimum size and variation bound (typically half the values of the corresponding maximum bounds) are distributed to neighboring clusters (see Figure 3.1). Note that this potentially increases the size and variation of the clusters beyond the user-specified maxima. Figure 3.2 illustrates incremental clustering, where oriented circular splats are used to indicate the sampling distribution.

3.2.2 Hierarchical Clustering

An alternative method for computing the set of clusters recursively splits the point cloud using a binary space partition. The point cloud P is split, if

- the size $|P|$ is larger than the user specified maximum cluster size $n_{\max}$ or
- the variation $\sigma_n(P)$ is above a maximum threshold $\sigma_{\max}$.

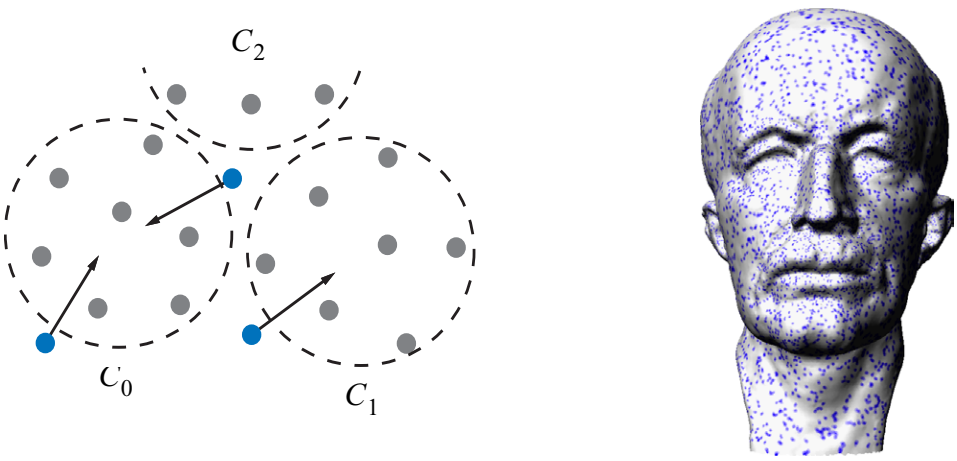


Figure 3.1 Fragmentation of incremental clustering. Gray dots correspond to sample points that are assigned to clusters that reached the necessary size and variation bounds. All other points are "stray samples" (blue dots) and will be attached to the cluster with closest centroid.

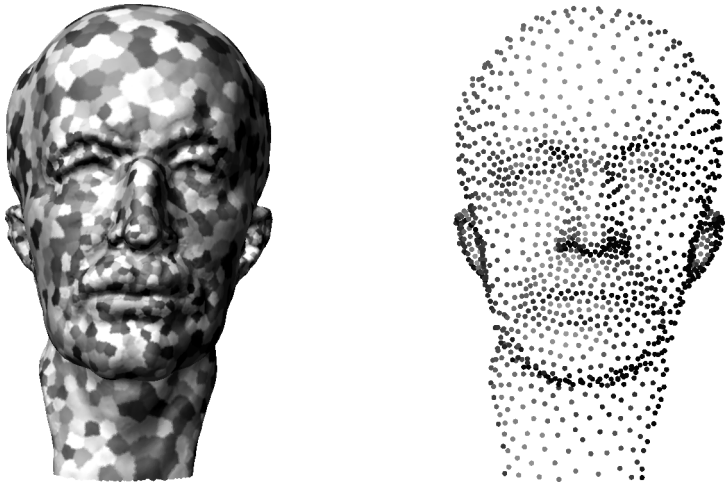


Figure 3.2 Uniform incremental clustering: The left image illustrates the corresponding clusters on the original point set (296,850 points), while the right image shows the simplified point set (2,413 points).

The split plane is defined by the centroid of P and the eigenvector $\mathbf{v}_2$ of the covariance matrix of P with largest corresponding eigenvector (see also Figure 2.2). Hence the point cloud is always split along the direction of greatest variation [103]. If the splitting criterion is not fulfilled, the point cloud P becomes a cluster C_i. As shown in Figure 3.3, hierarchical clustering builds a binary tree, where each leaf of the tree corresponds to a cluster. A straightforward extension to the recursive scheme uses a priority queue to order the splitting operations [14, 103]. While this leads to a significant increase in computation time, it allows direct control over the number of generated samples, which is difficult to achieve by specifying $n_{\max}$ and $\sigma_{\max}$ only. Figure 3.4 illustrates adaptive hierarchical clustering.

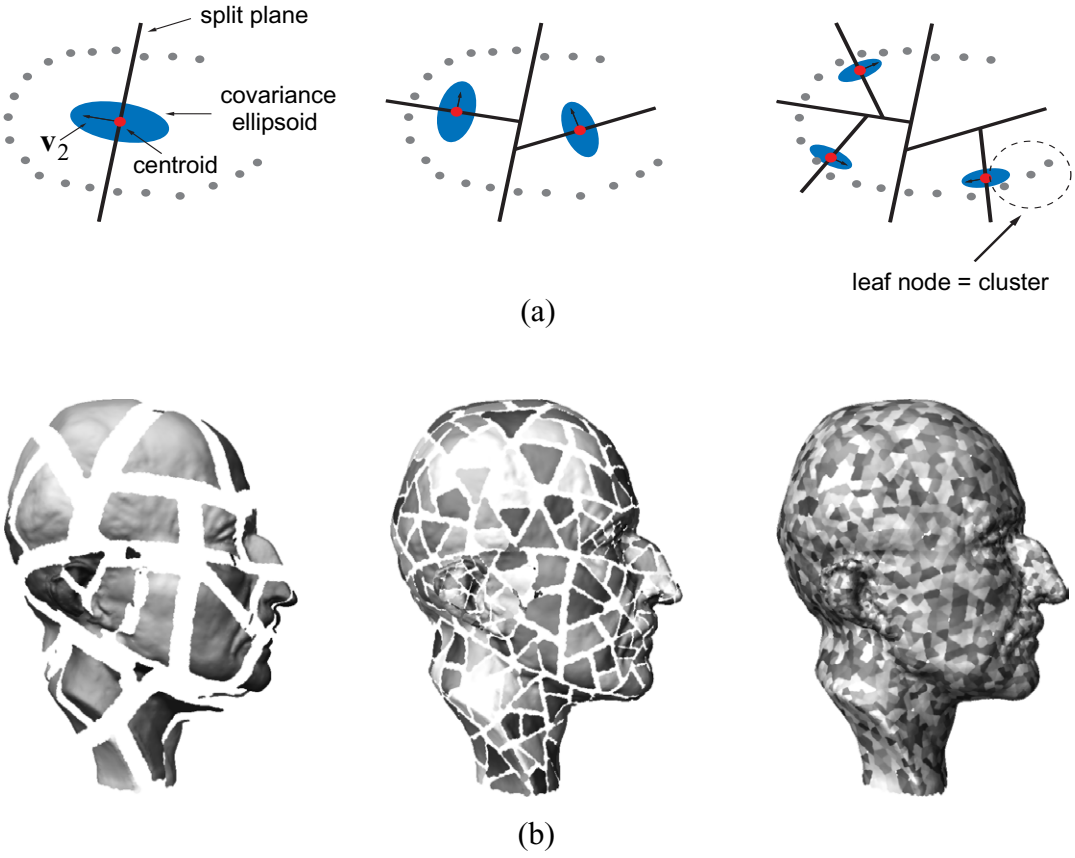


Figure 3.3 Three intermediate steps of the hierarchical clustering algorithm. (a) 2D sketch, (b) uniform hierarchical clustering for the Max Planck model.

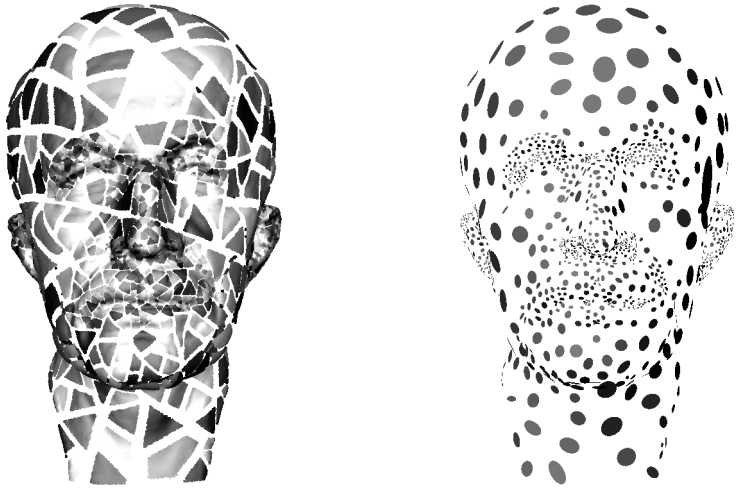


Figure 3.4 Adaptive hierarchical clustering: The left image illustrates the clusters on the original point set, while the right image shows the simplified point set (1,831 points). The size of the splats on the right image is proportional to the corresponding cluster size.

3.3 ITERATIVE SIMPLIFICATION

A different strategy for point-based surface simplification iteratively reduces the number of points using an atomic decimation operator. This approach is very similar to mesh-based simplification methods for creating progressive meshes [52]. Decimation operations are usually arranged in a priority queue according to an error metric that quantifies the error caused by the decimation. The iteration is then performed in such a way that the decimation operation causing the smallest error is applied first. Earlier work [4, 81] uses simple point removal, i.e., points are iteratively removed from the point cloud, resulting in a simplified point cloud that is a true subset of the original point set. As discussed above this can lead to undesirable artefacts, which can be avoided by using point-pair contraction instead of point removal. This extension of the common edge collapse operator replaces two points $\mathbf{p}_1$ and $\mathbf{p}_2$ by a new point $\bar{\mathbf{p}}$ implicitly applying a low-pass filter by computing a weighted average of the contracted point pair.

To rate the cost of a contraction operation, an adaptation of the quadric error metric is used as presented for polygonal meshes in [39]. The idea there is to approximate the surface locally by a set of tangent planes and to estimate the geometric deviation of a mesh vertex $\mathbf{v}$ from the surface by the sum of the squared distances to these planes. The error quadrics for each vertex $\mathbf{v}$ are initialized with a set of planes defined by the triangles around that vertex and can be represented by a symmetric 4×4 matrix $Q_{\mathbf{v}}$. The quality of the collapse $(\mathbf{v}_1, \mathbf{v}_2) \rightarrow \bar{\mathbf{v}}$ is then rated according to the minimum of the error functional $Q_{\bar{\mathbf{v}}} = Q_{\mathbf{v}_1} + Q_{\mathbf{v}_2}$.

In order to adapt this technique to the decimation of unstructured point clouds, manifold surface connectivity is replaced by the k-nearest neighbor relation (see Section 2.1.1). The error quadrics for every point sample $\mathbf{p}$ are initialized by estimating a tangent plane E_i for every *edge* that connects $\mathbf{p}$ with one of its neighbors $\mathbf{p}_i$. This tangent plane is spanned by the vector $\mathbf{e}_i = \mathbf{p} - \mathbf{p}_i$ and $\mathbf{b}_i = \mathbf{e}_i \times \mathbf{n}$, where $\mathbf{n}$ is the estimated normal vector at $\mathbf{p}$. After this initialization the point cloud decimation works exactly like mesh decimation with the point $\bar{\mathbf{p}}$ inheriting the neighborhoods of its ancestors $\mathbf{p}_1$ and $\mathbf{p}_2$ and being assigned the error functional $Q_{\bar{\mathbf{p}}} = Q_{\mathbf{p}_1} + Q_{\mathbf{p}_2}$. Figure 3.5 shows an example of a simplified point cloud created by iterative point-pair contraction.

3.4 PARTICLE SIMULATION

In [111], Turk introduced a method for re-sampling polygonal surfaces using particle simulation. The desired number of particles is randomly spread across the surface and their position is equalized using a point repulsion algorithm. Point movement is restricted to the surface defined by the individual polygons to ensure an accurate approximation of the original surface. Turk also included a curvature estimation method to concentrate more samples in regions of high curvature. Finally, the new vertices are re-triangulated yielding the re-sampled triangle mesh. This scheme can easily be adapted to point-sampled geometry.

3.4.1 Spreading Particles

Turk initializes the particle simulation by randomly distributing points on the surface. Since a uniform initial distribution is crucial for fast convergence, this random choice

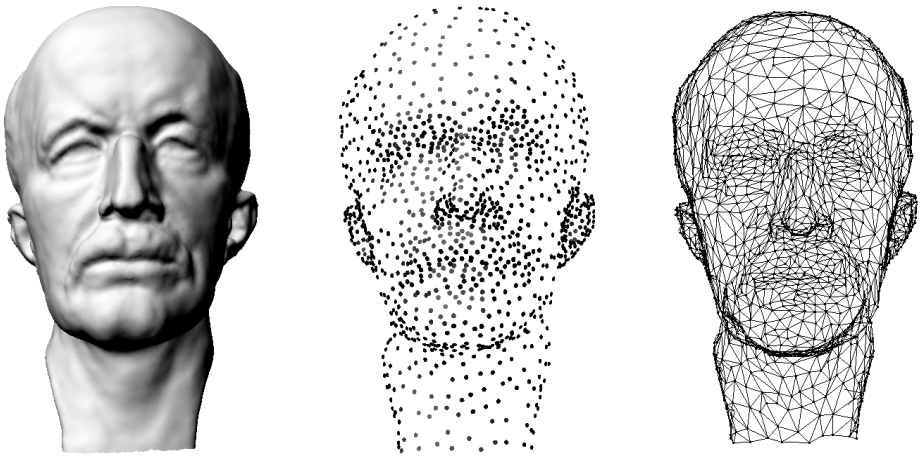


Figure 3.5 Iterative simplification of the Max Planck model from 296,850 (left) to 2,000 sample points (middle). The right image shows all remaining potential point-pair contractions indicated as an edge between two points. Note that these edges do not necessarily form a consistent triangulation of the surface.

is weighted according to the area of the polygons. For point-based models, this area measure can be replaced by a density estimate ρ (see Section 2.1.2). Thus by placing more samples in regions of lower sampling density (which correspond to large triangles in the polygonal setting), uniformity of the initial sample distribution can ensured.

3.4.2 Repulsion

For repulsion the same linear force term is used as in [111], because its radius of influence is finite, i.e., the force vectors can be computed very efficiently as

$$F_i(\mathbf{p}) = k(r - \|\mathbf{p} - \mathbf{p}_i\|) \cdot (\mathbf{p} - \mathbf{p}_i), \tag{3.2}$$

where $F_i(\mathbf{p})$ is the force exerted on particle $\mathbf{p}$ due to particle $\mathbf{p}_i$, k is a force constant and r is the repulsion radius. The total force exerted on $\mathbf{p}$ is then given as

$$F(\mathbf{p}) = \sum_{i \in N_p} F_i(\mathbf{p}), \tag{3.3}$$

where N_p is the neighborhood of $\mathbf{p}$ with radius r. Using a 3D grid data structure, this neighborhood can be computed efficiently in constant time (see also Section 8.2.2).

3.4.3 Projection

In Turk's method, displaced particles are projected onto the closest triangle to prevent the particles from drifting away from the surface. Since no explicit surface representation is available, the MLS projection operator Ψ (see Section 2.3.2) is applied to keep the particles on the surface. However, applying this projection every time a particle position is altered is computationally too expensive. Therefore, a different approach has been used: A particle $\mathbf{p}$ is kept close to the surface by simply projecting it onto the tangent plane of the point $\mathbf{p}'$ of the original point cloud that is closest to $\mathbf{p}$. The full moving least squares projection is only applied at the end of the

simulation, which alters the particle positions only slightly and does not change the sampling distribution noticeably.

3.4.4 Adaptive Simulation

Using the variation estimate of Section 2.1.3, more points can be concentrated in regions of high curvature by scaling their repulsion radius with the inverse of the variation σ_n. It is also important to adapt the initial spreading of particles accordingly to ensure fast convergence. This can be done by replacing the density estimate ρ by $\rho \cdot \sigma_n$. Figure 3.6 gives an example of an adaptive particle simulation.

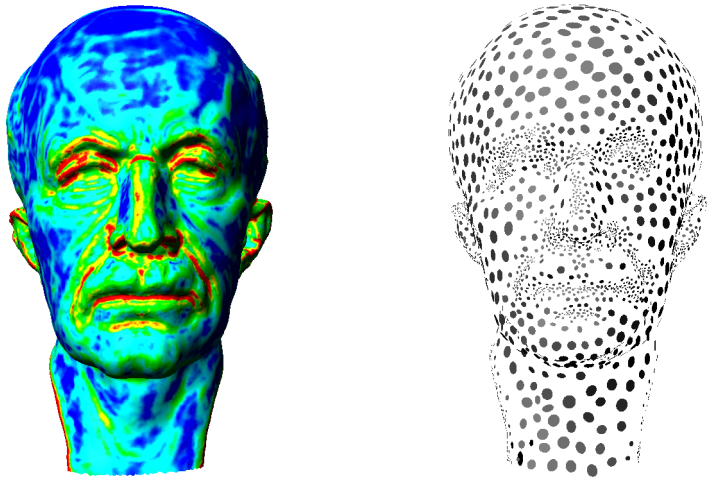


Figure 3.6 Simplification by adaptive particle simulation. The left image shows repulsion radius on the original model determined from the surface variation estimate. Blue indicates a large radius, while red indicates small radius. On the right, a simplified model consisting of 3,000 points is shown, where the size of the splats is proportional to the repelling force of the corresponding particle.

Scaling of the repulsion radius also provides an easy means for user-controlled surface re-sampling. For this purpose a painting tool has been developed (see also Chapter 7) that allows the user to directly paint the desired repulsion radius onto the surface and thus control the sampling distribution of the re-sampled surface. As illustrated in Figure 3.7, particle simulation automatically creates smooth transitions between areas of different sampling density.

3.5 COMPARISON

The previous sections have introduced different algorithms for point-based surface simplification. As mentioned before, none of these methods attempts to find an optimal point distribution with respect to a global distance metric, but rather uses some built-in heuristics to approximate such an optimum:

- Clustering methods try to partition the point cloud into clusters of equal size and/or surface variation, assuming that each cluster describes an equally important part of the surface.

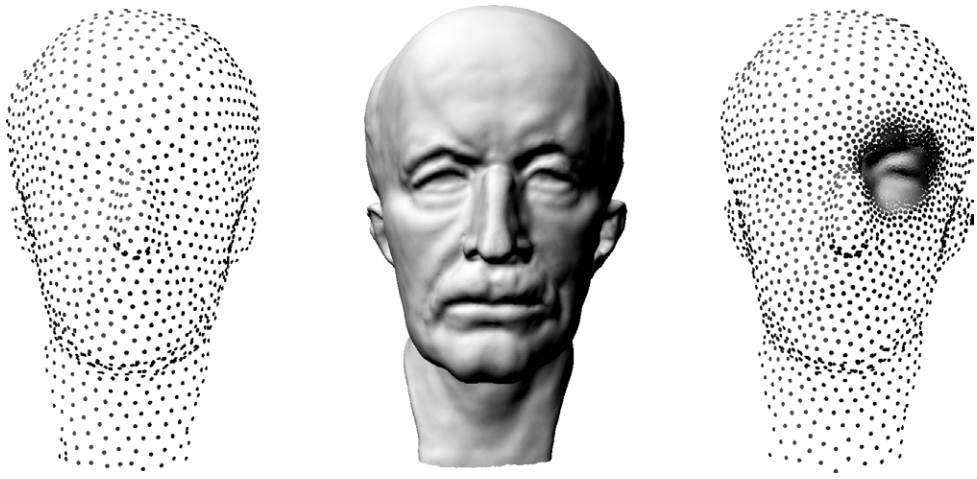


Figure 3.7 Uniform and user-controlled particle simulation. The model in the middle has been simplified to 2,000 points in the left image using uniform repulsion forces. On the right, the repulsion radius has been scaled down to 10% around the eye, leading to a higher concentration of samples in this region.

- Iterative simplification optimizes locally according to the quadric error metric.
- Particle simulation is based on the assumption that a minimum of the potential energy of the particles minimizes the distance between original and re-sampled surface.

Since these heuristics are fundamentally different, the surfaces generated by these algorithms will differ significantly in general. Thus, to evaluate and compare the quality of the simplified surfaces, some generic technique for measuring the geometric distance between two point-sampled surfaces is required. This distance measure should be general in the sense that no additional knowledge about the specific method used to generate the simplified surface should be required.

Further aspects in the evaluation of the various simplification algorithms include sampling distribution of the simplified model, time and space efficiency, and implementational issues.

3.5.1 Surface Error

Assume that two point clouds P and P' are given, which represent two surfaces S and S', respectively. The distance, or error, between these two surfaces is measured using a sampling approach similar to the method applied in the Metro tool [18].

Let Q be a set of points on S and let $d(\mathbf{q}, S') = \min_{\mathbf{x} \in S'} d(\mathbf{q}, \mathbf{x})$ be the minimum distance of a point $\mathbf{q} \in Q$ to the surface S'. Then two error measures can be defined:

- *Maximum error:*

$$\Delta_{\max}(S, S') = \max_{\mathbf{q} \in Q} d(\mathbf{q}, S') \tag{3.4}$$

The maximum error approximates the two-sided Hausdorff distance of the two surfaces. Note that the surface-based approach is crucial for meaningful error estimates, as the Hausdorff distance of the two point sets P and P' does not adequately measure the distance between S and S'. As an example, consider a

point cloud P' that has been created by randomly subsampling P. Even though the corresponding surfaces can be very different, the Haussdorff distance of the point sets will be zero.

- *Average error:*

$$\Delta_{\text{avg}}(S, S') = \frac{1}{|Q|} \sum_{\mathbf{q} \in Q} d(\mathbf{q}, S') \tag{3.5}$$

The average error approximates the area-weighted integral of the point-to-surfaces distances.

The point set Q is created using the uniform particle simulation of Section 3.4. This allows the user to control the accuracy of the estimates (3.4) and (3.5) by specifying the number of points in Q. To obtain a visual error estimate, the sample points of Q can be color-coded according to the point-to-surface distance $d(q, S')$ and rendered using a standard point rendering technique (see Figures 3.9 and 3.10).

$d(\mathbf{q}, S')$ is calculated using the MLS projection operator Ψ with linear basis functions (see Section 2.3.3). Effectively, Ψ computes the closest point $\mathbf{q}' \in S'$ such that $\mathbf{q} = \mathbf{q}' + d \cdot \mathbf{n}$ for a $\mathbf{q} \in Q$, where $\mathbf{n}$ is the surface normal at $\mathbf{q}'$ and d is the distance between $\mathbf{q}$ and $\mathbf{q}'$ (see Figure 3.8). Thus the point-to-surface distance $d(\mathbf{q}, S')$ is given as $d = \|\mathbf{q} - \mathbf{q}'\|$ (see also the decomposition operator defined in Section 4.3).

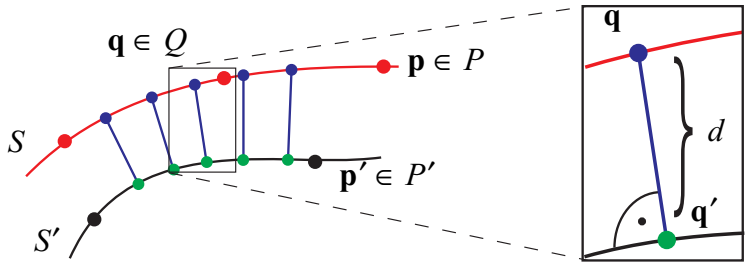
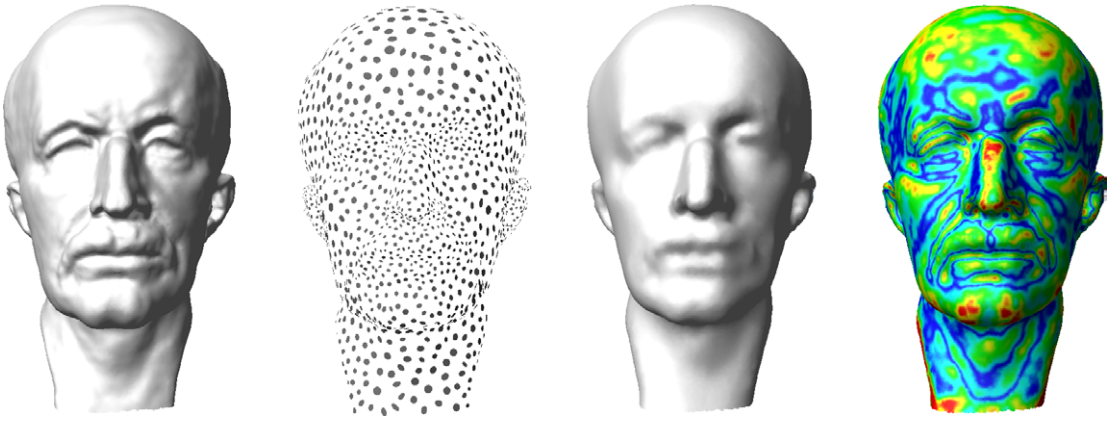


Figure 3.8 Measuring the distance between two surfaces S (red curve) and S' (black curve) represented by two point sets P (red dots) and P' (black dots). P is up-sampled to Q (blue dots) and for each $\mathbf{q} \in Q$ a base point $\mathbf{q}' \in S'$ is found (green dots), such that the vector $\mathbf{q} - \mathbf{q}'$ is orthogonal to S'. The point-to-surface distance $d(\mathbf{q}, S')$ is then equal to $d = \|\mathbf{q} - \mathbf{q}'\|$.

Figure 3.10 shows visual and quantitative error estimates (scaled according to the object's bounding box diagonal) for the David model that has been simplified from 2,000,606 points to 5,000 points. Uniform incremental clustering has the highest average error. Since all clusters consist of roughly the same number of sample points, most of the error is concentrated in regions of high curvature. Adaptive hierarchical clustering performs slightly better, in particular in the geometrically complex regions of the hair. Iterative simplification and particle simulation provide lower average error and distribute the error more evenly across the surface. In general the iterative simplification method using quadric error metrics has been found to produce the lowest average surface error.

Clustering methods perform worse with respect to surface error, because they do not have the fine-grain adaptivity of the iterative simplification and particle simulation



(a) (b) (c) (d)

Figure 3.9 Measuring surface error. (a) original surface, (b) simplified point cloud, (c) surface obtained by up-sampling the simplified point cloud, (d) color-coded error, where blue corresponds to a small error, while red indicates a large error.

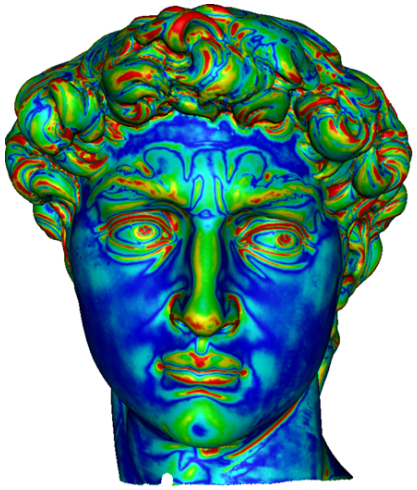
methods. For example, consider the hierarchical clustering algorithm as illustrated in Figure 3.3. The split planes generated in the earlier stages of the recursion are propagated down to all subsequent levels. This means that once a top-level split plane has been chosen, the method cannot adapt to local variations across that split plane, which leads to increased surface error.

3.5.2 Sampling Distribution

Apart from the geometric error, the distribution of samples within the surface can be an important aspect for certain applications. As mentioned before, all simplification algorithms presented in this thesis create a point cloud that is in general not a subset of the original sample set. Where this is required, methods such as those presented by Alexa et al. [4] or Linsen [81] are preferable.

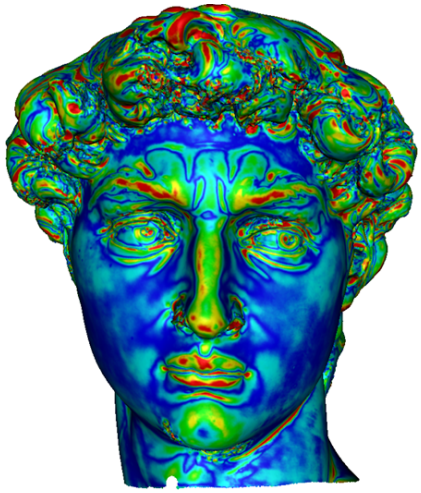
For clustering methods the sampling distribution in the final model is closely linked to the sampling distribution of the input model. In some applications this might be desirable, e.g., where the initial sampling pattern carries some semantic information, such as in geological models [55]. Other applications, e.g., pyramid algorithms for multi-level smoothing [64] or texture synthesis [112], require uniform sampling distributions, even for highly non-uniformly sampled input models. Here non-adaptive particle simulation is most suitable, as it distributes sample points uniformly and independently of the sampling distribution of the underlying surface. As illustrated in Figure 3.7, particle simulation also provides a very easy mechanism for locally controlling the sampling density by scaling the repulsion radius accordingly. While similar effects can be achieved for iterative simplification by penalizing certain point-pair contractions, particle simulation offers much more intuitive control.

Note that none of the surface simplification methods described above gives any guarantees that the resulting point cloud satisfies the sampling criteria specified in Section 2.2. It is rather left to the application to specify a suitable target sampling rate.



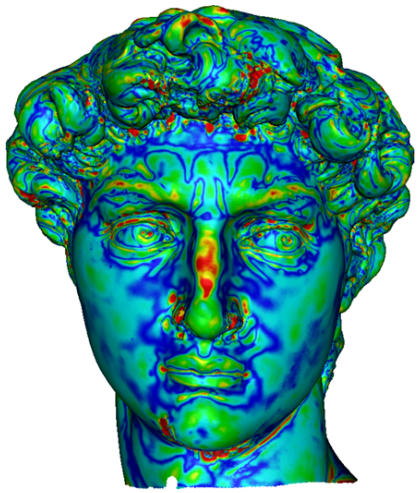
$\Delta_{avg} = 6.32 \cdot 10^{-4}$ $\Delta_{max} = 0.0049$

(a) uniform incremental clustering



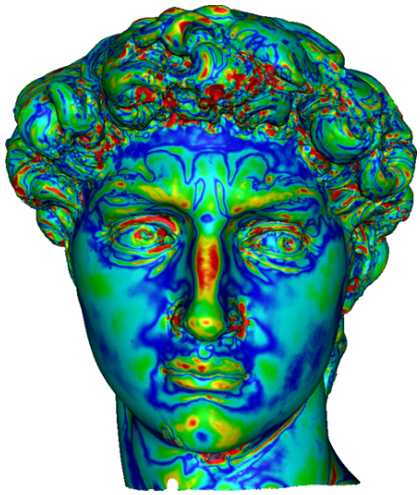
$\Delta_{avg} = 6.14 \cdot 10^{-4}$ $\Delta_{max} = 0.0046$

(b) adaptive hierarchical clustering



$\Delta_{avg} = 5.43 \cdot 10^{-4}$ $\Delta_{max} = 0.0052$

(c) iterative simplification



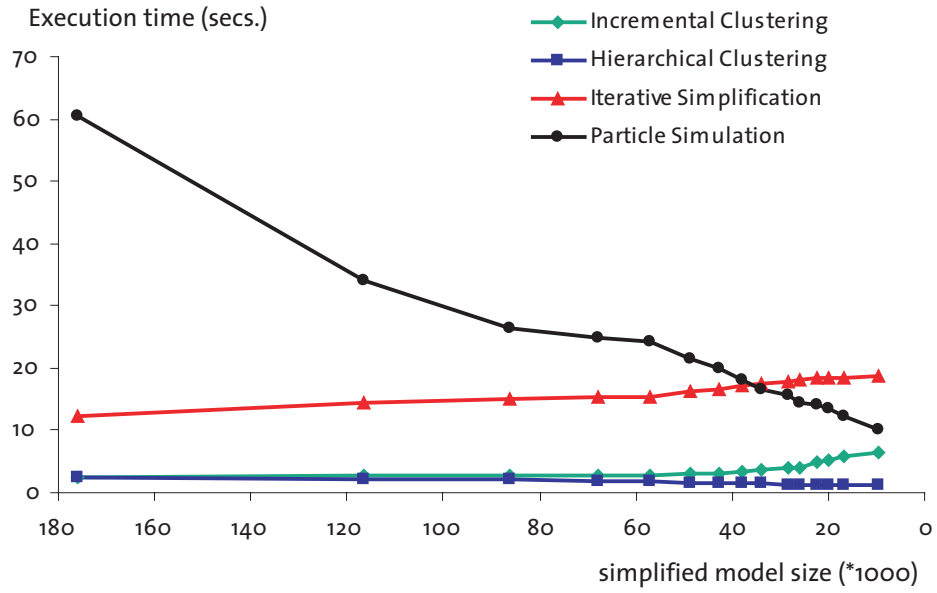
$\Delta_{avg} = 5.69 \cdot 10^{-4}$ $\Delta_{max} = 0.0061$

(d) particle simulation

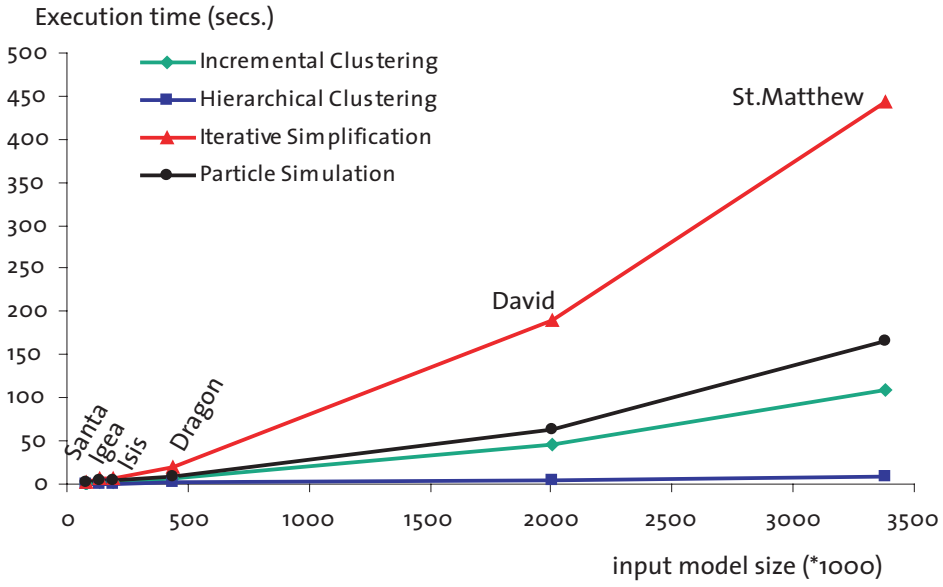
Figure 3.10 Surface Error for Michelangelo's David simplified from 2,000,606 points to 5,000 points.

3.5.3 Computational Effort

Figure 3.11 shows computation times for the different simplification methods both as a function of target model size and input model size. Due to the simple algorithmic structure, clustering methods are by far the fastest simplification techniques presented in this thesis. Iterative simplification has a relatively long pre-computing phase, where initial contraction candidates and corresponding error quadrics are determined and the priority queue is set up. The simple additive update rule of the quadric metric (see Section 3.3) make the simplification itself very efficient, however. In the current implementation particle simulation is the slowest simplification technique for large target model sizes, mainly due to slow convergence of the relaxation step. A possible



(a)



(b)

Figure 3.11 Execution times for simplification, measured on a Pentium 4 (1.8GHz) with 1Gb of main memory: (a) as a function of target model size for the dragon model (435,545 points), (b) as a function of input model size for a simplification to 1%.

improvement is the hierarchical approach introduced in [114]. The algorithm would start with a small number of particles and relax until the particle positions have reached equilibrium. Then particles are split, their repulsion radius is adapted and relaxation continues. This scheme can be repeated until the desired number of particles is obtained.

It is interesting to note that for incremental clustering and iterative simplification the execution time increases with decreasing target model size, while hierarchical

clustering and particle simulation are more efficient the smaller the target models. Thus the latter are more suitable for real-time applications where the fast creation of coarse model approximations is crucial.

3.5.4 Memory Requirements and Data Structures

Currently all simplification methods presented in this thesis are implemented in-core, i.e., require the complete input model as well as the simplified point cloud to reside in main memory. For incremental clustering a balanced kd-tree is used for fast nearest-neighbor queries, which can be implemented efficiently as an array [101], requiring $4 \cdot n$ bytes, where n is the size of the input model. Hierarchical clustering builds a BSP tree, where each leaf node corresponds to a cluster. Since the tree is built by re-ordering the sample points, each node only needs to store the start and end index in the array of sample points and no additional pointers are required. Thus this maximum number of additional bytes is $2 \cdot 2 \cdot 4 \cdot m$, where m is the size of the simplified model. Iterative simplification requires 96 bytes per point contraction candidate, 80 of which are used for storing the error quadric (floating point numbers are stored in double precision, since single precision floats lead to numerical instabilities). Assuming six initial neighbors for each sample point, this amounts to $6/2 \cdot 96 \cdot n$ bytes. Particle simulation uses a 3D grid data structure with bucketing to accelerate the nearest neighbor queries, since a static kd-tree is not suitable for dynamically changing particle positions. This requires a maximum of $4 \cdot (n + k)$ bytes, where k is the resolution of the grid (see also Section 8.2).

Thus incremental clustering, iterative simplification and particle simulation need additional storage that is linearly proportional to the number of input points, while the storage overhead for hierarchical clustering depends only on the target model size.

3.5.5 Comparison to Mesh Simplification

Figure 3.12 shows a comparison of point-based simplification and simplification for polygonal meshes. In (a), the initial point cloud is simplified from 134,345 to 5,000 points using the iterative simplification method of Section 3.3. The resulting point cloud has then been triangulated using the surface reconstruction method of [40]. In (b), the input point cloud has first been triangulated and the resulting polygonal surface has then been simplified using the mesh simplification tool QSlim [39]. Both methods produce similar results in terms of surface error and both simplification processes take approximately the same time (~3.5 seconds). However, creating the triangle mesh from the simplified point cloud took 2.45 seconds in (a), while in (b) reconstruction time for the input point cloud was 112.8 seconds. Thus when given a large unstructured point cloud, it is much more efficient to first do the simplification on the point data and then reconstruct a mesh (if desired) than to first apply a reconstruction method and then simplify the triangulated surface. This illustrates that point-based simplification methods can be very useful when dealing with large geometric models stemming from 3D acquisition.

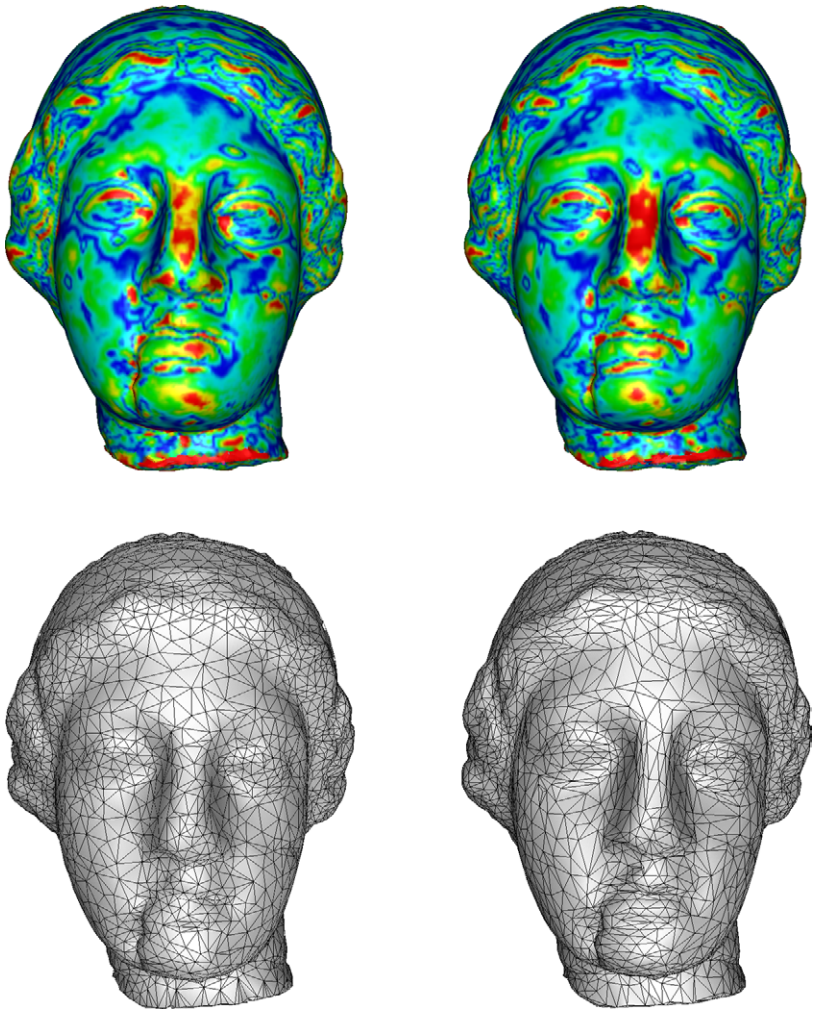
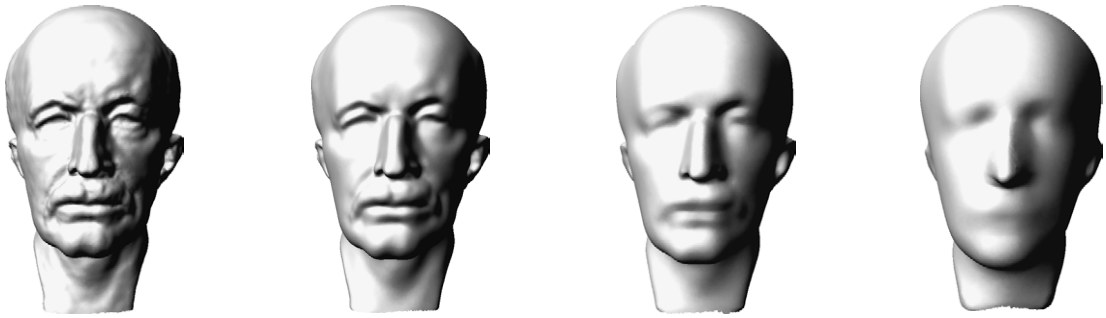


Figure 3.12 Comparison of point-based (left column) and mesh-based (right column) surface simplification. The top row shows the error images, the bottom row the final, triangulated surfaces.

4

MULTI-SCALE SURFACE REPRESENTATION



In this chapter the surface representation introduced in Section 2.3.2 is extended from a single-scale representation to a multi-scale representation, using the concept of scale-space. This extension is motivated by the need for higher level editing semantics, which allow surface modifications at different scales. The user should be able to edit the surface at different approximation levels to perform coarse-scale edits on the whole model as well as very localized modifications on the surface detail. To this end, a multi-scale surface representation provides a set of surface approximations with different levels of geometric detail. Depending on the intended editing operation, a suitable level is determined and the operation is performed on the corresponding surface approximation.

A key issue in the definition of a multi-scale surface representation is the interconnection between successive levels in the hierarchy. To obtain intuitive editing semantics, it is essential that changes in one level are propagated naturally to the next higher levels. This can be achieved by encoding each level of the discrete multi-scale representation as a normal displacement of its immediate smoother approximation. Thus the "difference" between two successive levels can be expressed as a set of scalar detail coefficients that determine for each point the distance between the two approximations in normal direction. These detail coefficients can be considered as

discrete frequency bands that allow spectral filtering methods to be applied to point-sampled models (see Section 4.4).

It is important to note the distinction between a multi-scale and a multi-resolution surface representation. The former describes a surface at different levels of smoothness, without any reference to a particular sampling distribution. The latter, on the other hand, refers to a set of surface approximations with varying sampling resolution, thus describing a surface at different levels of coarseness (see also [64]). Multi-resolution surface representations have been used successfully in the context of efficient rendering [97], surface compression and progressive transmission [62], surface analysis [28, 57], and morphing [70].

Recent surface editing systems combine multi-scale and multi-resolution representations, using, for example, multi-resolution subdivision surfaces [116]. However, Kobbelt et al. observed in [64] that rigid multi-resolution representations like subdivision hierarchies can lead to less flexibility when defining editing metaphors. In this thesis the focus is on surface editing, so only a multi-scale representation will be defined. This requires two main building blocks:

- A *fairing operator*, i.e., a geometric low-pass filter that generates successively smoother approximation levels of a given input surface and
- a *decomposition operator*, i.e., a method to encode each level relative to the next smoother level using normal displacements to ensure intuitive detail preservation.

After a brief introduction to scale-space, various fairing operators will be derived from a discrete approximation of a surface diffusion process. Then a multi-scale decomposition operator will be defined based on the MLS projection method. Since the MLS projection also implements a geometric low-pass filter (see Section 2.3.2), the decomposition operator can be extended to directly incorporate the smoothing approximation. Using the surface simplification methods of Chapter 3, this leads to an efficient multi-level decomposition operator for point-sampled surfaces. Section 4.4 shows how the multi-scale representation can be used to define discrete spectral filters that implement advanced modeling operations such as enhancement or band-stop filtering. The chapter is concluded in Section 4.5 with a discussion on multi-scale surface deformation.

4.1 SCALE-SPACE FOR FUNCTIONS

Scale-space methods have been used extensively in image and volume data analysis for the last two decades (see [78] for an overview). The idea is to model a signal at different approximation levels, or scales, to better analyze the inherent structures of the signal.

Given a d-dimensional signal f: $\mathbb{R}^d \to \mathbb{R}$, its *linear scale-space representation* L: $\mathbb{R}^d \times \mathbb{R}_+ \to \mathbb{R}$ is defined as the convolution

$$L(\mathbf{x}, t) = f(\mathbf{x}) \otimes g(\mathbf{x}, t), \tag{4.1}$$

where t is the *scale parameter* and

$$g(\mathbf{x}, t) = \frac{1}{(2\pi t)^{d/2}} e^{-\mathbf{x}^T\mathbf{x}/(2t)} \tag{4.2}$$

is a Gaussian kernel whose standard deviation σ_t is related to the scale parameter through $\sigma_t = \sqrt{t}$. The scale-space representation of Equation 4.1 is often used in combination with Fourier techniques, which allow an efficient evaluation of the convolution operation [13].

It was first realized by Koenderink [66] that the generating equation of a linear scale-space representation L is the linear diffusion equation. So L can also be obtained as the solution to the diffusion equation

$$\frac{\partial L}{\partial t} - \lambda \cdot \Delta L = 0, \tag{4.3}$$

where λ is the diffusion constant and Δ the Laplacian operator. The initial conditions of Equation (4.3) are given as $L(\mathbf{x}, 0) = f(\mathbf{x})$, suitable boundary conditions depend on the specific application. A detailed discussion of scale-space techniques can be found in [78].

4.1.1 Discrete Scale-Space Representation for Height Fields

Before extending the concept of scale-space to point-sampled surfaces, this section briefly describes how a discrete scale-space representation can be defined for regularly sampled functions in 2D. Assume an $(N_x + 1)(N_y + 1)$ discrete sample $\{f_{i,j}\}$ of a function $f\colon [0, 1] \times [0, 1] \to \mathbb{R}$ is given as

$$f_{i,j} = f(x_i, y_j) \qquad \begin{matrix} x_i = i/N_x & i = 0, \ldots, N_x \\ y_j = j/N_y & j = 0, \ldots, N_y \end{matrix}. \tag{4.4}$$

Equation (4.3) can be written as

$$\frac{\partial L}{\partial t}(x, y, t) - \lambda \cdot \left(\frac{\partial^2 L}{\partial x^2}(x, y, t) + \frac{\partial^2 L}{\partial y^2}(x, y, t) \right) = 0. \tag{4.5}$$

The discretization of this equation in the time interval $t \in [0, T]$ yields a set of unknown samples $l_{i,j,k} = L(x_i, y_j, t_k)$ with $t_k = k \cdot T/N_t$, $k = 0, \ldots, N_t$ and $l_{i,j,0} = f_{i,j}$. The second order derivatives can be approximated by second order divided differences as

$$\frac{\partial^2 L}{\partial x^2}(x, y, t) = \frac{l_{i-1,j,k} - 2l_{i,j,k} + l_{i+1,j,k}}{h_x^2} + O(h_x^2), \tag{4.6}$$

where $h_x = 1/N_x$. Using an analogous expression for the second derivative with respect to y, this leads to an explicit Euler equation of the form

$$l_{i,j,k+1} = l_{i,j,k} + \lambda h_t \cdot \Delta l_{i,j,k}, \tag{4.7}$$

where $h_t = T/N_t$ and

$$\Delta l_{i,j,k} = \frac{l_{i-1,j,k} - 2l_{i,j,k} + l_{i+1,j,k}}{h_x^2} + \frac{l_{i,j-1,k} - 2l_{i,j,k} + l_{i,j+1,k}}{h_y^2} \tag{4.8}$$

is a discrete approximation of the Laplacian. This explicit integration scheme can be reformulated as an implicit Euler integration as

$$(1 - \lambda h_t \Delta) l_{i,j,k+1} = l_{i,j,k}, \tag{4.9}$$

which requires the solution of a linear system, but considerably improves the stability and thus allows larger time steps. Equation (4.7) defines a discrete sample of the continuous scale-space representation of the function f both in space and in time. Figure 4.1 shows examples of such a sample for 2D image data and an artificial terrain model. Observe how high-frequency detail is gradually smoothed out with increasing scale parameter.

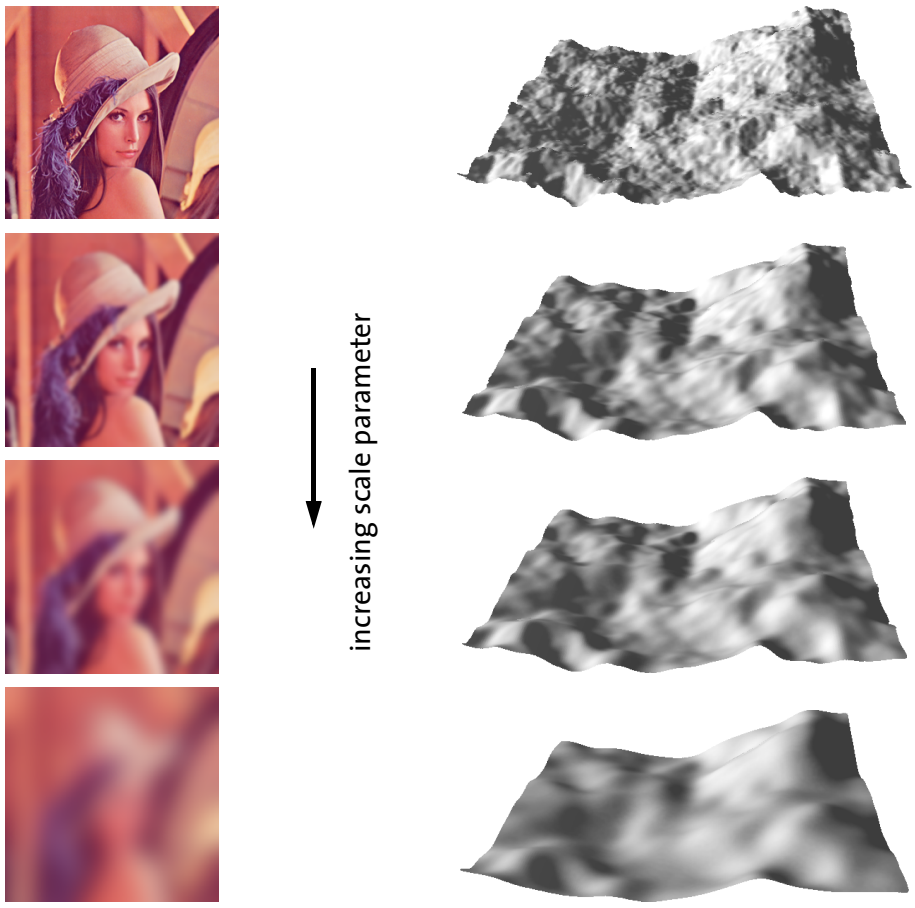


Figure 4.1 Discrete scale-space representation of 2D image data (left column) and discrete height field data (right column).

4.2 SCALE-SPACE FOR POINT-SAMPLED SURFACES

The derivation of a discrete scale-space representation for point-sampled surfaces follows the same path described in the previous section for the functional setting. First a continuous definition of scale-space for surfaces is given using the surface diffusion equation. A discretization both in space and time then yields an explicit or implicit integration scheme for computing smoother approximations of the input surface. Special attention has to be given to a suitable discretization of the Laplacian, as neither

global parameterization nor regular sampling pattern are provided in the surface setting.

Continuous Formulation

Assume a surface S is given. The surface diffusion equation defines a continuous set of evolving surfaces $S(t)$ subject to

$$\frac{\partial \mathbf{x}}{\partial t} - \Delta \mathbf{x} = 0, \tag{4.10}$$

where $\mathbf{x} \in S(t)$ is a point on the surface, $t \in [0, \infty) \subset \mathbb{R}$ is the time or scale parameter and $\Delta \mathbf{x} = \kappa \cdot \mathbf{n}$ denotes the Laplace-Beltrami operator with κ the mean curvature and $\mathbf{n}$ the surface normal at $\mathbf{x}$ (see [25]). The initial condition to Equation (4.10) is given as $S(0) = S$.

Equation (4.10) defines an evolving surface, where each point on the surface moves in the direction defined by the surface normal with a speed given by the mean curvature. This method, also known as *mean curvature flow,* has been studied in the context of evolving interfaces [102]. An interesting property of mean curvature flow is that the evolving surface converges to a minimal surface, which by definition has zero mean curvature.

4.2.1 Discrete Fairing

For discrete surfaces, a variety of *surface fairing* methods have been introduced in recent years based on a discretization of Equation 4.10. Taubin pioneered these methods and presented various approximate low-pass filters for triangle meshes using different discrete approximations of the Laplacian [108]. His approach aims at generalizing signal processing techniques to manifold surfaces. For this purpose he defines discrete geometric frequencies (see also [43]) as the eigenvectors of the Laplacian matrix, which describes the discretization of the Laplacian as a weighted sum of adjacent vertices. This discretization leads to the common iterative explicit Euler integration formula for Gaussian smoothing

$$\mathbf{v}' = \mathbf{v} + \lambda \Delta \mathbf{v}, \tag{4.11}$$

where $\mathbf{v}'$ is the new mesh vertex position and $\Delta \mathbf{v}$ is some discrete approximation of the Laplacian at vertex $\mathbf{v}$ (see also Equation 4.7). Taubin demonstrates that this iterative scheme attenuates high frequencies while preserving low frequencies, and thus implements approximate low-pass filter behavior. A complete derivation of the discretization of the Laplacian can be found in [55] and will here only briefly be recalled for completeness.

For a smooth manifold surface S, the Laplacian can be defined as

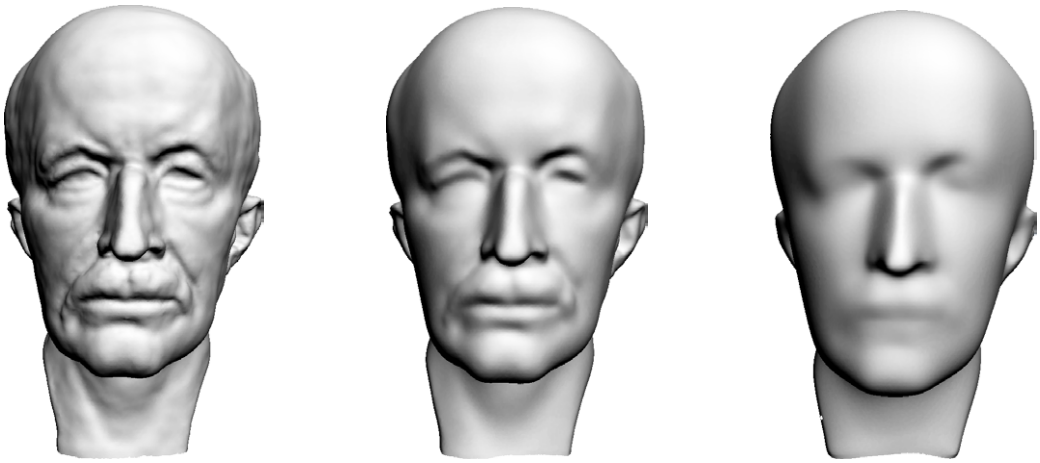
$$\Delta \mathbf{x} = \frac{2}{\pi} \int_0^{\pi} \frac{\partial^2}{\partial r} S(0,0)\, d\varphi, \qquad r = (\cos\varphi, \sin\varphi), \tag{4.12}$$

where S is parameterized locally such that $\mathbf{x} = S(0,0)$. To achieve independence of a specific parameterization, $\Delta \mathbf{x}$ is defined as the normalized integral over all possible parameterizations. Discretization of Equation 4.12 finally leads to

$$\Delta \mathbf{v} = \frac{1}{\Omega} \sum_{i \in N_v} \omega_i (\mathbf{v}_i - \mathbf{v}), \qquad \Omega = \sum_{i \in N_v} \omega_i , \tag{4.13}$$

where the point $\mathbf{x} \in S$ is replaced by a vertex $\mathbf{v}$ of a triangle mesh surface and N_v describes the one-ring neighborhood of $\mathbf{v}$. The weights ω_i depend on further simplifying assumptions as discussed in detail in [55]. For example, the uniform umbrella operator is obtained for $\omega_i = 1, \forall i$, assuming that all edges $\mathbf{v}_i - \mathbf{v}$ have uniform length and all angles of the incident triangles at $\mathbf{v}$ are constant. A more adaptive scale-dependent umbrella operator that takes different edge lengths into account results from $\omega_i = 1 / \|\mathbf{v}_i - \mathbf{v}\|$.

These concept can be extended to point clouds by replacing the one-ring neighborhood relation with a point neighborhood relation introduced in Section 2.1.1. The degree of smoothness can be controlled by the parameter λ (see Equation 4.11), the number m of iterations and the size k of the local neighborhoods of the sample points (see Figure 4.2). To improve the efficiency of the computations, local neighborhoods should computed at the beginning of the smoothing operation and cached during the iteration. This also increases the stability of the smoothing filter, since it prevents clustering effects due to the tangential drift of sample points.



original model

$\lambda = 1$
$m = 20$
$k = 20$

$\lambda = 1$
$m = 2$
$k = 1000$

Figure 4.2 Iterative Laplacian smoothing.

4.3 DISCRETE MULTI-SCALE SURFACE REPRESENTATION

Section 4.2 describes how a scale-space approximation of a given point-sampled surface can be computed using geometric low-pass filtering. To define a discrete multi-scale surface representation, subsequent approximation levels need to be encoded relative to each other in a meaningful way. This is done by means of a *decomposition operator* that encodes a detailed surface as a normal displacement of its smooth approximation. As will be demonstrated below, normal displacements ensure intuitive editing semantics and provide a compact representation of surface detail. While the

decomposition operator defines the transition from detailed to smooth approximation, the *reconstruction operator* describes the inverse operation, i.e., adds detail to a smooth surface.

Formally, a discrete multi-scale representation for point-sampled surfaces can be defined as follows:

Let $P = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ be a point cloud representing a surface S and $S(t)$ a continuous multi-scale representation of S as defined by Equation 4.10. A discrete, point-based multi-scale representation of S is a sequence of point clouds $\mathbf{P} = \{P^0, \ldots, P^k\}$, such that

- for all $l \in \{0, k-1\}$ there exists a $t_l \in [0, \infty)$ such that the surface represented by P^l approximates $S^l = S(t_l)$ with $t_l > t_{l+1}$, where $t_k = 0$ and $P^k = P$,
- $|P^l| = |P| = n$ for all $l \in \{0, k\}$
- for all $\mathbf{p}_i^l \in P^l$ and all $l \in \{1, k\}$ there exists a $\mathbf{p}_i^{l-1} \in P^{l-1}$ such that

$$\mathbf{p}_i^l = \mathbf{p}_i^{l-1} + d_i^{l-1} \cdot \mathbf{n}_i^{l-1}, \tag{4.14}$$

where $\mathbf{n}_i^{l-1}$ is the surface normal at $\mathbf{p}_i^{l-1}$ and $d_i^{l-1} \in \mathbb{R}$ a scalar-valued detail coefficient.

Thus each sample $\mathbf{p}_i \in P$ is represented by a point $\mathbf{p}_i^0 \in P^0$ plus a sequence of normal displacement offsets $d_i^0, \ldots, d_i^{k-1}$. To reconstruct the position of $\mathbf{p}_i^l$ at a certain level l the point $\mathbf{p}_i^0$ is recursively displaced in normal direction, i.e.,

$$\mathbf{p}_i^l = \mathbf{p}_i^0 + d_i^0 \mathbf{n}_i^0 + d_i^1 \mathbf{n}_i^1 + \ldots + d_i^{l-1} \mathbf{n}_i^{l-1}. \tag{4.15}$$

Let $\mathbf{D} = \{D^0, \ldots, D^{k-1}\}$, where $D^l = \{d_0^l, \ldots, d_n^l\}$ is the set of detail coefficients at level l. The reconstruction operator $+ : (\mathbf{P}, \mathbf{D}) \rightarrow \mathbf{P}$ can be defined by applying Equation 4.14 for each point of the argument point cloud, such that

$$P^l = P^{l-1} + D^{l-1}. \tag{4.16}$$

Then the reconstruction of a point cloud P^l can be written as

$$P^l = P^0 + \sum_{m=0}^{l-1} D^m, \tag{4.17}$$

which directly corresponds to Equation 4.15. The inverse of the reconstruction operator is the decomposition operator $- : (\mathbf{P}, \mathbf{P}) \rightarrow \mathbf{D}$, that determines the detail coefficients of the normal displacement offset between two point clouds. Figure 4.3 shows a discrete multi-scale representation of the Max Planck model. The 2D drawing illustrates how the set of point clouds $\{P^0, \ldots P^k\}$ that represent the surfaces $S^0, \ldots S^k$ can be understood of as a discretization of Equation 4.10 both in space and in time (or scale).

The definition of a discrete multi-scale representation implies that each sample point $\mathbf{p}_i \in P$ is active on all levels. The polygon $\mathbf{p}_i^0, \ldots, \mathbf{p}_i^k$ defines the normal trajectory of $\mathbf{p}_i$, since each polygon edge $\mathbf{p}_i^l \mathbf{p}_i^{l+1}$ is aligned to the normal vector $\mathbf{n}_i^l$. At each level l the point set P^l defines the model surface S^l at the corresponding scale, with level 0 being the smoothest approximation. Note that no subsampling operator is

applied to the lower levels, since unambiguous reconstruction of the normal trajectory requires that each intermediate point $\mathbf{p}_i^j$ is present.

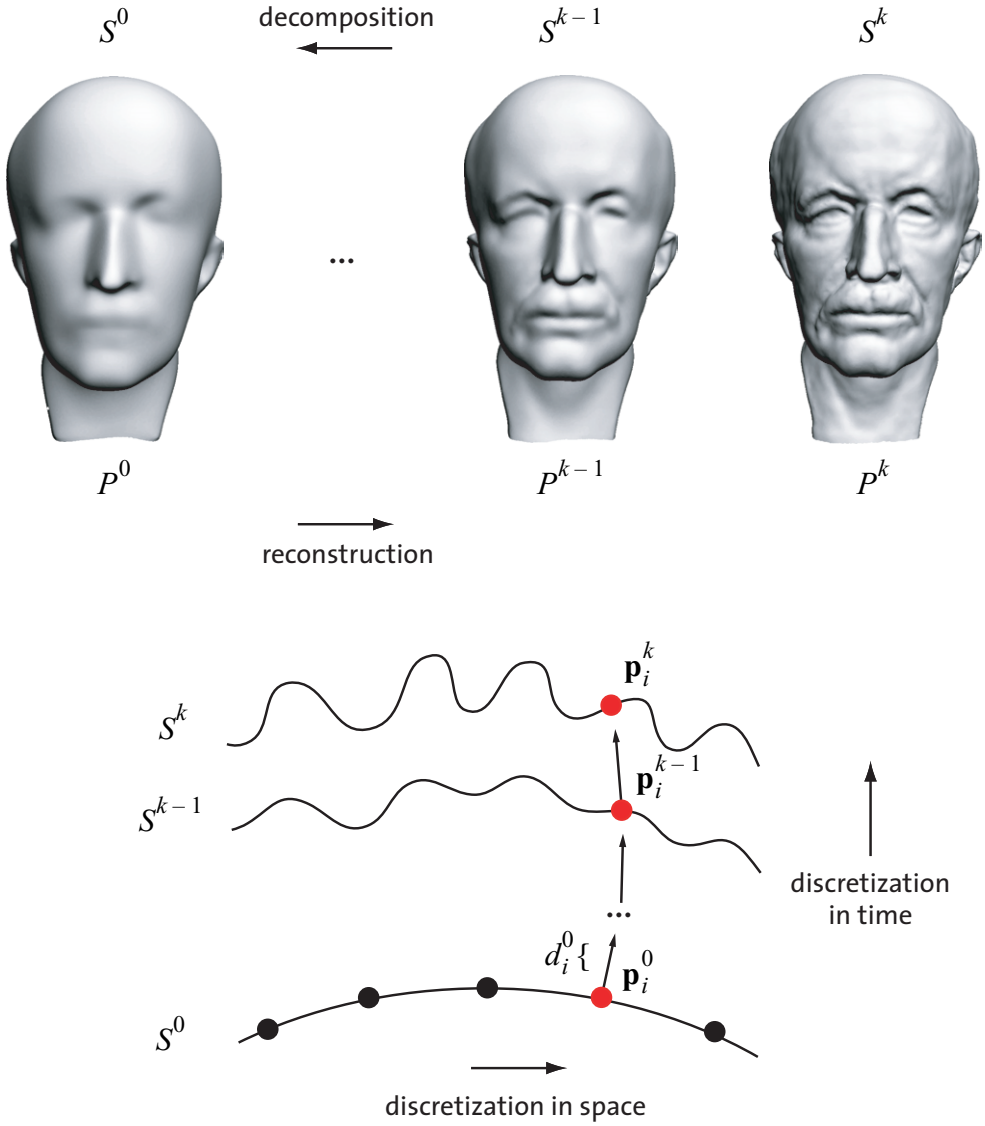


Figure 4.3 Discrete multi-scale representation. Top row: 3D surface model, bottom row: 2D illustration.

4.3.1 Encoding

Suppose a discrete scale-space approximation $\mathbf{Q} = \{Q^0, \ldots, Q^k\}$ of a surface S has been computed using geometric low-pass filtering as described in Section 4.2. To encode two subsequent levels Q^{l-1} and Q^l, the detail coefficients D^{l-1} need to be computed. In general, however, it is not possible to find for each $\mathbf{q}_i^l \in Q^l$ a corresponding sample $\mathbf{q}_i^{l-1} \in Q^{l-1}$ such that the displacement is in normal direction, i.e.,

$$(\mathbf{q}_i^{l-1} - \mathbf{q}_i^l) \cdot \mathbf{n}_i^{l-1} = 0\,.$$

Thus to obtain a discrete multi-scale representation $\mathbf{P} = \{P^0, \ldots, P^k\}$ for the surface S, the surfaces represented by the Q^l need to be re-sampled to fulfill the normal displacement criterion. There are two alternatives to compute the sequence $\mathbf{D}$ of sets of detail coefficients and the sequence $\mathbf{P}$ from the discrete scale-space approximation $\mathbf{Q}$. Suppose two subsequent levels S^{l-1} and S^l are given, represented by two point clouds Q^{l-1} and Q^l, respectively. The goal is to find two point clouds P^{l-1} and P^l approximating the surfaces S^{l-1} and S^l, and a set of detail coefficients D^{l-1} such that $P^l = P^{l-1} + D^{l-1}$.

Bottom-up Encoding by Ray-Shooting

One approach is to shoot a ray from each $\mathbf{q}_i^{l-1} \in Q^{l-1}$ in normal direction and find the intersection point $\mathbf{r}_i^l \in S^l$ with the surface S^l. Then the corresponding detail coefficient is given as

$$d_i^{l-1} = \left\| \mathbf{r}_i^l - \mathbf{q}_i^{l-1} \right\|. \tag{4.18}$$

Furthermore,

$$P^{l-1} = Q^{l-1} \text{ and} \tag{4.19}$$

$$P^l = \{ \mathbf{r}_i^l \,|\, i = 1, \ldots, n \}. \tag{4.20}$$

This means that the point cloud Q^{l-1} representing the smooth surface S^{l-1} is left unchanged, while the point cloud Q^l representing the detailed surface is re-sampled to fulfill the normal displacement condition.

This method of ray-shooting has been applied successfully in previous mesh-based approaches, e.g., in [48] to build a normal mesh hierarchy. An algorithm for intersecting a ray with an MLS surface has been introduced in [2], based on the MLS projection operator. In this method a point on the ray is iteratively projected onto the surface until it converges to a point both on the ray and the surface.

Top-down Encoding by Projection

The second alternative would be to start with a point $\mathbf{q}_i^l \in Q^l$ and orthogonally project it onto the surface S^{l-1} Thus a point $\mathbf{r}_i^{l-1} = \Psi_{Q^{l-1}}(\mathbf{q}_i^l) \in S^{l-1}$ is obtained, where $\Psi_{Q^{l-1}}$ is the MLS projection operator with respect to the point cloud Q^{l-1}. This yields

$$d_i^{l-1} = \left\| \mathbf{q}_i^l - \mathbf{r}_i^{l-1} \right\|, \tag{4.21}$$

$$P^{l-1} = \{ \mathbf{r}_i^{l-1} \,|\, i = 1, \ldots, n \} \text{ and} \tag{4.22}$$

$$P^l = Q^l. \tag{4.23}$$

Here the smooth surface is re-sampled, while the detailed point cloud remains unaltered. This approach can easily be implemented using the MLS projection method as described in Section 2.3.

Discussion

Figure 4.4 illustrates the encoding of two subsequent levels. In this thesis only the top-down approach is used for the following two reasons:

Figure 4.4 Multi-scale encoding: The point cloud P^l is obtained as a normal displacement of the point cloud P^{l-1}. The right image shows the corresponding detail coefficients, where blue indicates maximum negative displacement and red maximum positive displacement for outward pointing normals.

- First, it can be implemented with a single MLS projection per point and does not require an iteration of projections. This is due to the fact that the MLS projection with linear bases is orthogonal to the surface it projects onto (see Section 2.3). Thus top-down encoding is significantly faster than the bottom-up approach.
- Second, since the surface S^{l-1} is a smooth approximation of the surface S^l, re-sampling S^{l-1} will lead to fewer sampling artefacts, such as aliasing, as compared to re-sampling of S^l. The explanation for this can be found in an analogy with Fourier sampling: When re-sampling a continuous function f with a given regular sampling grid, the function has to be band-limited to ensure exact reconstruction without sampling artefacts (see also [117]). More precisely, all frequencies above the Nyquist limit of the sampling grid need to be zero in the Fourier spectrum of the function f (see [13] for more details). For discrete manifold surfaces, no such precise sampling theory has yet been formulated. Nevertheless, since S^{l-1} is a low-pass filtered version of S^l, it contains less high-frequency components and thus better fulfills a "surface Nyquist criterion".

The complete construction of the multi-scale representation starts with the original point cloud $P^k = Q^k$ and encodes it with respect to the point cloud Q^{k-1}. This yields a re-sampled point cloud P^{k-1}, which is itself encoded with respect to the next smoother level Q^{k-2}. This process is iterated as illustrated in Figure 4.5.

The computational effort is quite significant since the MLS projection operator has to be applied $n \cdot k$ times, where $n = |P|$ is the number of sample points in the model. However, once a multi-scale representation is build, reconstruction of individual levels is very efficient, since it only requires the points to be displaced in the normal direction according to the detail coefficients.

As mentioned above, the multi-scale point cloud representation is a discrete sample of the continuous representation both in space and in scale. In this context, the low-pass filter determines the *sampling in the scale dimension*, by controlling the rate of smoothness between two subsequent levels. The decomposition algorithm, on the other

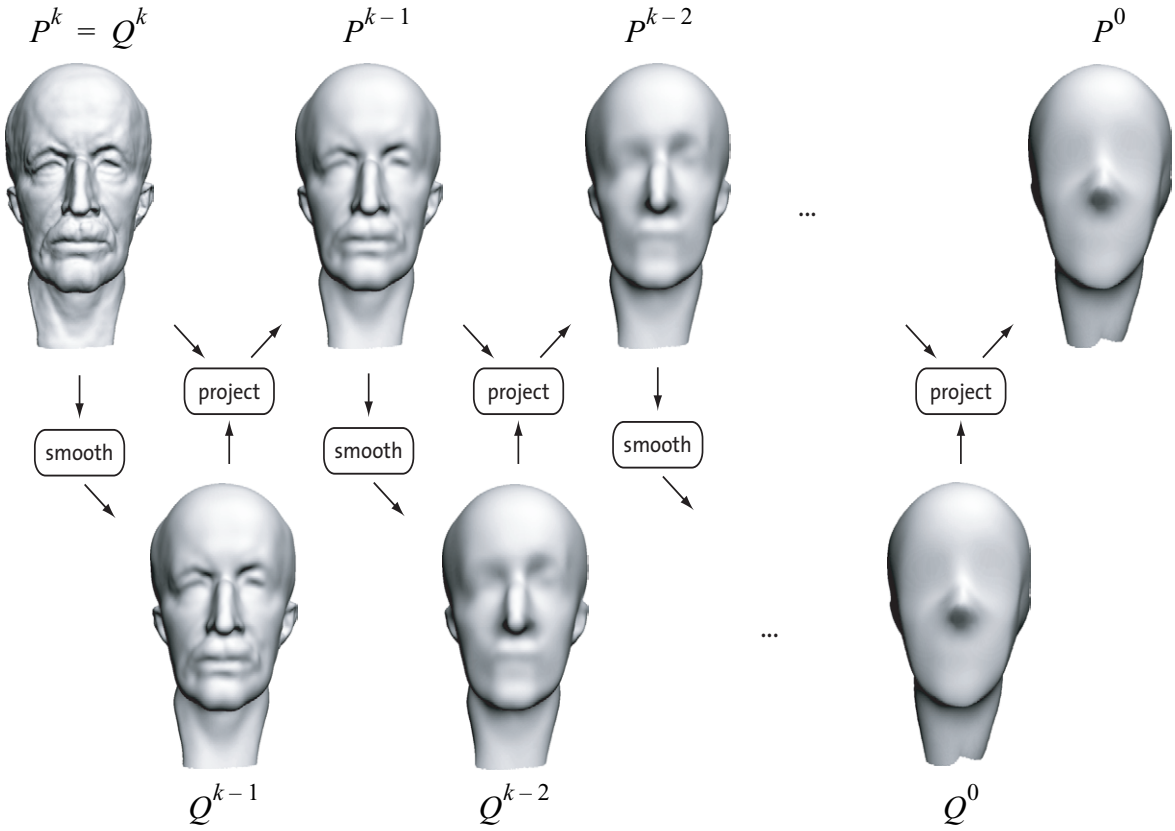


Figure 4.5 Building a discrete multi-scale representation.

hand, needs to find the base points on the smoother level to define the normal displacements and thus determines the *sampling in the spatial dimension*.

4.3.2 Continuous Representation

Note that even though the sequence of point clouds $\mathbf{P} = \{P^0, \ldots, P^k\}$ defines a discrete sample along the scale axis, a continuous scale-space approximation can be obtained by interpolation. For example, a linear blend between two successive levels P^{l-1} and P^l can be defined as

$$P^l(\alpha) = P^{l-1} + \alpha \cdot D^{l-1}, \tag{4.24}$$

where $\alpha \in [0, 1]$ is the blending parameter, $P^l(0) = P^{l-1}$, and $P^l(1) = P^l$. This corresponds to a linear blend between each individual sample described as

$$\mathbf{p}_i^l(\alpha) = \mathbf{p}_i^{l-1} + \alpha \cdot d_i^{l-1} \cdot \mathbf{n}_i^{l-1}. \tag{4.25}$$

Figure 4.6 illustrates a linear blend between two subsequent levels.

4.3.3 MLS Filtering

In Section 2.3 it has been observed that the MLS projection operator implements a low-pass filter, whose filter characteristics are determined by the kernel width h of the Gaussian weight function (see also Figure 2.7). This suggests an alternative smoothing method that creates smooth approximations of a given point set P by simply projecting all points of P onto the MLS surface defined by P. By adjusting the MLS kernel width

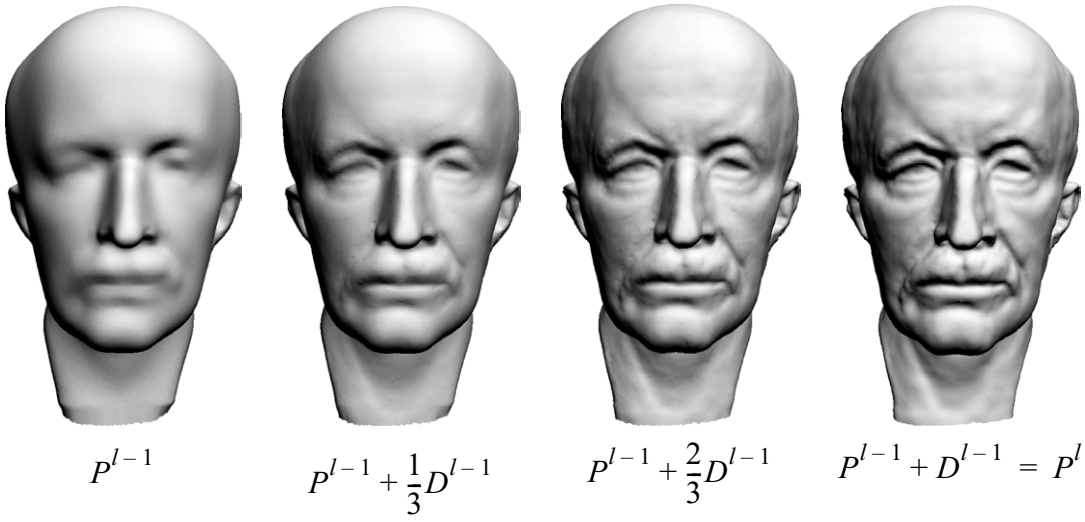


Figure 4.6 Linear blend between two subsequent levels P^{l-1} and P^{l} of a multi-scale representation.

h, different degrees of smoothness can be obtained. In fact, $h = h_t$ can be understood as a continuous scale parameter in the sense of the scale-space representation described above. This means that the MLS projection replaces the fairing operator for creating the scale-space approximation of a given surface S. Since the same projection is used for detail encoding, low-pass filtering and decomposition can be combined into a single projection operation.

Decimation

One difficulty with using a large kernel width in the MLS projection is that large subsets of the point cloud have to considered in the least-squares optimization. As described in Section 2.3 this quickly leads to excessive computation times. A substantial improvement in performance can be obtained by integrating a decimation operator. Since large kernels imply that each individual point contributes less to the optimization, clusters of points can be replaced by a single point without significant loss of accuracy. Thus by first decimating the point cloud using one of the simplification techniques introduced in Chapter 3, the MLS projection operator can be evaluated much more efficiently (see Figure 4.7).

Note that even though a decimation is applied, the projected point clouds $P^{k-1}, \ldots, P^{0}$ still have the same number of sample points as the original point cloud P^{k}. The decimation only affects the intermediate point clouds $Q^{k-1}, \ldots, Q^{0}$ that define the base surface of the MLS projection operator.

A crucial question remains: What is the appropriate decimation rate for a certain kernel width? The qualitative argument given above indicates that the larger the kernel width, i.e., the smoother the resulting surface should be, the higher can the decimation rate be without losing too much in accuracy. However, since no rigorous sampling theory has been defined for discrete manifolds yet, a precise quantitative relation is difficult to formulate.

Thus a different approach is chosen that controls the degree of smoothness using the sampling resolution. The kernel width is coupled to the local sampling density, e.g., as a constant times the local sample spacing as defined in Section 2.1.2. Thus instead of

specifying a discrete set of kernel widths, the discrete scale-space approximation is defined by a set of sampling resolutions $\{n^k, \ldots, n^0\}$ such that $n^k = n = |P^k|$ and $|Q^l| = n^l$ and $n^l > n^{l-1}$ for $1 \le l \le k$.

A suitable choice for the n^l is a geometric series, i.e., $n^{l-1} = \lfloor \gamma \cdot n^l \rfloor$, where $\gamma \in (0, 1)$ is the decimation factor. This approach defines a logarithmic number of discrete scale-space levels in the number of input samples n and corresponds to pyramid algorithms used in multi-resolution methods [47].

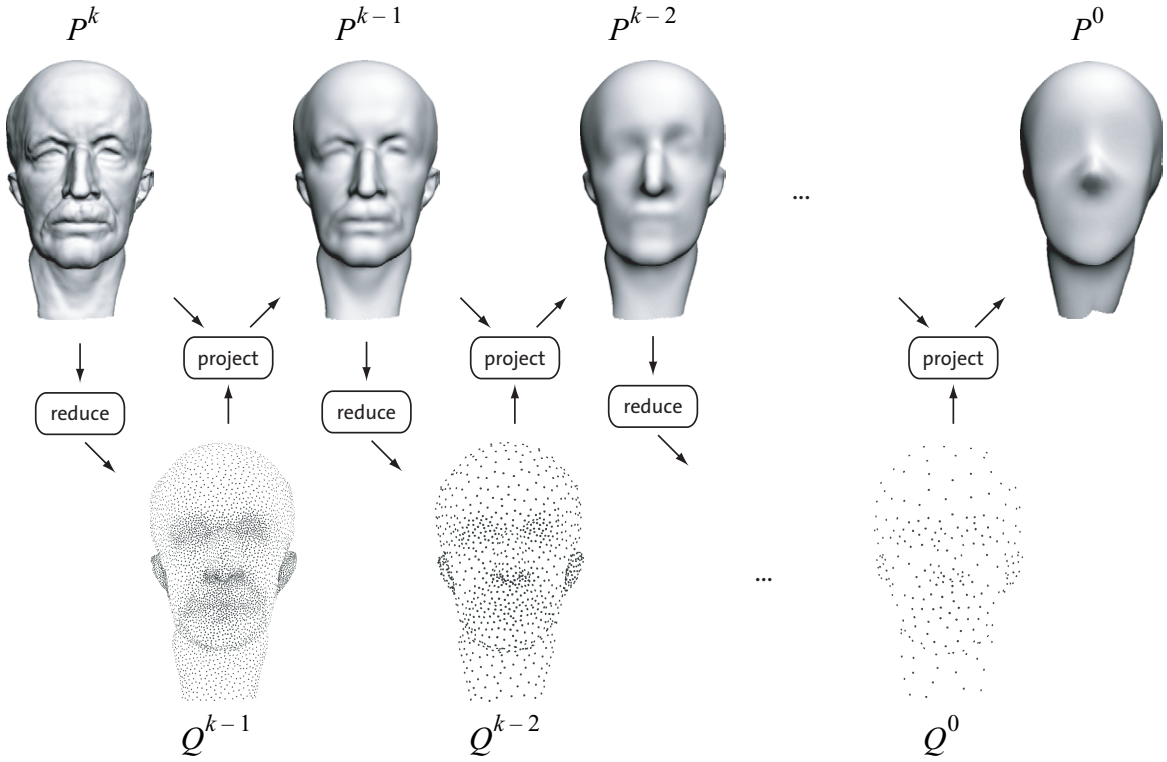


Figure 4.7 Building a multi-scale representation by MLS filtering and decimation. The projection step integrates low-pass filtering and decomposition. The reduction step improves the performance of the projection.

4.4 SPECTRAL FILTERING

The previous section already hinted at a connection between the discrete multi-scale representation and the discrete Fourier transform. This section elaborates on these ideas and defines various spectral filters that can be used for surface editing. First a brief review of the discrete Fourier transform shows that spectral coefficients can be computed without applying the actual transformation by using a suitable band-pass filter. This concept is then transferred to the multi-scale representation to illustrate that the detail coefficients D^l can be considered as discrete spectral bands.

4.4.1 Discrete Fourier Transform

Assume a discrete sample $\{f_i\}, i = 0, \ldots, N-1$ is given. The 1D discrete Fourier transform of $\{f_i\}$ is defined as $\{F_j\}, j = 0, \ldots, N-1$, where

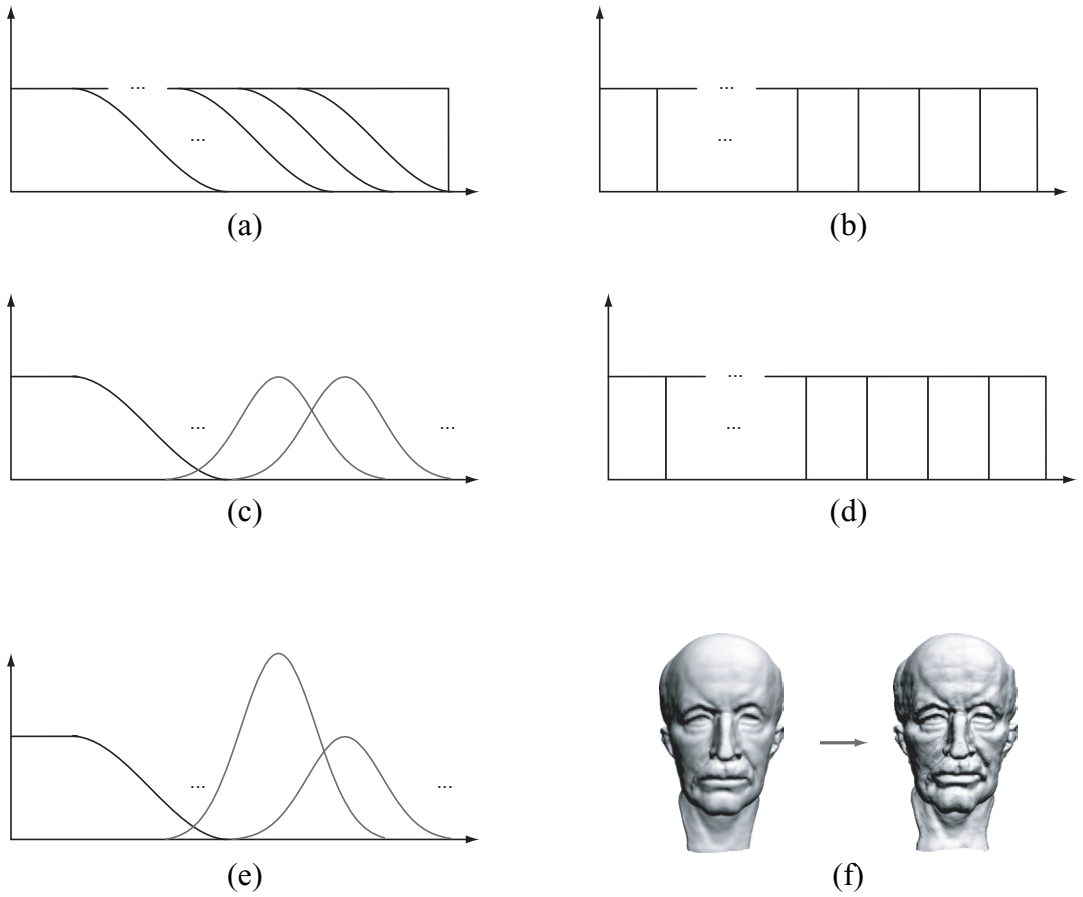


Figure 4.8 Illustration of spectral decomposition. (a) and (c) transfer functions and resulting spectral bands for Gaussian filtering, (b) and (d), transfer functions and resulting spectral bands for ideal low-pass filtering, (e) scaling of a spectral band leads to the enhancement effect shown in (f).

$$F_j = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} f_i \cdot e^{\frac{-2\pi i j \sqrt{-1}}{N}}, \qquad j = 0, \ldots, N-1. \tag{4.26}$$

The inverse discrete Fourier transform is defined as

$$f_i = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} F_j \cdot e^{\frac{2\pi i j \sqrt{-1}}{N}}, \qquad i = 0, \ldots, N-1. \tag{4.27}$$

Let H_k be a discrete filter that transforms a signal $\{f_i\}$ into $\{H_k(f_i)\}$. Such a filter is called an ideal low-pass filter with cut-off frequency k if

$$F_j^{H_k} = \begin{cases} F_j & j < k \\ 0 & j \geq k \end{cases}, \tag{4.28}$$

where $F_j^{H_k}$ is the discrete Fourier transform of the signal $\{H_k(f_i)\}$. Similarly, an ideal band-pass filter can be defined as $H_{k,l} = H_l - H_k$, where $l > k$. Given such a filter, a

spectral coefficient F_j of a signal $\{f_i\}$ can be obtained from the filtered signal $\{H_{j,j+1}(f_i)\}$ using Equation 4.27 yielding

$$H_{j,j+1}(f_i) = \frac{1}{\sqrt{N}} F_j \cdot e^{\frac{2\pi i j \sqrt{-1}}{N}} \qquad \forall i \in [0, N-1]. \tag{4.29}$$

For $i = 0$ this leads to $F_j = H_{j,j+1}(f_0) \cdot \sqrt{N}$. Note that the separation into distinct Fourier coefficients is only possible if an ideal band-pass filter $H_{j,j+1}$ can be defined and the basis functions of the spectral transform are known.

4.4.2 Multi-Scale Surface Filtering

A multi-scale representation $\mathbf{P} = \{P^0, \ldots, P^k\}$ is build by successively applying low-pass filters with wider filter kernels. Since the detail coefficients D^l encode the difference between to subsequent levels, they can be understood as the result of a band-pass filter applied to the original point cloud P^k. Thus the detail coefficients can be considered as the spatial representation of a spectral band. However, since the low-pass filters are not ideal, the spectral bands are overlapping and cannot be used to determine the coefficients of a spectral transform. Also, unlike the Fourier transform, the spectral basis functions are not explicitly defined and can only be approximated as the eigenfucnctions of the Laplacian operator [43].

Figure 4.8 illustrates the spectral decomposition of the multi-scale surface representation. (a) and (b) show the filter transfer functions for Gaussian smoothing and ideal low-pass filtering, respectively. While the latter leads to a sharp separation into discrete frequency bands as shown in (d), Gaussian filtering creates overlapping spectral bands as shown in (c). Nevertheless this decomposition can still be used for filtering, as illustrated in (e), where one of the detail coefficients was scaled by a factor of two, leading to an enhancement effect as shown in (f).

Figure 4.9 shows various spectral filters that modify the surface geometry by scaling the detail coefficients. These filtering effects would be very difficult to achieve without the multi-scale decomposition. One problem that does not occur in the functional setting is that the modifications of the detail coefficients can lead to self-intersections, which would corrupt the model surface.

The multi-scale surface representation defines a decomposition of a point-sampled surface into discrete frequency bands. Even though many analogies to Fourier analysis exists, and many concepts of Fourier theory serve as a motivation for the presented techniques, this representation should not be understood as an extension of the Fourier transform to discrete surfaces. At most, the multi-scale approach allows to implement various filtering methods that mimic the characteristics of Fourier-based spectral filters. The main benefit of the multi-scale representation in the context of this thesis is the explicit representation of surface detail, which can be used to implement sophisticated editing operations.

As indicated above, the multi-level decomposition operator introduced in Section 4.3 does not require any correspondence of the sampling patterns between to subsequent levels as long as the two surfaces are geometrically close. This can be exploited for advanced filtering applications that use entirely different models to create a decomposition. Figure 4.10 shows such an example where the Max Planck model has been encoded onto a sphere. First the point cloud has been globally re-scaled to roughly

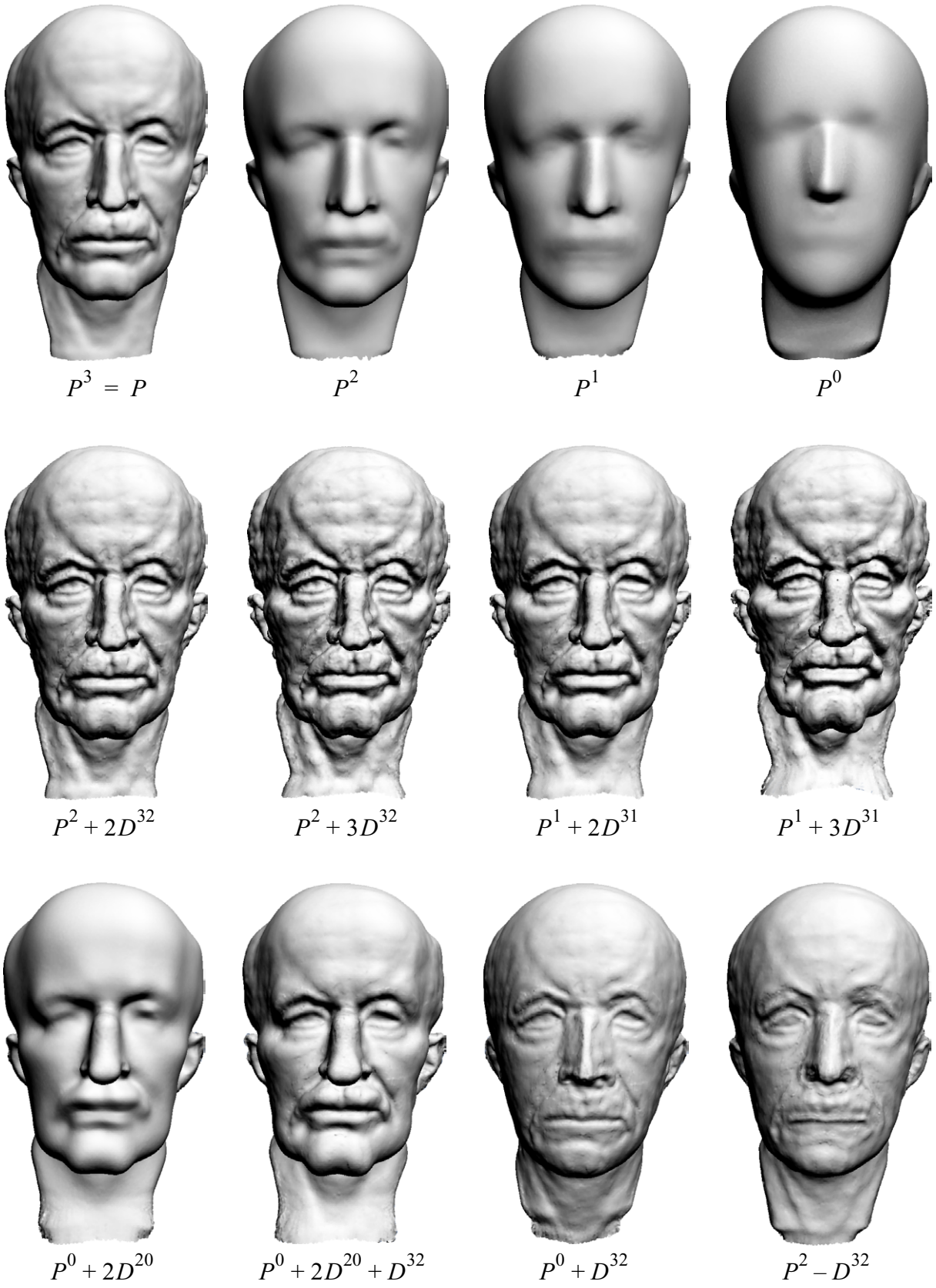


Figure 4.9 Spectral filtering. The top row shows a discrete multi-scale decomposition of the Max Planck model. The second row illustrates different enhancement filters, where D^{ij} denotes the difference between surface S_i and S_j. The bottom row shows band-pass and band stop filters.

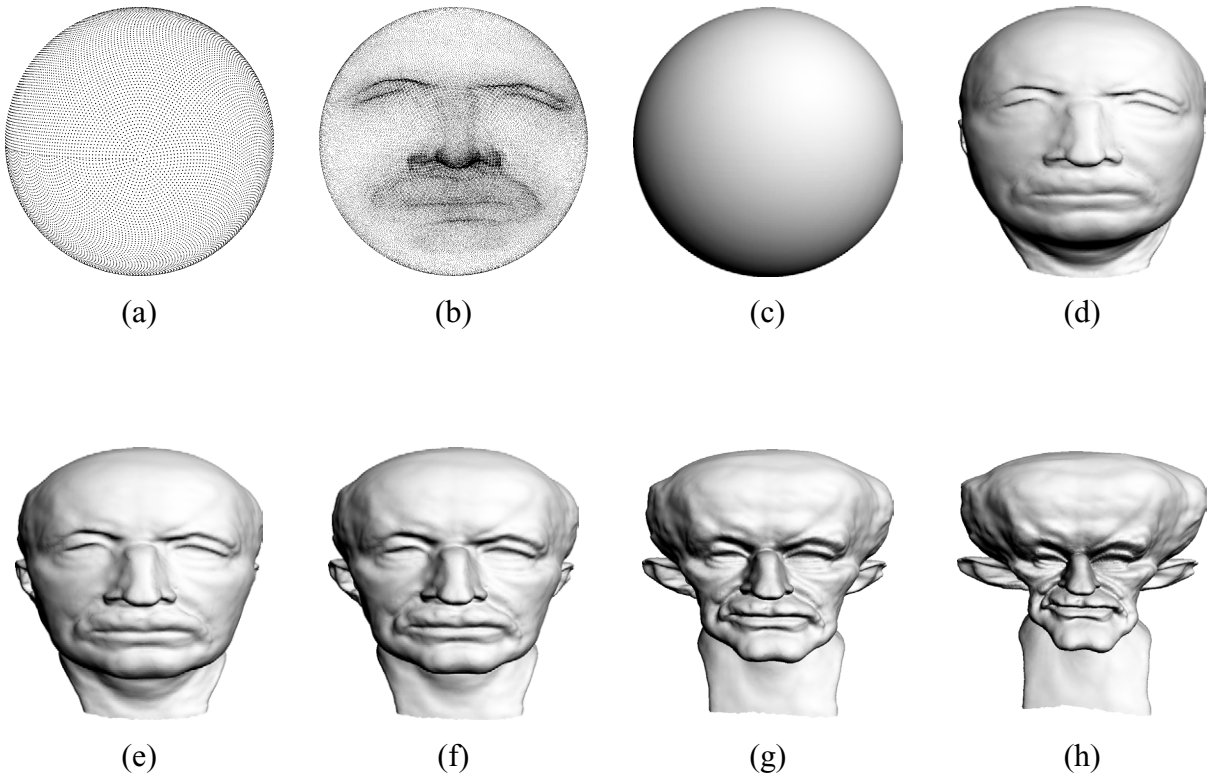


Figure 4.10 Encoding the Max Planck model onto a sphere. (a) sampling distribution of the sphere, (b) sampling distribution of the Max Planck model projected onto the sphere of (a), (c) - (h): reconstructed models with detail coefficients scaled to 0% (c), 50% (d), 75% (e), 100% (f), 150% (g), and 200% (h).

match the shape of the sphere. Then each sample point has been projected onto the sphere to obtain the re-sampled base surface and corresponding detail coefficients. Scaling the later leads to different filtering effects, as illustrated in Figure 4.10. For scaling factors between zero and one, a blend between sphere and Max Planck model can be obtained. Scaling factors larger than one lead to enhancement effects similar to Figure 4.9.

Morphing

This scheme can also be used to implement simple morphing operations (see also [3] and [70] for more general morphing methods). Assume that two surfaces S_1 and S_2 represented by two point clouds P_1 and P_2 are given. The goal is to find a point cloud P_M that describe a linear blend between the two. For this purpose a common base surface S_B defined by P_B has to be found such that S_B is geometrically close to both S_1 and S_2. In a first step, P_1 and P_2 are projected onto P_B yielding two point clouds P_1' and P_2'. To implement a morphing operation between the two models, a one-to-one correspondence of the sample points needs to be defined. This can be achieved by interpolating the detail coefficients of P_2' onto the sampling distribution of P_1', or vice versa. A linear blend of the two surfaces can then be obtained using

$$\mathbf{p}_i(\alpha, \beta) = \mathbf{p}_i + (\alpha \cdot d_i^1 + \beta \cdot d_i^2) \cdot \mathbf{n}_i, \tag{4.30}$$

where $\alpha, \beta \in [0, 1]$ with $\alpha + \beta = 1$, $\mathbf{p}_i \in P_1'$, $\mathbf{n}_i$ is the surface normal at $\mathbf{p}_i$, d_i^1 is the detail coefficient of model P_1', and d_i^2 is the interpolated detail coefficient of model P_2'. For interpolation, a simple scheme based on k-nearest neighbors is used, as described in more detail in Section 6.2.3.

Note that this method is only guaranteed to work correctly if both input models are already geometrically close and at least one of the two surfaces can be defined by a functional mapping from the common base domain. Figure 4.11 shows an example of a morph between the Igea and Max Planck models, where both surfaces have first been scaled to a common bounding box. Then the models have been aligned manually to create a rough correspondence of surface feature points. The base domain has been chosen as the maximal ellipsoid that fits into the common bounding box. As the bottom right images illustrate, the scaling factors are not restricted to a convex combination, but can be chosen arbitrarily to enable more flexible blending operations.

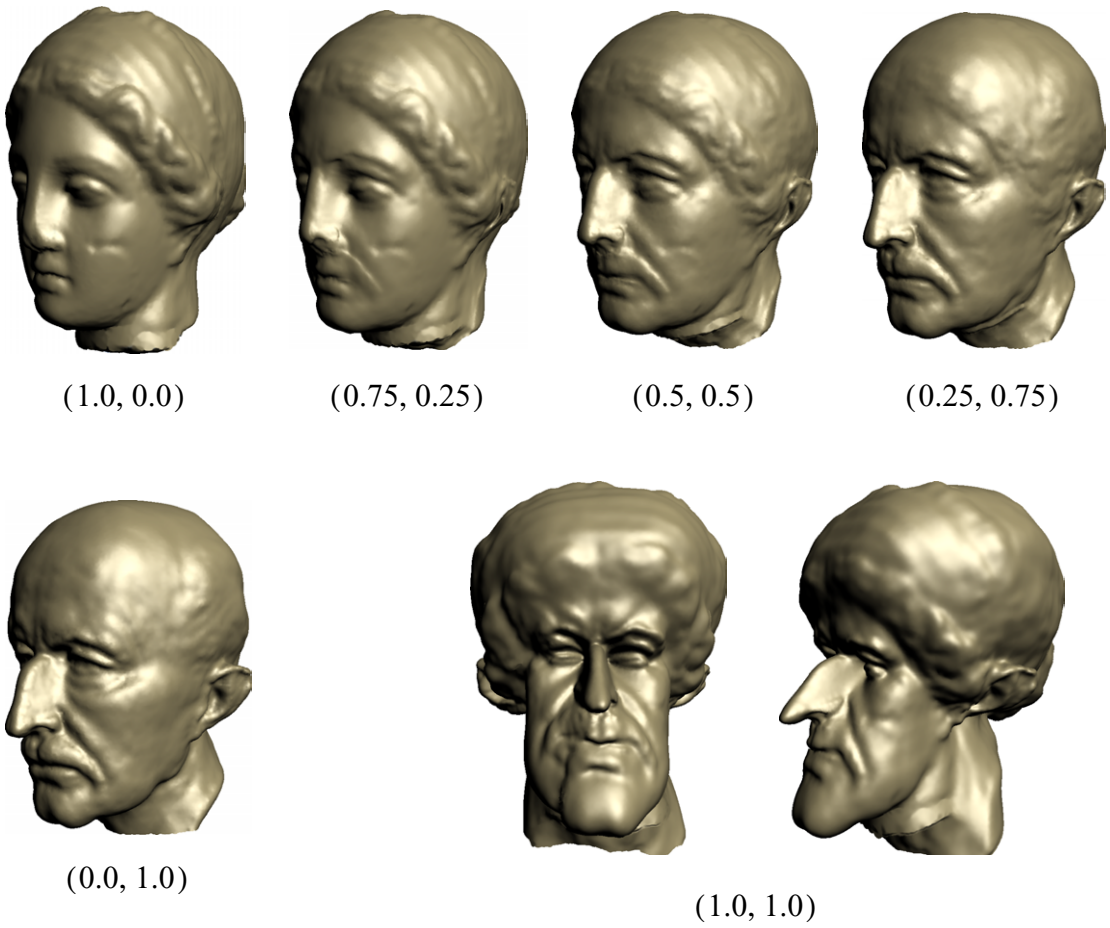


Figure 4.11 Morphing between the Igea and Max Planck models. The numbers in parentheses are the α and β scaling factors of Equation 4.30.

4.5 MULTI-SCALE DEFORMATION

Apart from the filtering methods presented in the previous section, the multi-scale surface representation also offers advanced modeling semantics for free-form deformation (see also Section 6.2). Figure 4.12 illustrates the difference between

single-scale and multi-scale deformation. The surface shown in (b) has been encoded with respect to the surface depicted in (a), i.e., the embossed text defines the surface detail. A smooth deformation of (b) yields the surface shown in (c) and (e). Observe that as a consequence of the deformation, the embossed text has been sheared and is no longer orthogonal to the surface base plane. A more intuitive result can be obtained by applying the deformation to the smooth surface of (a) and subsequently adding the detail using normal displacements, as described in Section 4.3. The resulting surface is shown in (d) and (f). Here the embossing is still in normal direction, which yields a more natural deformation as compared to the single-scale operation.

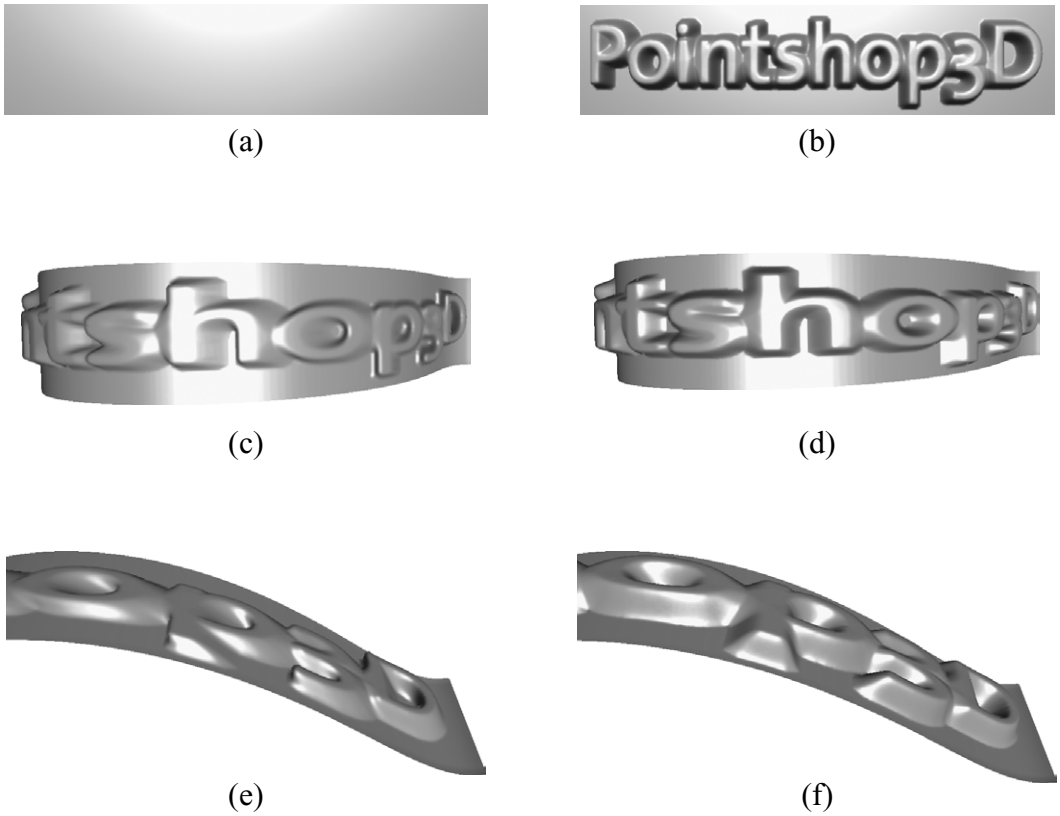


Figure 4.12 Multi-scale vs. single-scale modeling. (a) base surface, (b) surface with detail, (c) and (e) single scale deformation of (b), (d) and (f) multi-scale deformation of (a) with subsequent normal displacement.

Figure 4.13 shows the same effect for a topologically more complex surface of genus 16. The surface in (a) has been deformed in (d) and (f) by displacing each sample point into the normal direction of the base plane, using a smooth deformation function as explained in Section 6.2. For multi-scale deformation, the surface of (a) has first been encoded onto the plane shown in (b). This plane has then been deformed with the same deformation function used in the single-scale deformation, yielding the deformed base surface of (c). Finally, the sample points have been displaced using the detail coefficients obtained in the encoding stage. Observe how for the multi-scale deformation the blue handles are still orthogonal to the deformed surface, whereas for single-scale deformation no such preservation of angles is achieved.

This example also illustrates that detailed surface and base surface are not required to have the same genus. This is a distinct feature for point-sampled surfaces, since no consistency constraints are imposed on the sample positions, i.e., the neighborhood relations of the detailed surface need not be consistent with the neighborhood relations on the base surface.

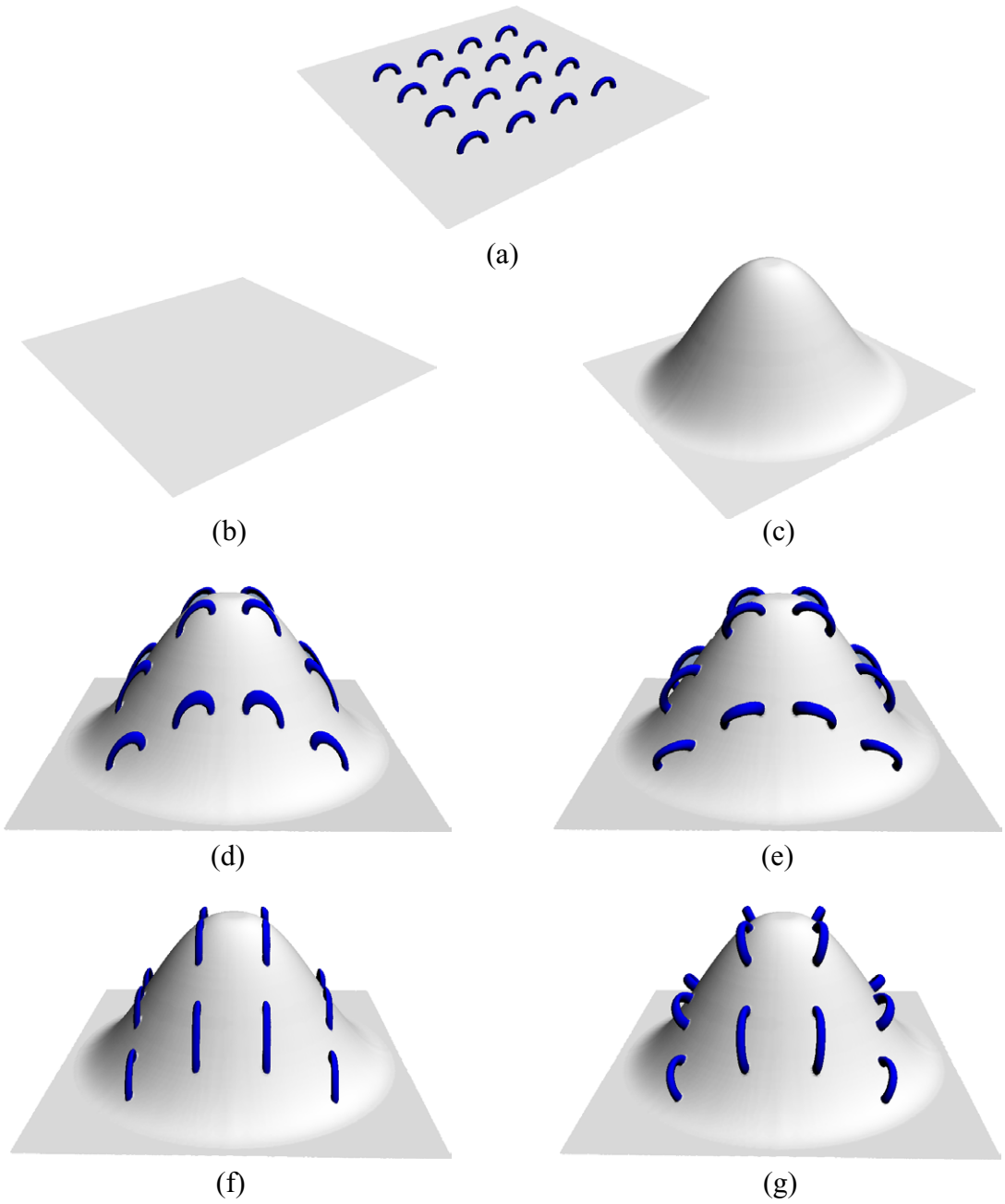
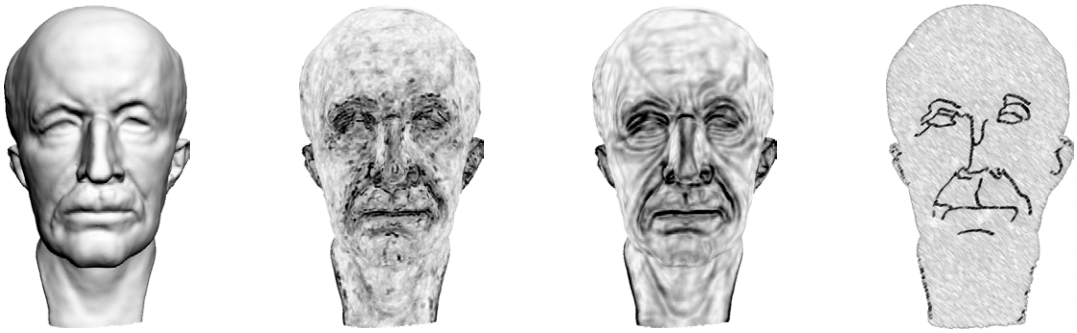


Figure 4.13 Multi-scale vs. single-scale modeling. (a) original surface, (b) base domain, (c) deformed base surface, (d) and (f) single-scale deformation, (e) and (g) multi-scale deformation.

5

FEATURE EXTRACTION



In this chapter a new method for detecting and extracting line-type features on point-sampled surfaces is presented [87]. This type of information can serve as input for many geometry processing applications such as meshing, model segmentation, or anisotropic fairing [19]. Feature lines can also be used for visualization to enhance the semantics of renditions of 3D objects. Section 5.5 will show how artistic line drawings of point-sampled surfaces can be created using the extracted feature curves.

Features are usually defined as entities of an object that are considered important by a human for an accurate description of the object. This definition is highly subjective, however, and very difficult to express in algorithmic form.

The goal of this work is to design a feature extraction algorithm that requires no additional semantic information about the object. Also, the method should be semi-automatic, i.e., only require the user to specify a few thresholding parameters. Additional interaction with the object, such as setting seed points or guiding feature movement, is not necessary. Therefore, the definition of a feature is based on low-level information using a statistical operator that measures local surface variation as introduced in Section 2.1.3. This operator classifies points according to the likelihood that they belong to a feature. To improve the robustness and reliability of the classification stage, the operator is applied at multiple scales (see also Chapter 4),

which allows to measure the persistence of a feature [30]. Additionally, multi-scale classification provides further structural information per classified point, e.g., the characteristic scale at which a feature is most prominent.

This work concentrates on line-type features. These are probably the most important features for surfaces, which are often composed of patches that are framed by feature lines. A feature line approximately passes along a ridge of maximum inflection, which is adequately captured in the surface variation estimate.

Since the feature definition relies solely on low-level surface information, some user feedback is required, in particular for the example application of an artistic line-drawing renderer. To obtain visually pleasing renditions, the user has to adjust the various parameters of the feature extraction pipeline until she is satisfied with the result. Hence particular emphasis is put on efficiency, allowing interactive control in a low-latency feedback loop.

5.1 PREVIOUS WORK

Feature extraction is a well-studied research area in many scientific fields, including computer vision, medical imaging and computational fluid dynamics. Most of the past research efforts concentrated on data defined in a Euclidean domain, e.g., images, volume data, or flow fields. Feature extraction on surfaces, i.e., 2-manifolds embedded in 3-space, has gained less attention, but is important in many fields such as range data analysis or reverse engineering. In these applications, the input data is typically a large point set acquired by some 3D scanning device. Thus feature extraction directly on the point cloud is very attractive, as it can be used to support early processing steps such as surface reconstruction or adaptive decimation.

The feature extraction method presented here combines and extends existing techniques from different research fields, integrating recent results from image processing, discrete geometric modeling and scale-space theory.

Canny [15] introduced an optimal filter for step-edge detection in images. He found that there exists a natural uncertainty principle between detection and localization performance and derives operators that are optimal at any scale. His method of hysteresis thresholding will be used during feature classification (see Section 5.4.1).

Hubeli and Gross [57] introduced a multi-resolution framework for feature extraction on triangle meshes. Based on various classification operators they identify a set of feature edges and use thinning to extract line-type features from the set of selected edges.

Feature sensitive meshing of surfaces defined as the zero-set of a discrete 3D distance function has been presented in [65]. Here features are detected using the width of the normal cone spanned by adjacent vertices as a measure of surface curvature. The focus in this work is on avoiding the aliasing artefacts generated by the standard marching cubes algorithm and the authors report that their simple feature classification method yields good results for this purpose.

Geometric snakes have been used in [72] to extract feature lines in triangle meshes based on normal variation of adjacent triangles. This system requires the user to specify an initial feature curve, which is then evolved under internal and external forces and re-projected onto the surface using a local parameterization.

Gumhold et al. [46] presented a feature extraction method for point clouds that is similar to the method introduced here. They also use covariance analysis for classification and compute a minimum spanning graph of the resulting feature nodes. Their scheme is extended in this work by using a multi-scale classification that allows robust feature extraction for noisy surfaces. Additionally, more control on the smoothness of the extracted feature lines can be achieved by modeling the feature lines using snakes, as compared to the spline fitting method used in [46].

5.2 OVERVIEW

Figure 5.1 gives an overview of the feature extraction pipeline. Given an unstructured point cloud P that approximates a surface S, the algorithm starts by classifying points according to the likelihood that they belong to a feature (Section 5.3). This is done using a multi-scale approach that assigns weights ω_i to each $\mathbf{p}_i \in P$. After thresholding these weights, a minimum spanning graph of the remaining sample points is computed (Sections 5.4.1 and 5.4.2). Each separate component of the graph is modeled by a snake, an energy-minimizing spline that is attracted to the feature vertices. The snakes can be smoothed using Euler integration, while maintaining a close connection to the underlying surface (Section 5.4.3). The extracted feature lines can then be visualized using non-photorealistic point-based rendering (Section 5.5).

5.3 FEATURE CLASSIFICATION

The first stage of the feature extraction pipeline is classification. For each point $\mathbf{p}_i \in P$ a weight ω_i is computed that measures the confidence that $\mathbf{p}_i$ belongs to a feature. Feature classification is based on surface variation estimation using covariance analysis of local neighborhoods (see Section 2.1.3). This statistical approach can be incorporated into a scale-space framework that allows feature classification at multiple scales.

Given the multi-scale representation introduced in Chapter 4, feature classification can be performed at multiple scales by applying a suitable classification operator, e.g., curvature estimation, on each of the discrete surface approximations. For large models, the computational overhead quickly becomes excessive, however, since the curvature estimation has to be applied for each sample point at each scale. An alternative approach uses the surface variation estimate based on local neighborhoods, as defined in Section 2.1.1.

5.3.1 Multi-scale Variation Estimation

The concepts of scale-space (see Section 4.1) can be applied for feature classification using surface variation $\sigma_n(\mathbf{p})$ based on the observation that the size n of the neighborhood of a sample $\mathbf{p}$ can be understood as a discrete scale parameter (see also Figure 2.3). In fact, increasing the size of the local neighborhood is similar to applying a smoothing filter. This becomes intuitively clear when looking at the way the covariance matrix is defined as sums of squared distances from the neighborhood's centroid. If the neighborhood size is increased, each individual point contributes less to the surface variation estimate. Hence high-frequency oscillations are attenuated, analogous to standard low-pass filter behavior.

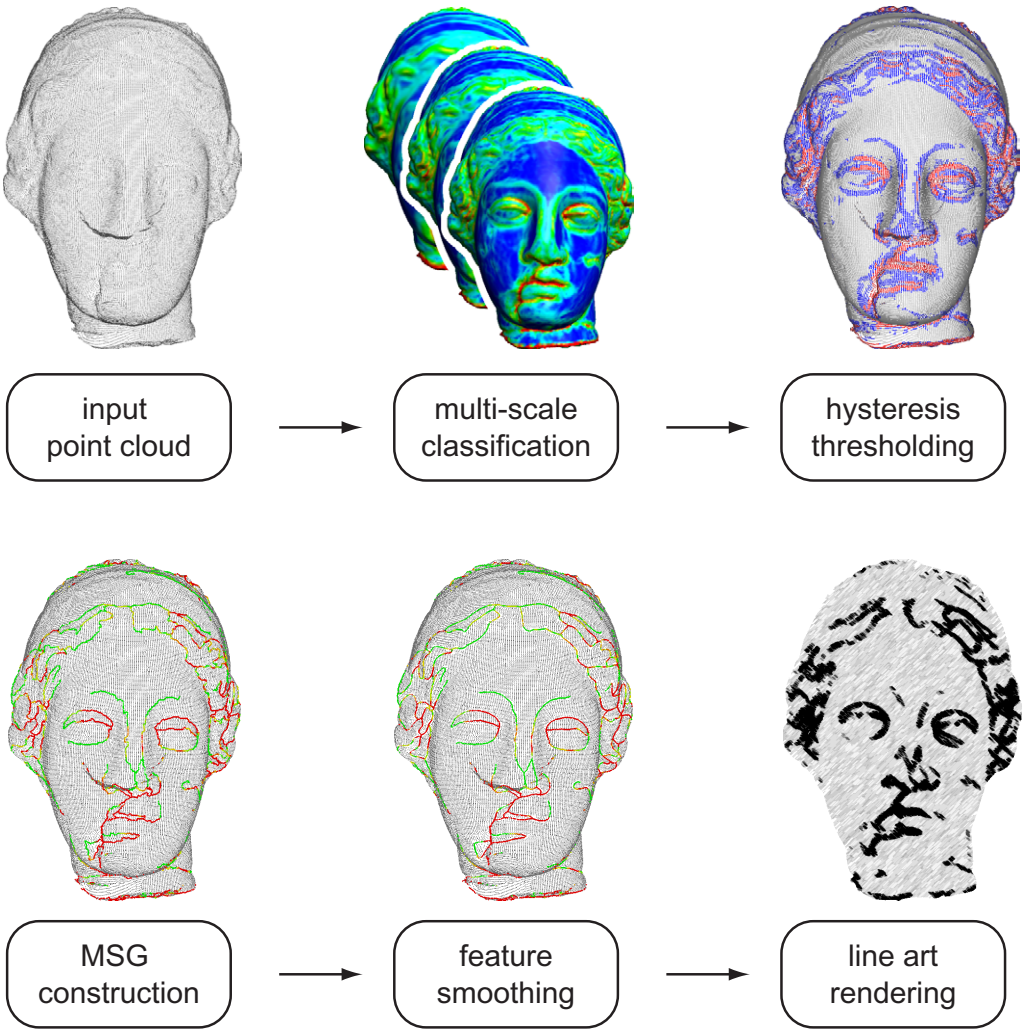


Figure 5.1 Feature extraction pipeline.

Comparison to Gaussian Smoothing

To evaluate the multi-scale variation estimation, the above method is compared to the traditional multi-scale approach using Gaussian filter kernels (see Section 4.1). For this purpose a terrain model is used that is defined as a regularly sampled height-field. Since this surface can be parameterized without distortion, coarse scale representations can be computed using standard Gaussian filtering for grids. As illustrated in Figure 5.2, the classification on the smoothed surfaces using surface variation of smaller neighborhood sizes corresponds very well to the output of the variation estimate on the rougher surfaces with bigger neighborhood sizes. Even though some quantitative deviations are observable, in terms of feature classification both methods are almost equivalent and thus interchangeable.

Efficient Computation

This section presents an incremental method for computing the surface variation $\sigma_n(\mathbf{p})$ at a point $\mathbf{p}$ for increasing neighborhood size n (see also Section 2.1.3). Assume a neighborhood N_p has already been computed with mean $\bar{\mathbf{p}}$, covariance matrix $\mathbf{C}$ and corresponding eigenvalues $\lambda_0 \leq \lambda_1 \leq \lambda_2$. Now the neighborhood size is

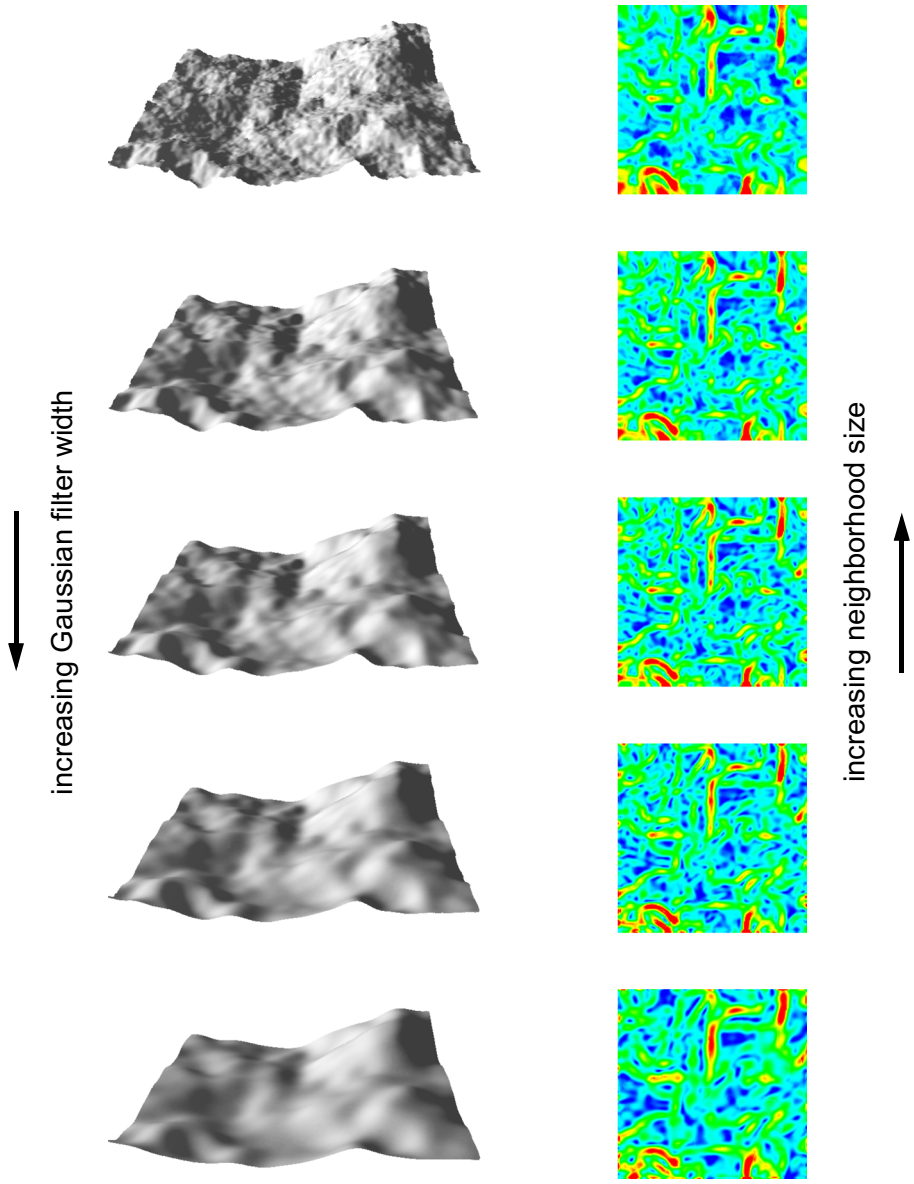


Figure 5.2 Multi-scale surface variation on height field data. Left column: Scale-space representation of a terrain model with increasing smoothness from top to bottom. Right column: Corresponding surface variation with increasing neighborhood size from bottom to top.

increased by one, i.e., the next closest point $\mathbf{q}$ to N_p is included. The new mean $\bar{\mathbf{p}}'$ can be obtained as

$$\bar{\mathbf{p}}' = (n\bar{\mathbf{p}} + \mathbf{q})/(n+1) \tag{5.1}$$

and the new covariance matrix $\mathbf{C}'$ as

$$\mathbf{C}' = \frac{n}{n+1}\left(\mathbf{C} + \frac{\mathbf{q}'\mathbf{q}'^T}{n+1}\right), \tag{5.2}$$

where $\mathbf{q}' = \mathbf{q} - \bar{\mathbf{p}}$ [49]. The eigenvalues can be computed as the roots of the characteristic polynomial, i.e., by solving

$$\Gamma(\lambda) = |\mathbf{C} - \lambda\mathbf{I}| = \lambda^3 + p\lambda^2 + q\lambda + r = 0\,. \tag{5.3}$$

The explicit formula for analytically computing the roots of a cubic polynomial uses trigonometric functions. A more efficient method exploits coherence of the local neighborhood: Since only the smallest eigenvalue λ_0' of $\mathbf{C}'$ is of interest (note that $-p = \lambda_0 + \lambda_1 + \lambda_2$, cf. Equation 2.16) a Newton iteration can be used to find λ_0'. Since the characteristic polynomial $\Gamma(\lambda)$ changes only slightly when adding another sample point to the neighborhood of $\mathbf{p}$, λ_0 provides a very good initial guess for λ_0' (see Figure 5.3). Due to the quadratic convergence of the Newton scheme, typically less than 3 iterations are sufficient.

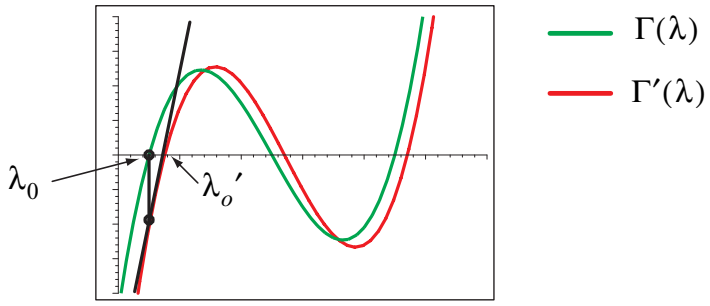


Figure 5.3 Exploiting coherence when computing surface variation for increasing neighborhood sizes using Newton's method.

5.3.2 Determining Feature Weights

Given a multi-scale variation estimate, the user can specify the appropriate scale of interest and use the variation estimate of that scale as the feature weights ω_i. Thus by selecting a single parameter, the scale, the user can decide whether fine-scale or coarse-scale features should be extracted.

Automatic Scale Selection

However, finding the right scale parameter is often difficult and this is why methods for automatic scale selection have been of interest in many fields. Lindeberg pioneered these techniques for functional scale-space representations [79]. His principle for scale selection states that the scale level at which some normalized derivative operator assumes a local maximum reflects a characteristic length of the corresponding structure in the data. As illustrated in Figures 5.4 and 5.5 this principle can easily be transferred to the scale-space representation introduced above. To determine the feature weights, the strongest local maximum in the surface variation at all points across the scale axis is determined. The points on the ear, nose and leg in Figure 5.5, for example, have been classified as feature points because they exhibit a distinct local maximum in surface variation, while the point on the back shows no such characteristic.

Persistence

Instead of using a single local maximum for classification, one can also look at the number of times that the surface variation exceeds a certain threshold $\sigma_{\max}$. For a point $\mathbf{p}_i \in P$ and a neighborhood size n define

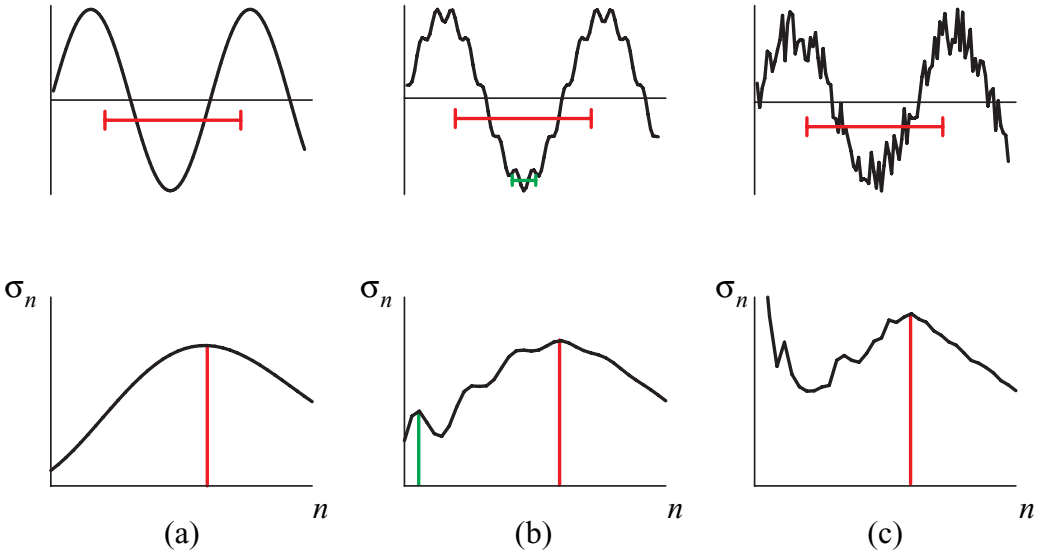


Figure 5.4 Automatic scale selection for a 1D signal. The top row shows the signal, the bottom row the variation at the central point as a function of neighborhood size. The local maxima are indicated as vertical lines and the characteristic lengths as horizontal bars. In (a) a simple sine curve is analyzed, while in (b) another high-frequency sine wave has been added. Note how the different frequencies are reflected in the distinct local maxima of the variation estimate. In (c) random noise has been added to the signal. When looking at the coarser scales, i.e., larger n, the feature can still be recovered faithfully.

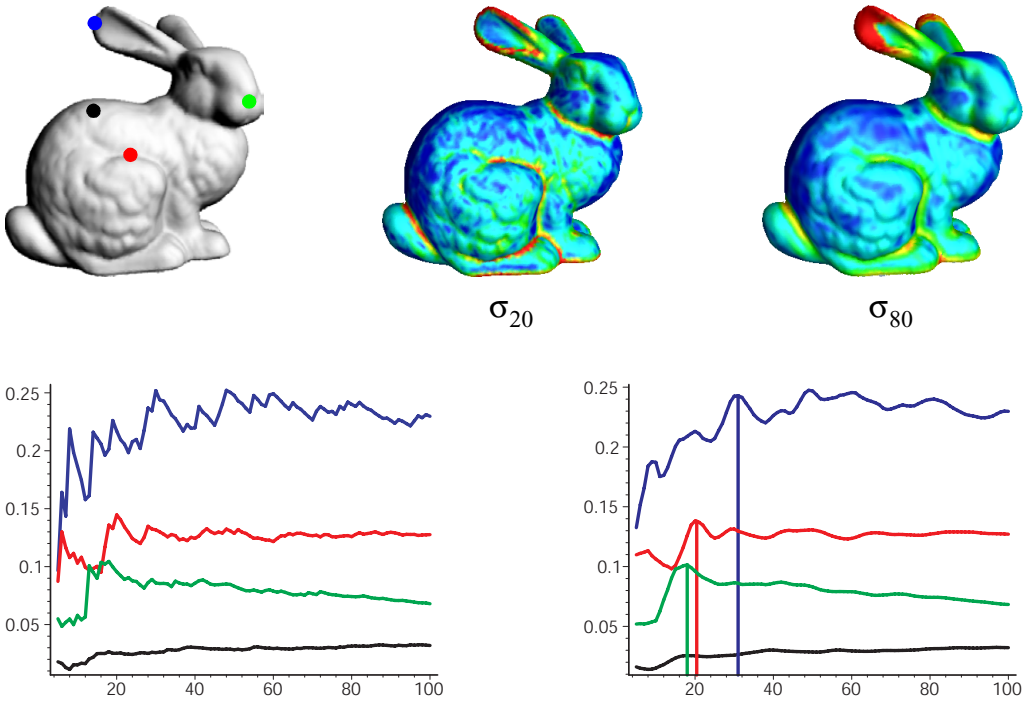


Figure 5.5 Multi-scale surface variation. The left diagram shows values of σ_n for different points on the bunny as a function of neighborhood size n. To avoid instabilities in the detection of local maxima, these curves have been pre-smoothed as shown on the right. The vertical lines show the scale of the extracted maxima in surface variation.

$$\Omega(\mathbf{p}_i, n) = \begin{cases} 1 & \sigma_n(\mathbf{p}_i) > \sigma_{\max} \\ 0 & \sigma_n(\mathbf{p}_i) \le \sigma_{\max} \end{cases}, \tag{5.4}$$

so that the corresponding feature weight ω_i is given as

$$\omega_i = \sum_n \Omega(\mathbf{p}_i, n). \tag{5.5}$$

Thus this counting approach measures the *persistence* of a feature over all scales (see also [30]).

Surface Boundaries

When dealing with non-closed surfaces, the feature classification method is extended to include points that lie on the surface boundary. These points are detected using the method of Linsen et al. [81]. All points of a local neighborhood of size n_b of a point $\mathbf{p}_i \in P$ are projected into the tangent plane and ordered according to angle. Whenever the angular distance between two consecutive points exceeds some maximum angle α_b, $\mathbf{p}_i$ is classified as a boundary point and its feature weight ω_i is set to maximum. The parameters n_b and α_b effectively control the size of the holes that are detected as boundaries.

Neighborhood Size

Increasing the size of the local neighborhood when computing the variation estimate eventually violates the prerequisite that all points of the neighborhood belong to the same connected region of the underlying surface (see Section 2.1.1). This problem can be addressed in two different ways: Either neighborhood relations that are more sophisticated than Euclidean distance are used, e.g., a Riemannian graph or mesh connectivity, if available. Or one can try to estimate when the neighborhood becomes too large and stop the calculations of the variation measure. A simple heuristic that works well in practice is to look for jumps in σ_n, as they indicate strong deviations in the normal direction. Figure 5.6 shows an example for this method for a point on the bunny's ear. When increasing the size of the neighborhood, points from the opposite side of the ear will eventually be included in the set of k-nearest neighbors, as illustrated in Figure 5.6 (a). This critical neighborhood size can be determined by examining the variation-scale curve as shown in Figure 5.6 (b).

5.4 FEATURE RECONSTRUCTION

Feature reconstruction consists of three stages: First a set $Q \subset P$ of *feature nodes* is selected, i.e., points that with high probability belong to a feature. Then a minimum spanning tree (MST) for these feature nodes is computed. After pruning short branches and closing cycles, each component of the resulting graph is modeled as a snake, which allows user-controlled smoothing of the feature lines.

5.4.1 Selecting Feature Nodes

In the classification stage of Section 5.3 weights ω_i have been assigned to each sample point $\mathbf{p}_i \in P$ that measure the confidence that $\mathbf{p}_i$ belongs to a feature. To select the

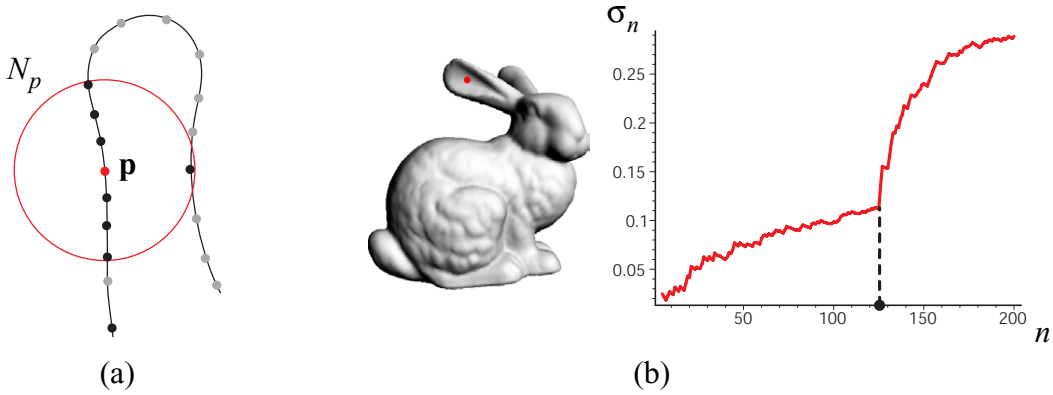


Figure 5.6 (a) Illustration of an invalid neighborhood, (b) the critical neighborhood size for a point on the bunny's ear occurs at a jump in the variation-scale curve.

relevant feature nodes $Q \subset P$, all points whose weights fall below a certain threshold could simply be discarded. This hard thresholding can cause undesirable artefacts, however, such as interrupted or dangling feature lines. As suggested in [15], hysteresis thresholding can alleviate these effects by using two thresholds $\omega_{\min} < \omega_{\max}$. Points with weights smaller than $\omega_{\min}$ are discarded, while points with $\omega_i > \omega_{\max}$ are included into the set of feature nodes Q. All points with $\omega_{\min} \leq \omega_i \leq \omega_{\max}$ will be used to bridge the gaps between feature nodes during the construction of the minimum spanning tree (see below).

5.4.2 Minimum Spanning Graph

To create a set of feature patterns, a minimum spanning tree (MST) of the set of feature nodes Q is computed. In the beginning all feature nodes $\mathbf{q}_i \subset Q$ are sorted according to their weights. Then the feature node $\mathbf{q}$ with biggest weight is chosen as the seed point for the construction of the MST. Each of the k-nearest neighbors $\mathbf{q}_i$ of $\mathbf{q}$ defines an edge with corresponding edge cost

$$c(\mathbf{q}, \mathbf{q}_i) = \frac{1}{\omega_{\mathbf{q}} \omega_{\mathbf{q}_i}} + \gamma \cdot \frac{\|\mathbf{q} - \mathbf{q}_i\|}{d}, \tag{5.6}$$

where d is the length of the diagonal of the bounding box of all points in Q, and γ is an additional parameter that allows to balance feature weights against Euclidean distance. All these edges are put on a heap ordered by increasing cost values. Then the edge with the smallest cost is taken from the heap and added to the MST, if both edge nodes are not already part of the tree. For the new feature node a new set of edges and corresponding cost values is computed and also put on the heap. This process is repeated until the heap is empty. Figure 5.7 (b) shows the MST of the dinosaur head generated with this algorithm.

Pruning and Closing of Cycles

As can be seen in this example, the MST of all feature nodes contains many short branches, which are usually treated as artefacts that do not describe salient features. A bottom-up graph pruning method is applied to eliminate these short branches from the set of feature patterns. The algorithm starts by sorting all leaves of the MST according

to their depth. By traversing the tree upward from the deepest node, the longest path can be determined, which defines a new root of the MST. Now all branches of the tree are computed recursively and each branch is assigned an importance value that is given as the length of the branch multiplied by the product of all edge weights. Thus short branches are retained that contain feature nodes with high confidence values and only those branches are pruned that with low probability are part of a feature line.

The MST construction above does not support cycles in the feature lines. It is often desirable, however, to allow closed loops as these more naturally describe certain feature lines. Therefore, the MST is transformed into a graph by closing cycles that are longer than a user-specified threshold, using those edges whose feature nodes are already in the graph. From both of these nodes the tree is traversed upward until the two paths cross at a common node. The sum of the two path lengths then equals the cycle length. Note that the method for pruning and closing cycles does not require an expensive breadth first search as was done, for example, in [46]. Figure 5.7 (c) shows the MST of Figure 5.7 (b) after pruning and closing cycles.

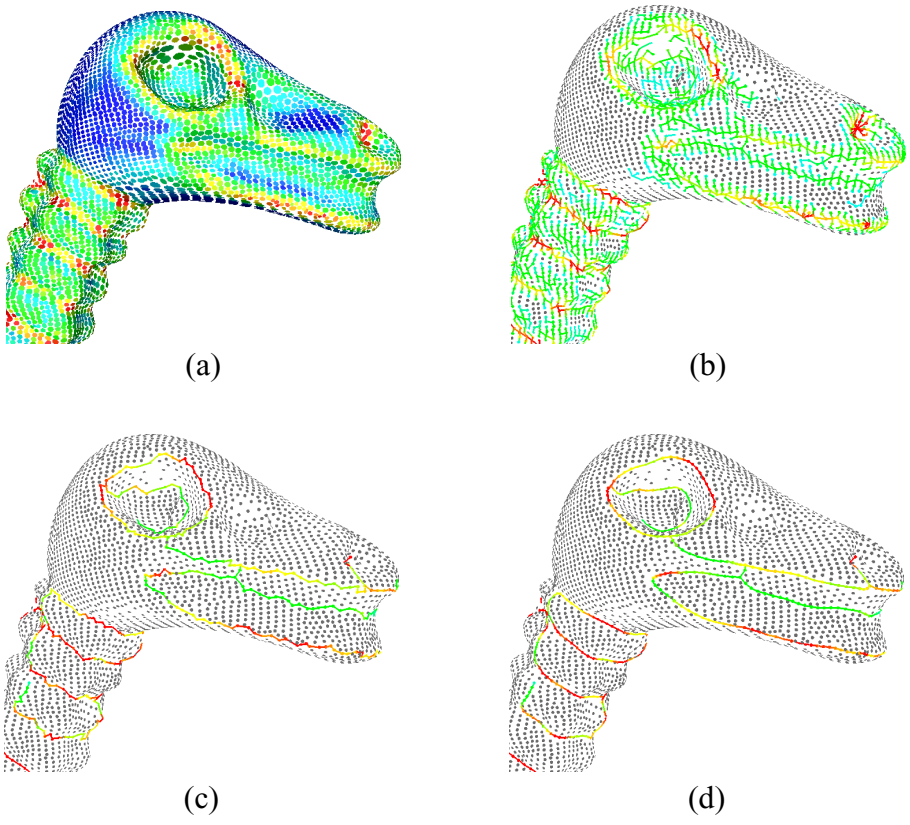


Figure 5.7 Feature reconstruction on the dinosaur head: (a) feature weights, (b) minimum spanning tree of feature nodes, (c) MST after pruning and closing cycles, (d) smoothing with snakes.

5.4.3 Active Contour Models

As can be seen in Figure 5.7 (c), the extracted feature lines connect samples of the original point cloud and are often jagged. This might be acceptable for partitioning algorithms, but for feature-based visualization methods (see Section 5.5) it leads to inferior rendering quality. Such applications require some mechanism for smoothing feature lines. Spline fitting has been used in previous approaches [46], but it provides

little flexibility and insufficient control over the smoothness and accuracy of the extracted feature lines. In [60], Kass et al. introduced *snakes*, active contour models, for detecting features in images. A snake is an energy-minimizing spline that moves under internal and external forces. For point-sampled surfaces, snakes can be used to smooth the feature curves, while maintaining a close contact to the surface. The main benefit of snakes is their explicit control over the degree of smoothness that can be adapted to the specific application needs. Additionally, external constraints can easily be incorporated, for instance to enable user interaction for positioning feature lines.

Energy Minimization

Each component of the MSG is modeled as a parametric curve $\mathbf{v}(s)$ that tries to minimize the energy functional

$$\mathbf{E} = \int \mathbf{E}_{int}(\mathbf{v}(s)) + \mathbf{E}_{ext}(\mathbf{v}(s))ds\,, \tag{5.7}$$

The internal spline energy consists of first- and second-order terms, modeling the behavior of a membrane and a thin plate, respectively:

$$\mathbf{E}_{int}(\mathbf{v}(s)) = \alpha(s) \cdot |\mathbf{v}(s)'|^2 + \beta(s) \cdot |\mathbf{v}(s)''|^2/2\,, \tag{5.8}$$

where $\alpha(s)$ and $\beta(s)$ control the relative weight of the two terms. The external energy is related to the surface variation:

$$\mathbf{E}_{ext}(\mathbf{v}(s)) = 1/\tilde{\sigma}(\mathbf{v}(s))\,, \tag{5.9}$$

where $\tilde{\sigma}(\mathbf{v}(s))$ is computed by interpolating the maximum variation of each point $\mathbf{p} \in P$ at $\mathbf{v}(s)$. Discretization of the functional $\mathbf{E}$ finally leads to a system of Euler equations that is solved using Euler integration (see [60] for details). Figure 5.7 (d) shows the smoothing effect on the dinosaur head.

5.5 NON-PHOTOREALISTIC RENDERING

The extracted feature lines can be used as input for a variety of processing algorithms, including mesh generation, anisotropic fairing and model segmentation. In this section a point rendering system is introduced for creating line drawings of point-sampled surfaces. Based on the surface splatting technique of Zwicker et. al. [119], the renderer takes as input a point model of the original surface and the output of the feature extraction method. Each feature line is converted into a set of feature points by sampling the model surface along the feature line. The sample points of the original model are only rendered into the z-buffer to resolve visibility and are assigned the background color. The final image is then only composed of the visible feature points. Note that no shading computations are applied, which significantly improves rendering performance.

The additional information of the classification stage can be utilized to enhance the semantics of these renditions. The splat radii of the feature points can be scaled according to scale and the intensity (e.g., grey level) adjusted according to the maximum surface variation. Thus features on coarser scales are rendered as thicker lines, while features on fine scales are rendered as thinner lines. Also prominent features are rendered at high intensities, while less significant features are rendered at

low intensity. With these simple extensions, a very intuitive effect is achieved, similar to what a painter would do when drawing an image. Additional screen space filters can be applied to obtain more artistic looking renditions, as illustrated in Figures 5.8 to 5.12.

5.6 EXAMPLES

This section illustrates the effectiveness of the feature extraction pipeline for a number of point-sampled models. Tables 5.1 and 5.2 summarize performance data for the different stages of the pipeline. For multi-scale classification the surface variation estimate σ_n is evaluated for each point $\mathbf{p} \in P$ for all neighborhood sizes n between 15 and 200 using the method introduced in Section 5.3.1. The set of k-nearest neighbors is computed using a kd-tree, as described in Section 8.2.1. Note that the interactive feedback loop for adjusting the various thresholding parameters does not include the multi-scale classification stage, which hence needs to be executed only once.

Model	multi-scale-classification	MST, pruning, closing cycles	snakes (100 Euler steps)
Igea	25.156	1.468	0.203
Cat	1.829	0.062	0.031
Gnome	6.703	0.203	0.047
Dinosaur	10.187	2.484	0.234
Dragon	64.422	8.469	1.125

Table 5.1 Timing of the feature extraction pipeline in seconds on an Intel Pentium IV, 2.8 GHz.

Model	#input points	#feature nodes	#snake points
Igea	134,345	40,509	7,327
Cat	10,000	3,593	798
Gnome	54,659	9,655	1,714
Dinosaur	56,194	45,879	6,395
Dragon	435,545	203,713	31,154

Table 5.2 Complexity of the different stages.

Figure 5.8 shows feature lines extracted on the Igea model. The middle image shows a rendition without the artistic screen space filter.

The cat model of Figure 5.9 is a difficult example, since it exhibits strong local imbalances in the sampling pattern. Still the salient surface features are faithfully recovered.

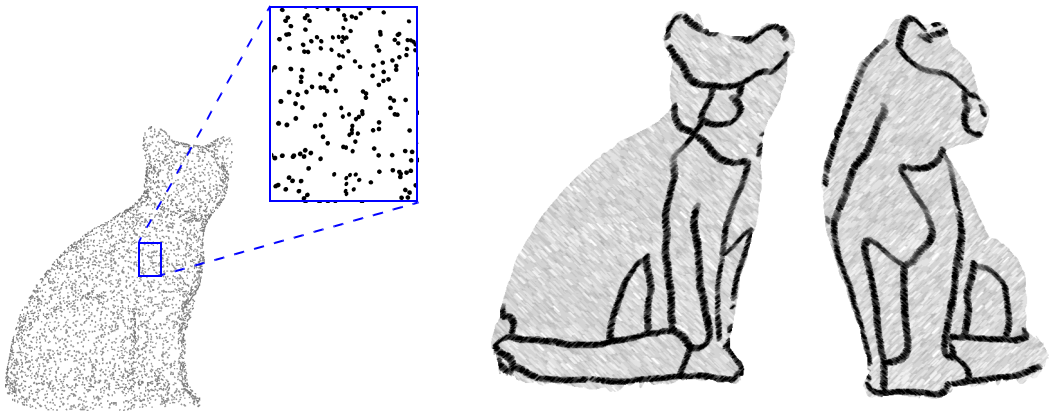
Figure 5.10 shows an example of a noisy laser range scan, which demonstrates that the multi-scale method is superior to single-scale classification. Also note that this is a difficult example for a (semi-) automatic feature extraction algorithm, because humans have a very clear and distinct perception of important feature lines in faces.

The dragon and dinosaur models (Figures 5.11 and 5.12) show that the feature reconstruction method in connection with the point-based rendering method is very suitable for generating artistic line drawings of complex geometric surfaces.

These examples demonstrate that multi-scale feature analysis offers a number of advantages. First it makes the method more robust in the presence of noise. Second it allows coarse-scale features to be extracted even though the curvature might be low. Third it provides additional structural information, which can be used, for example, to adapt the width of feature lines according to the scale.

(a) (b) (c)

Figure 5.8 Feature reconstruction on the Igea model. (a) shaded surface, (b) extracted feature lines, (c) rendition with artistic screen space filter.



(a) (b)

Figure 5.9 As illustrated in (a) the cat model is sampled very non-uniformly. (b) shows the extracted feature lines.

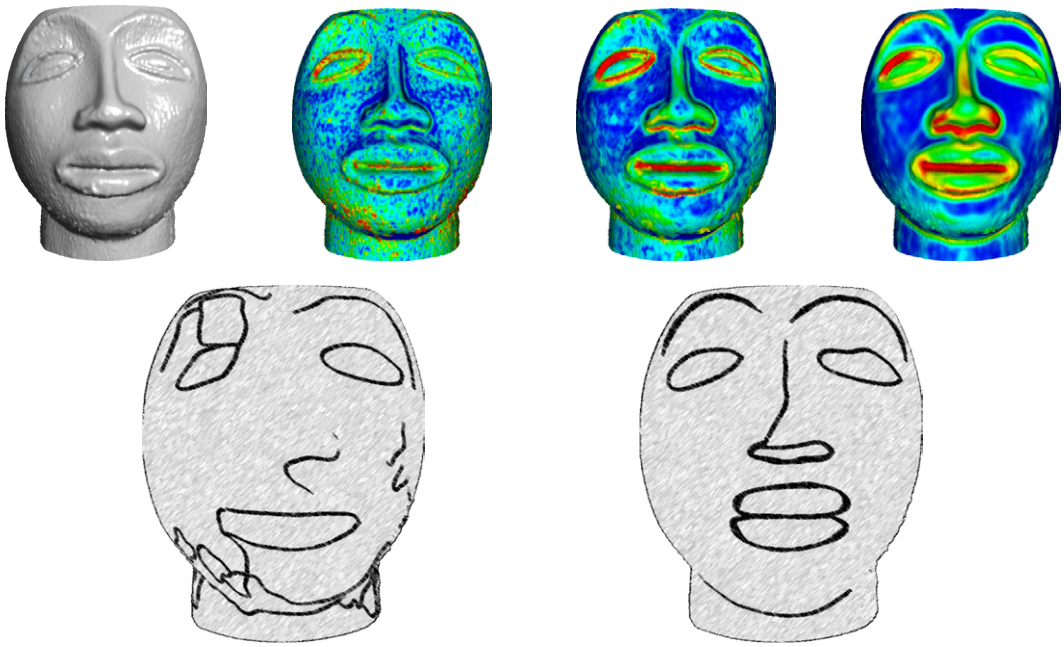


Figure 5.10 Multi-scale feature extraction (bottom right) is superior to single scale extraction (bottom left) on a noisy range scan. The top row shows the original point cloud and variation estimates for different scales.

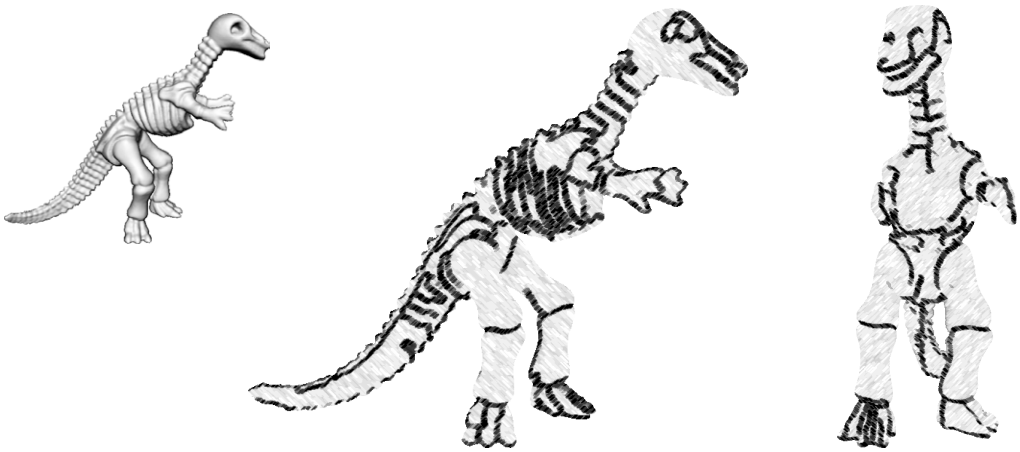
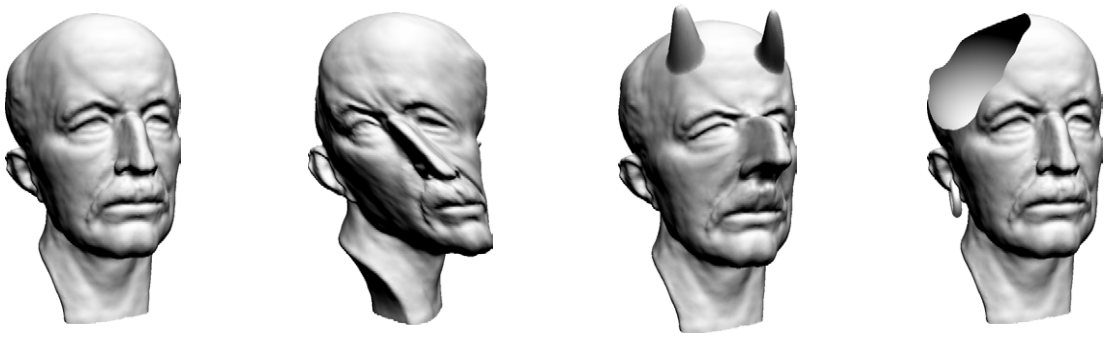


Figure 5.11 Feature extraction on the dinosaur model.

Figure 5.12 Feature extraction on the dragon model.

6

SHAPE MODELING



Modeling the shape of 3D objects is one of the fundamental techniques in digital geometry processing. In this chapter two fundamental modeling approaches for point-sampled geometry will be presented: Boolean operations and free-form deformation [88, 89]. While the former are concerned with building complex objects by combining simpler shapes, the latter defines a continuous deformation field in space to smoothly deform a given surface. As discussed in the introduction, boolean operations are most easily defined on implicit surface representations, since the required inside-outside classification can be directly evaluated on the underlying scalar field. On the other hand, free-form deformation is a very intuitive modeling paradigm for explicit surface representations. For example, mesh vertices or NURBS control points can be directly displaced according to the deformation field.

For point-based representations, the hybrid structure of the surface model defined in Section 2.3.2 can be exploited to integrate these two modeling approaches into a unified shape modeling framework. Boolean operations can utilize the signed distance function defined by the MLS projection (see Section 2.3.3) for inside-outside classification, while free-form deformations operate directly on the point samples.

6.1 BOOLEAN OPERATIONS

A common approach in geometric modeling is to build complex objects by combining simpler shapes using boolean operations [51] (see Figure 6.1). In constructive solid geometry (CSG) objects are defined using a binary tree, where each node corresponds to a union, intersection, or difference operation and each leaf stores a base shape. Operations such as ray-tracing, for example, are then implemented by traversing this tree structure. More commonly, surfaces are defined as boundary representations (B-Reps) of solids. Here boolean operations have to be evaluated explicitly, which requires an algorithm for intersecting two surfaces. Computing such a surface-surface intersection can be quite involved, however, in particular for higher order surfaces (see for example [68]).

As will be demonstrated below, the MLS projection operator (see Section 2.3.2) can be used both for inside/outside classification as well as for explicitly sampling the intersection curve. The goal is to perform a boolean operation on two orientable, closed surfaces S_1 and S_2 that are represented by two point clouds P_1 and P_2, to obtain a new point cloud P_3 that defines the resulting surface S_3. P_3 consists of two subsets $Q_1 \subseteq P_1$ and $Q_2 \subseteq P_2$ plus a set of newly generated sample points that explicitly represent the intersection curves. Thus in order to perform a boolean operation for point-sampled geometry, the following techniques are required:

- a classification predicate to determine the two sets Q_1 and Q_2,
- an algorithm to find samples on the intersection curve, and
- a rendering method that allows to display crisp features curves using point primitives.

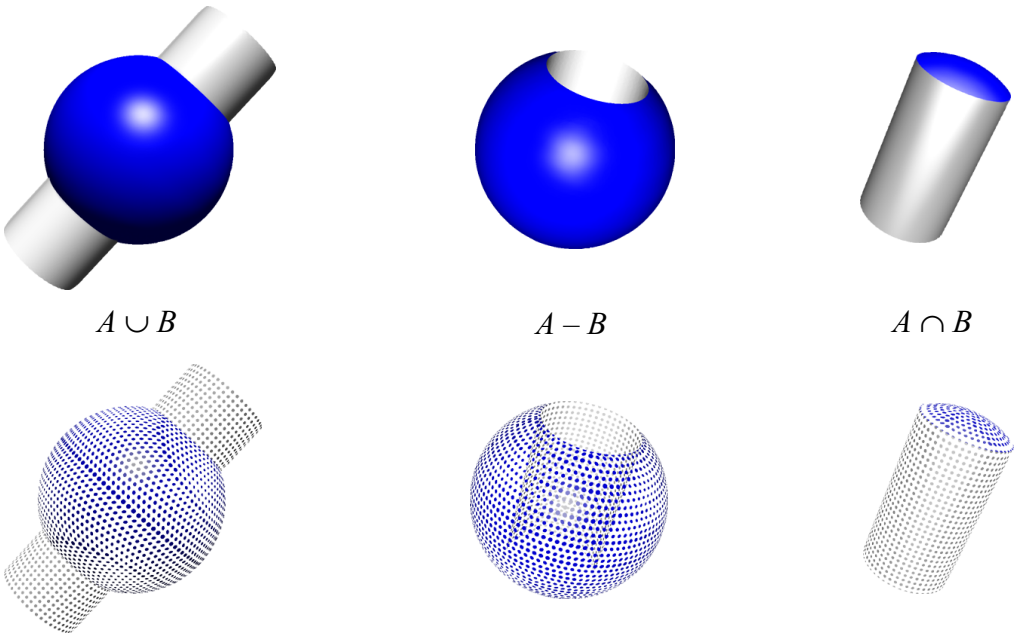


Figure 6.1 Boolean operations of a sphere A and a cylinder B. The bottom row illustrates the sampling distribution.

6.1.1 Classification

The goal of the classification stage is to determine which $\mathbf{p} \in P_1$ are inside or outside the volume enclosed by the surface S_2 and vice versa. For this purpose a classification predicate Ω_P is defined such that for $\mathbf{x} \in \mathbb{R}^3$

$$\Omega_P(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \in V \\ 0 & \mathbf{x} \notin V \end{cases} \tag{6.1}$$

where V is the volume bounded by the MLS surface S represented by the point cloud P. Let $\mathbf{y} \in S$ be the closest point on S from $\mathbf{x}$. It is well-known from differential geometry that, if S is continuous and twice differentiable, the vector $\mathbf{x} - \mathbf{y}$ is aligned with the surface normal $\mathbf{n}_y$ at $\mathbf{y}$ [25]. If surface normals are consistently oriented to point outwards of the surface, then $(\mathbf{x} - \mathbf{y}) \cdot \mathbf{n}_y > 0$ if and only if $\mathbf{x} \notin V$. Since only a discrete sample P of the surface S is given, the closest point $\mathbf{y}$ on S is replaced by the closest point $\mathbf{p} \in P$. Thus $\mathbf{x}$ is classified as outside if $(\mathbf{x} - \mathbf{p}) \cdot \mathbf{n}_p > 0$, i.e., if the angle between $\mathbf{x} - \mathbf{p}$ and the normal $\mathbf{n}_p$ at $\mathbf{p}$ is less than $\pi/2$ (see Figure 6.2, left image). This discrete test yields the correct inside/outside classification of the point $\mathbf{x}$ if the distance $\|\mathbf{x} - \mathbf{p}\|$ is larger than the local sample spacing η_p (see Section 2.1.2) at $\mathbf{p}$. If $\mathbf{x}$ is extremely close to the surface, the classification could fail, as illustrated in the right image of Figure 6.2. In this case the exact closest point $\mathbf{y} \in S$ is computed using the MLS projection (Section 2.3).

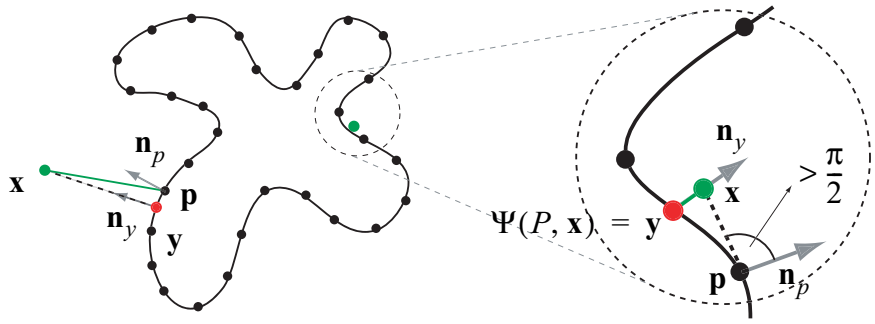


Figure 6.2 Inside/outside test. For $\mathbf{x}$ very close to the surface, the closest point $\mathbf{p} \in P$ can yield a false classification (right image). In this case, $\mathbf{x}$ is classified according to its MLS projection $\mathbf{y}$.

Since for classification only an inside/outside test is of interest, the performance can be significantly improved by exploiting local coherence:

$\Omega_P(\mathbf{x}) = \Omega_P(\mathbf{x}')$ for all points $\mathbf{x}'$ that lie in the sphere around $\mathbf{x}$ with radius $\|\mathbf{x} - \mathbf{p}\| - \eta_p$. Thus the number of closest point queries and MLS projections can be reduced drastically, in practice up to 90 percent.

Given the classification predicate Ω, the subsets Q_1 and Q_2 can be computed as shown in Table 6.1. As Figure 6.3 illustrates, the resulting inside/outside classification is very robust and easily handles complex, non-convex surfaces. Observe that boolean operations can easily create a large number of disconnected components, i.e., can lead to a significant change in genus.

	Q_1	Q_2
$S_1 \cup S_2$	$\{\mathbf{p} \in P_1 \mid \Omega_{P_2}(\mathbf{p}) = 0\}$	$\{\mathbf{p} \in P_2 \mid \Omega_{P_1}(\mathbf{p}) = 0\}$
$S_1 \cap S_2$	$\{\mathbf{p} \in P_1 \mid \Omega_{P_2}(\mathbf{p}) = 1\}$	$\{\mathbf{p} \in P_2 \mid \Omega_{P_1}(\mathbf{p}) = 1\}$
$S_1 - S_2$	$\{\mathbf{p} \in P_1 \mid \Omega_{P_2}(\mathbf{p}) = 0\}$	$\{\mathbf{p} \in P_2 \mid \Omega_{P_1}(\mathbf{p}) = 1\}$
$S_2 - S_1$	$\{\mathbf{p} \in P_1 \mid \Omega_{P_2}(\mathbf{p}) = 1\}$	$\{\mathbf{p} \in P_2 \mid \Omega_{P_1}(\mathbf{p}) = 0\}$

Table 6.1 Classification for boolean operations.

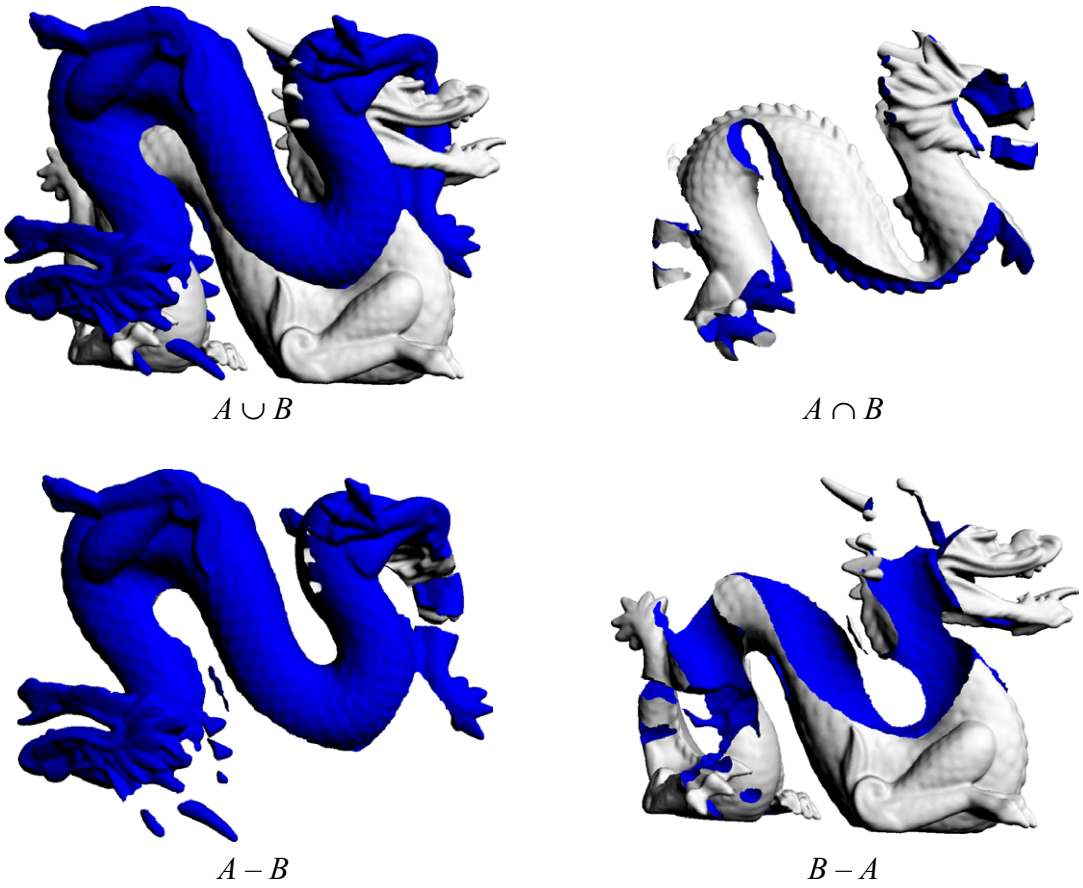


Figure 6.3 Boolean operations of a blue dragon (A) and a white dragon (B).

6.1.2 Intersection Curves

Taking the union of Q_1 and Q_2 will typically not produce a point cloud that accurately describes the surface S_3, since the intersection curve of the two MLS surfaces S_1 and S_2 is not represented adequately. Therefore, a set of sample points that lie on the intersection curve is explicitly computed and added to $Q_1 \cup Q_2$, to obtain the point cloud P_3. First, all points in Q_1 and Q_2 are found that are close to the intersection curve by evaluating the distance function induced by the MLS projection operator. From all closest pairs $(\mathbf{q}_1 \in Q_1, \mathbf{q}_2 \in Q_2)$ of these points a point $\mathbf{q}$ on the intersection curve is computed using a Newton-type iteration. This is done as follows (see Figure 6.4 (a-d)): Let $\mathbf{r}$ be the point on the intersection line of the two tangent planes

of $\mathbf{q}_1$ and $\mathbf{q}_2$ that is closest to both points, i.e., that minimizes the distance $\|\mathbf{r}-\mathbf{q}_1\|+\|\mathbf{r}-\mathbf{q}_2\|$. $\mathbf{r}$ is the first approximation of $\mathbf{q}$ and can now be projected onto S_1 and S_2 to obtain two new starting points $\mathbf{q}_1'$ and $\mathbf{q}_2'$ for the iteration. This procedure can be repeated iteratively until the points $\mathbf{q}_1$ and $\mathbf{q}_2$ converge to a point $\mathbf{q}$ on the intersection curve. Due to the quadratic convergence of the Newton iteration, this typically requires less than three iterations.

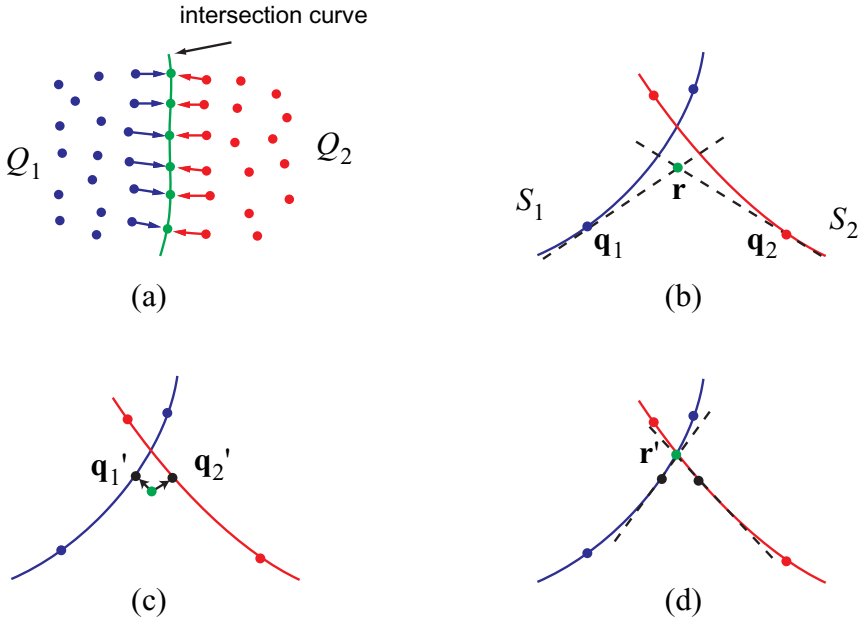


Figure 6.4 Sampling the intersection curve. (a) closest pairs of points in Q_1 and Q_2, (b) first estimate $\mathbf{r}$, (c) re-projection, (d) second estimate $\mathbf{r}'$.

The sampling density estimation of Section 2.1.2 is used to detect whether the sampling resolution of the two input surfaces differs significantly in the vicinity of the intersection curve. To avoid a sharp discontinuity in sampling density, the coarser model is up-sampled in this area to match the sampling density of the finer model, using the dynamic sampling method of Section 6.2.2.

Note that the above Newton scheme also provides an easy mechanism for adaptively refining the intersection curve. A simple subdivision rule is evaluated to create a new starting point for the Newton iteration, e.g., the average of two adjacent points on the curve. Applying the iteration then yields a new sample on the intersection curve (see Figure 6.5).

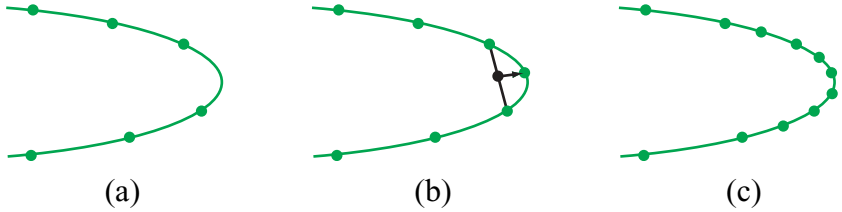


Figure 6.5 Adaptive refinement. (a) original intersection curve, (b) new point inserted in region of high curvature, (c) final, adaptively sampled intersection curve.

6.1.3 Rendering Sharp Creases

The accurate display of the intersection curves requires a rendering technique that can handle sharp creases and corners. For this purpose an extension of the *surface splatting* technique presented in [119] is used. In this method, each sample point is represented by a *surfel*, an oriented elliptical splat that is projected onto the screen to reconstruct the surface in image space (see also Section 8.1.2). A point on the intersection curve can now be represented by two surfels that share the same center, but whose normals stem from either one of the two input surfaces. During scan-conversion, each of these surfels is then clipped against the plane defined by the other to obtain a piecewise linear approximation of the intersection curve in screen space (see Figure 6.6). This concept can easily be generalized to handle corners as shown in Figure 6.6 (e).

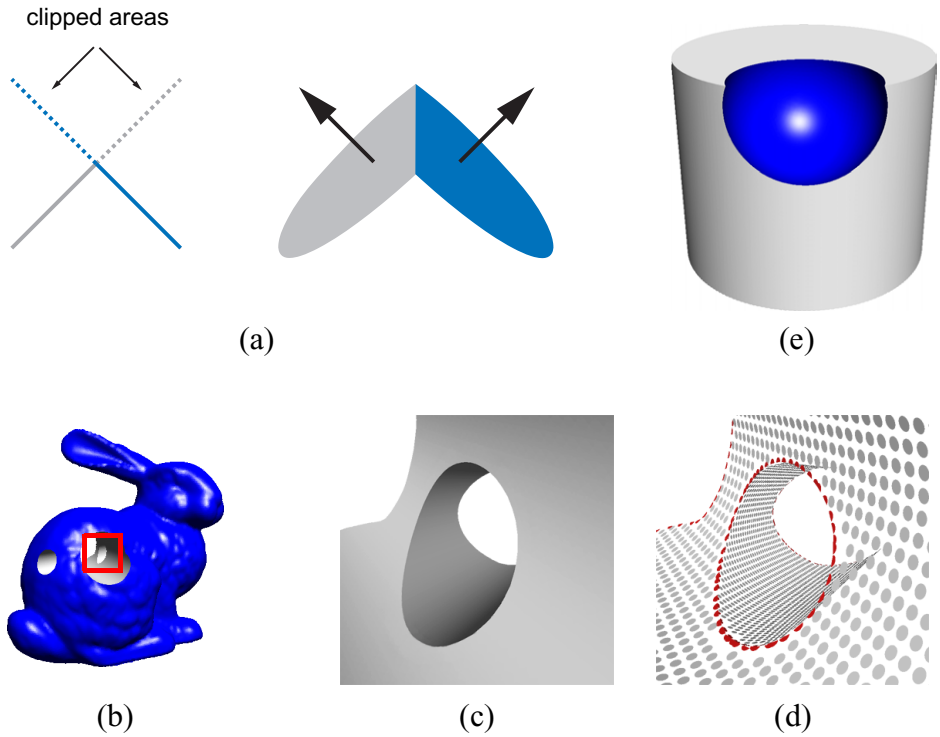


Figure 6.6 Rendering the intersection curve. (a) mutual clipping of two surfels on the intersection curve, (b) boolean differences on the bunny model, (c) zoom of the intersection curves, (d) sampling distribution, where samples on the intersection curve are rendered using two red half ellipses, (e) an example of a corner.

Figure 6.7 shows an example of a difficult boolean operation of two identical cylinders that creates two singularities. While the classification and intersection curve sampling work fine, the rendering method produces artefacts. This is due to numerical instabilities, since the clipping planes of two corresponding surfels are almost parallel. However, such cases are rare in typical computer graphics applications, e.g., digital character design. As such, the algorithms for boolean operations are less suited for industrial manufacturing applications, where robust handling of degenerated cases is of primary concern.

6.1.4 Particle-Based Blending

As illustrated in Figure 6.1, boolean operations typically produce sharp intersections. In some applications it is more desirable to create a smooth blend between the two

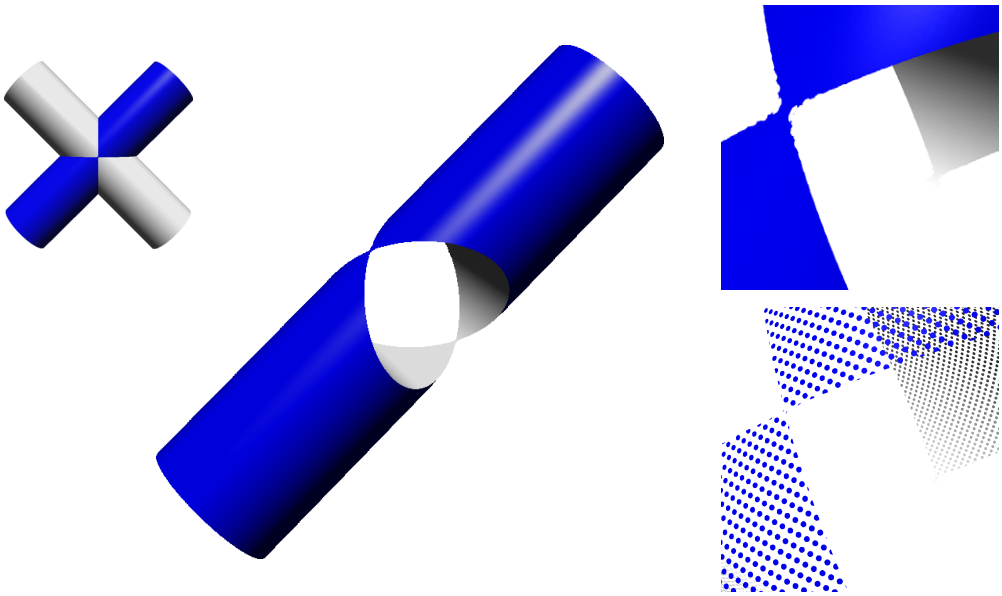


Figure 6.7 A difficult boolean difference operation that creates two singularities.

combined surface parts. To smooth out the sharp creases created by boolean operations an adaptation of oriented particles [107] has been implemented. The idea is to define inter-particle potentials $\Phi(\mathbf{p}_i, \mathbf{p}_j)$ in such a way that the minimum of the global potential function yields a smooth surface that minimizes curvature. Summing up these potentials yields a particle's total potential energy E_i. From this potential energy one can derive the positional and rotational forces that are exerted on each particle and compute its path of motion under these forces.

Additionally, an inter-particle repulsion force is applied to equalize the particle distribution (see also Section 3.4.2). All forces are scaled with a smooth fall-off function that measures the distance to the intersection curve to confine the particle simulation to a small area around the intersection curve without affecting other parts of the surface. A detailed discussion on implementational issues of the particle simulation used in this thesis can be found in [61].

Figure 6.8 shows the particle-based blending for the intersection of two planes, where the degree of smoothness can be controlled by the number of iterations of the simulation.

In Figure 6.9, a more complex blending operation is shown. A union operation of three tori has created numerous sharp intersection curves as shown in (a). These can be blended simultaneously as illustrated in (b) using the particle simulation described above. The same blending technique can of course also be applied to the intersection and difference operations described in Section 6.1.

Hole Filling

Witkin and Heckbert have introduced particle simulation to sample and control implicit surfaces [114]. Their algorithm dynamically adapts the distribution of particles by splitting or killing particles depending on the local particle density. This idea has been incorporated into the oriented particle simulation described above to handle non-uniformly sampled surfaces. An interesting application of this approach is hole filing for scanned surfaces, which are often incomplete due to occlusion in the scanning process (see Section 1.1.1). Figure 6.10 illustrates hole filling by adaptive

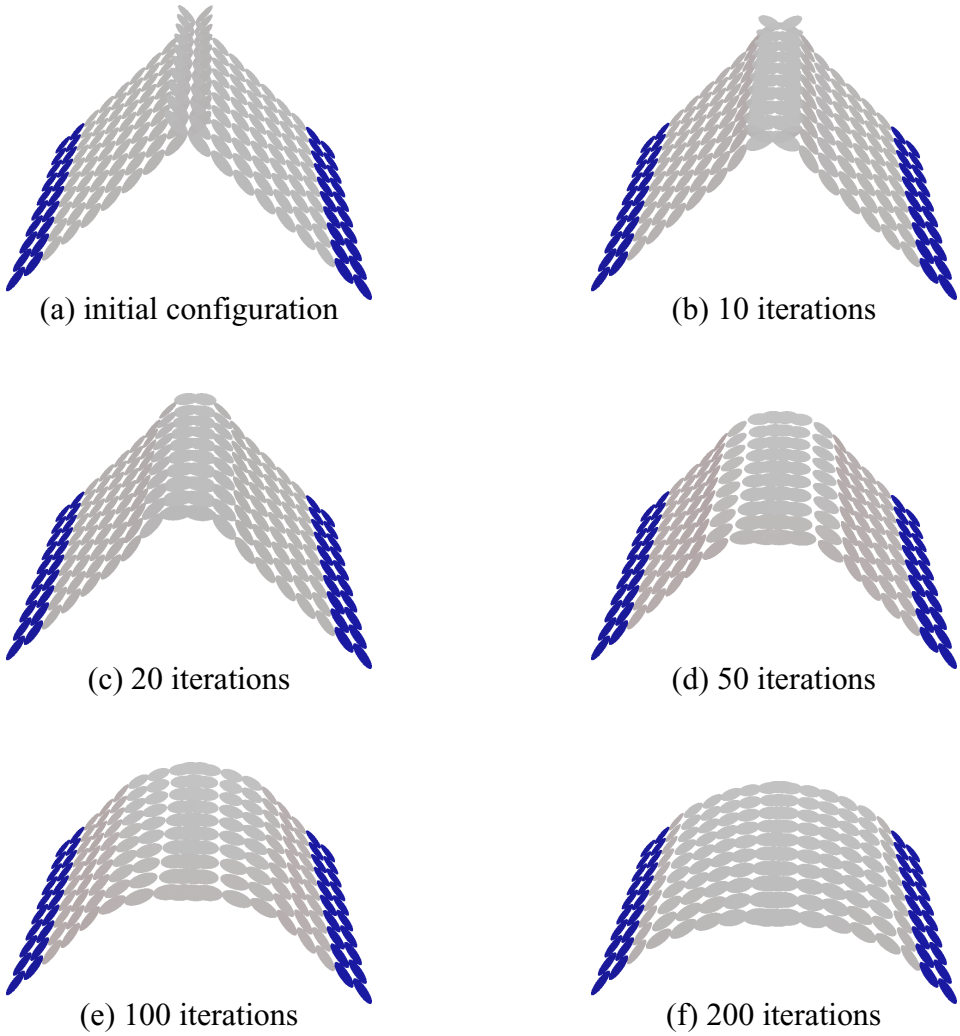
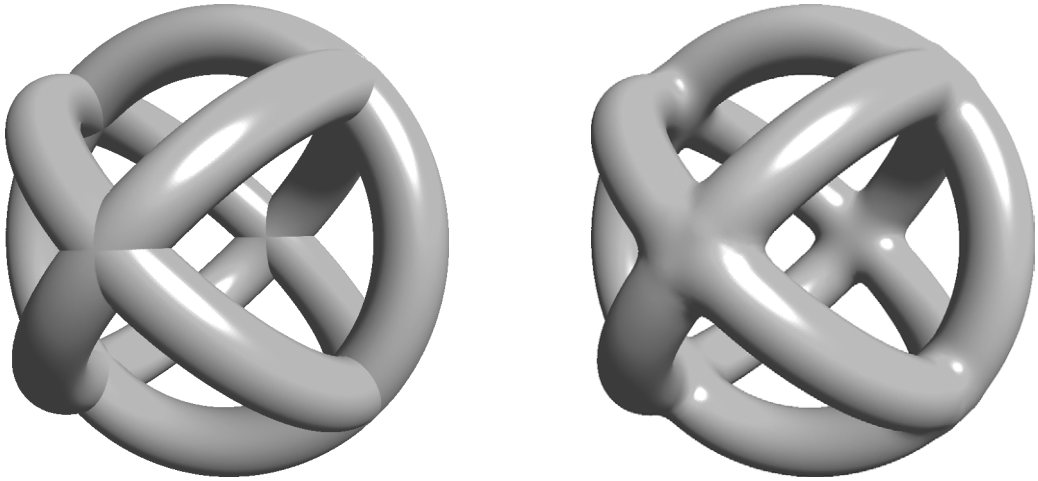


Figure 6.8 Particle simulation to blend two intersecting planes. Gray particles participate in the simulation, blue points indicate the fixed boundary.

particle simulation on the Igea model. Parts of the nose have been removed to simulate the effect of scanner occlusion. The samples on the so created boundary are marked as particles, either manually using a selection tool (see Section 8.1.1) or automatically using some surface boundary detection method (cf. Section 5.3.2). After starting the simulation, these particles will drift towards the center of the hole, away from its boundary. When the local particle density becomes too low, new particles will be introduced by splitting existing particles into two (see also Section 6.2.2). Eventually, the surface will be closed, as shown in Figure 6.10 (b) and (e).

Experiments showed that this hole-filling method works well for small holes with simple boundaries. It is not suitable as a general tool for pre-processing incomplete scanner data, however. When the size of the holes becomes too large, or their boundary is strongly convoluted, the particle simulation becomes unstable and does not converge to the desired watertight surface. For such cases, methods based on volumetric diffusion [21] are more appropriate.



(a) (b)

Figure 6.9 Boolean union of three tori. (a) reconstruction with sharp feature curves, (b) feature curves have been blended using particle simulation.

(a) (b) (c)

(d) (e) (f)

Figure 6.10 Hole filling using particle simulation. (a) original surface with hole, (b) surface after hole has been filled, (c) sampling distribution of (b), (d - f) zooms of (a - c).

6.2 FREE-FORM DEFORMATION

Apart from composition of surfaces using boolean operations, many shape design applications require the capability to modify objects using smooth deformations. These include bending, twisting, stretching, and compressing of the model surface. For this purpose a point-based free-form deformation tool is introduced that allows the user to interactively deform a surface by specifying a smooth deformation field.

The user first defines a deformable region $\chi_d \subset S$ on the model surface and marks parts of this region as a control handle. The surface can then be modified by pushing, pulling or twisting this handle. These user interactions are translated into a continuous tensor-field, which for each point in the deformable region defines a translatory and rotational motion under which the surface deforms. The tensor-field is based on a continuously varying scale parameter $t \in [0, 1]$ that measures the relative distance of a point from the handle. The closer a point is to the handle, the stronger will the deformation be for that point. More precisely, let $\chi_1 \subset \chi_d$ be the handle, also called *one-region*, and $\chi_0 = S - \chi_d$ the *zero-region*, i.e., all points that are not part of the deformable region. For both zero- and one-region distance measures d_0 and d_1, respectively, are defined as

$$d_j(\mathbf{p}) = \begin{cases} 0 & \mathbf{p} \in \chi_j \\ \min_{\mathbf{q} \in \chi_j}(\|\mathbf{p} - \mathbf{q}\|) & \mathbf{p} \notin \chi_j \end{cases}, \tag{6.2}$$

for $j = 0, 1$. From these distance measures the scale parameter t is computed as $t = \beta(d_0(\mathbf{p})/(d_0(\mathbf{p}) + d_1(\mathbf{p})))$, where $\beta{:}[0,1] \rightarrow [0, 1]$ is a continuous blending function with $\beta(0) = 0$ and $\beta(1) = 1$. Thus $t = 0$ for $\mathbf{p} \in \chi_0$ and $t = 1$ for $\mathbf{p} \in \chi_1$. Using this scale parameter, the position of a point $\mathbf{p} \in \chi_d$ after the deformation is determined as $\mathbf{p}' = F(\mathbf{p}, t)$, where F is a deformation function composed of a translatory and a rotational part. The deformation function can be written as $F(\mathbf{p}, t) = F_T(\mathbf{p}, t) + F_R(\mathbf{p}, t)$, where

- $F_T(\mathbf{p}, t) = \mathbf{p} + t \cdot \mathbf{v}$ with $\mathbf{v}$ a translation vector and
- $F_R(\mathbf{p}, t) = R(\mathbf{a}, t \cdot \alpha) \cdot \mathbf{p}$, where $R(\mathbf{a}, \alpha)$ is the matrix that specifies a rotation around axis $\mathbf{a}$ with angle α.

Figure 6.11 shows a translatory deformation of a plane where the translation vector $\mathbf{v}$ is equal to the plane normal. This figure also illustrates the effect of different choices of the blending function β. In Figure 6.12, two rotational deformations of a cylinder are shown, while a combination of both translatory and rotational deformations is illustrated in Figure 6.17.

To perform a free-form deformation the user only needs to select the zero- and one-regions and choose an appropriate blending function. She can then interactively deform the surface by displacing the handle with a mouse or trackball device, similar to [64]. This gives the method great flexibility for handling a wide class of free-form deformations, while still providing a simple and intuitive user interface. The deformable region and the handle can be specified using a simple paint tool that allows the user to mark points on the surface by drawing lines, circles, rectangles, etc. and applying flood filling and erasure. The system also supports pre-defined and user-editable selection stencils, which can be used to create embossing effects (Figure 6.13).

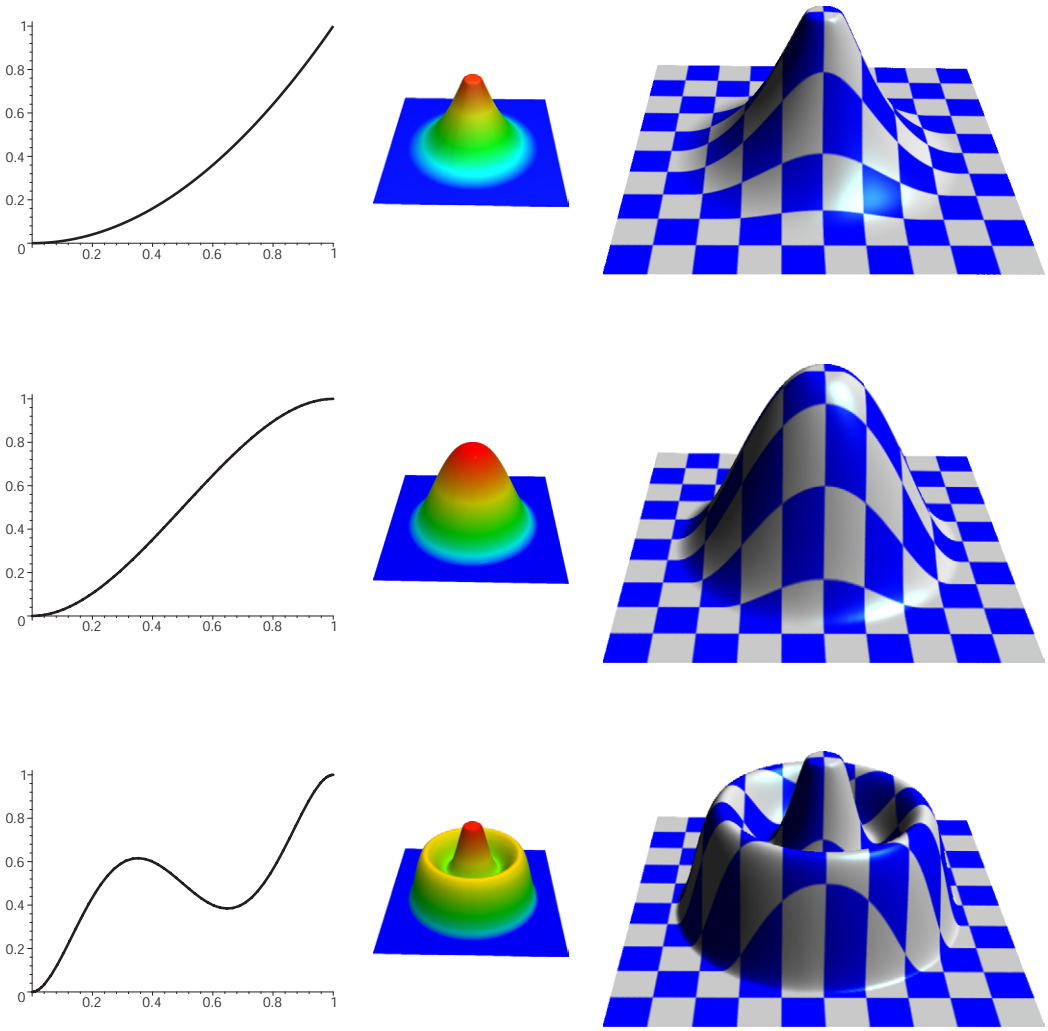


Figure 6.11 Deformations of a plane for three different blending functions. Left: Blending function, middle: Color-coded scale parameter, where blue indicates the zero region ($t = 0$) and red the one-region ($t = 1$), right: Final textured surface.

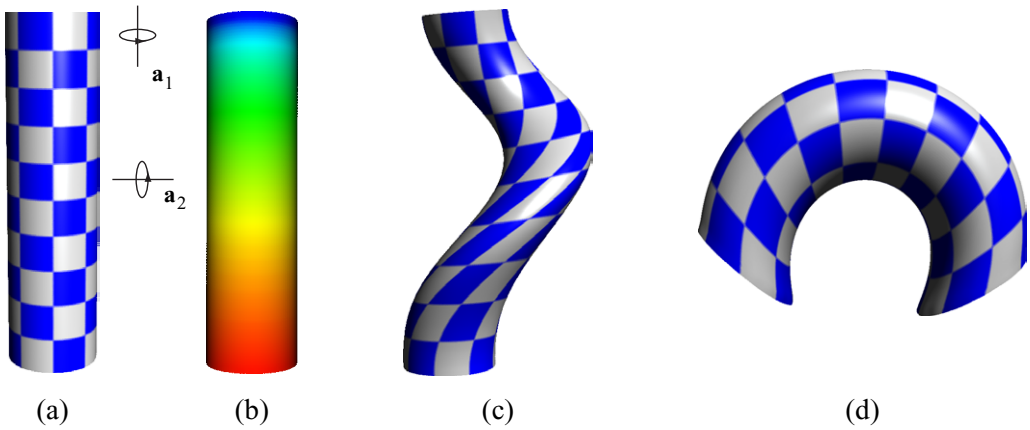


Figure 6.12 Rotational deformations of a cylinder. (a) original, (b) color-coded scale parameter, (c) rotation around axis $\mathbf{a}_1$, (d) rotation around axis $\mathbf{a}_2$.

Figure 6.13 Embossing effect. The Siggraph label on the top left has been converted to a selection stencil using simple image processing tools. This stencil can then be mapped to a surface, where blue color corresponds to the zero-region and red to the one-region. Subsequent deformation yields an embossing effect as shown on the right.

6.2.1 Topology Control

An important issue in shape design using free-form deformation is the handling of self-intersections. During deformation, parts of the deformable region can intersect other parts of the surface, which leads to an inconsistent surface representation. A solution to this problem requires a method for detecting and resolving such collisions.

Collision Detection

Similar to boolean operations, this requires an inside/outside classification to determine which parts of the surface have penetrated others. Thus the classification predicate Ω defined in Section 6.1.1 can be used for this purpose. First, the closest point $\mathbf{q} \in \chi_0$ to each sample point $\mathbf{p} \in \chi_d$ is computed. This defines an empty sphere s_p around $\mathbf{p}$ with radius $\|\mathbf{p} - \mathbf{q}\|$. If the point $\mathbf{p}$ only moves within this sphere during deformation, it is guaranteed not to intersect with the zero-region (see Figure 6.14). So additionally to exploiting spatial coherence as for boolean classification, this approach also exploits the temporal coherence induced by the smooth deformation field. The classification predicate Ω has to be re-evaluated only when $\mathbf{p}$ leaves s_p, which at the same time provides a new estimate for the updated sphere s_p.

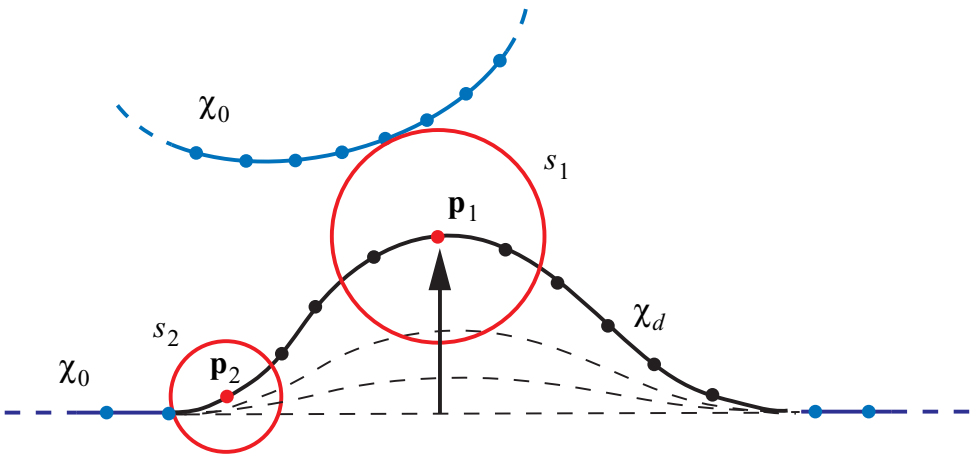


Figure 6.14 Temporal coherence for collision detection during deformation. The points $\mathbf{p}_1$ and $\mathbf{p}_2$ can move with the spheres s_1 and s_2, resp., without intersecting the zero-region.

Collision Handling.

There are different ways to respond to a detected collision. The simplest solution is to undo the last deformation step and recover the surface geometry prior to the collision. Alternatively, the penetrating parts of the surface can be joined using a boolean union operation to maintain the validity of the surface.

Figure 6.15 shows an editing session, where a deformation causes a self-intersection. After performing a boolean union, a sharp intersection curve is created as shown in (d). In the context of free-form deformation it is often more desirable to create a smooth transition between the two combined surface parts. Thus the particle simulation described in Section 6.1.4 can be used to blend the intersection region.

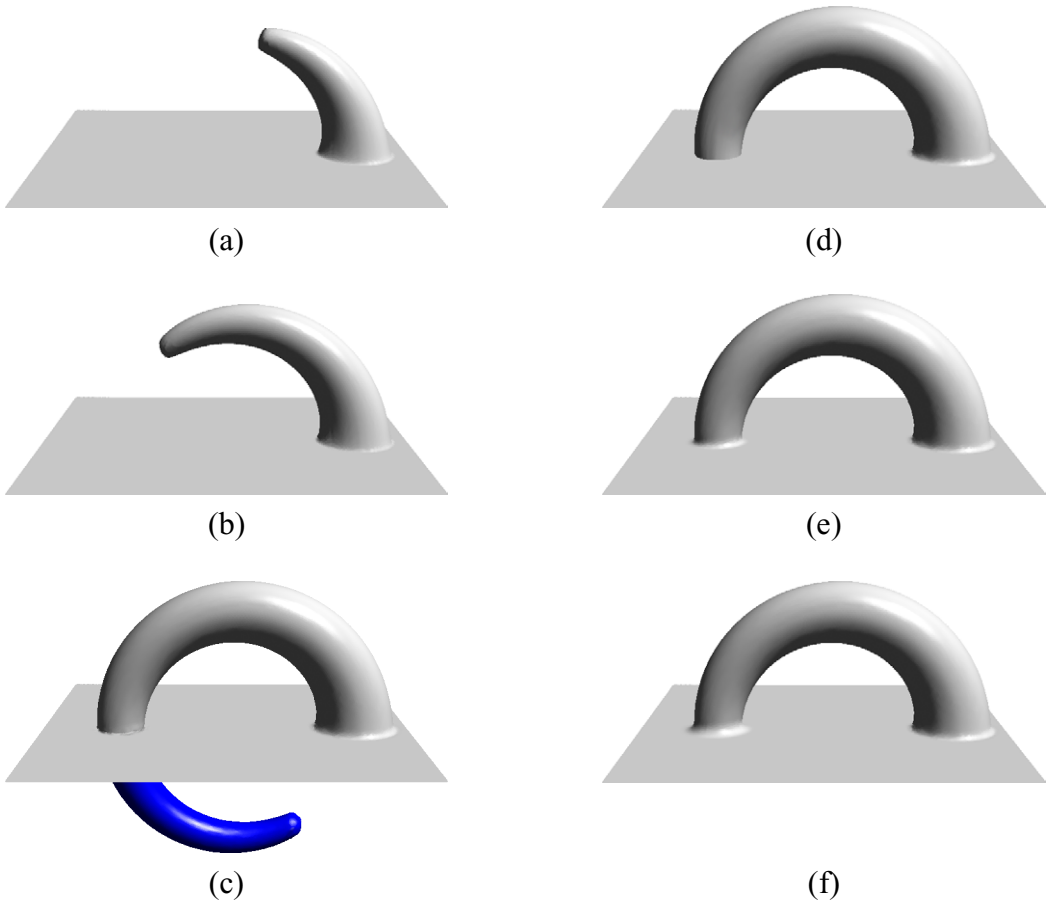


Figure 6.15 Interactive modeling session with collision detection. (a - b) intermediate steps of the deformation, (c) collision detection, where the blue part has been detected as self-intersecting, (d), boolean union with sharp intersection curve, (e - f), particle-based blending with different fall-off functions.

6.2.2 Dynamic Sampling

Large deformations may cause strong distortions in the distribution of sample points on the surface that can lead to an insufficient local sampling density. To prevent the point cloud from ripping apart and maintain a high surface quality, new samples have to be included where the sampling density becomes too low. This requires a method for measuring the surface stretch to detect regions of insufficient sampling density. Then new sample points have to be inserted and their position on the surface

determined. Additionally, scalar attributes, e.g., color values or texture coordinates, have to be preserved or interpolated.

Measuring Surface Stretch

The first fundamental form known from differential geometry [25] can be used to measure the local distortion of a surface under deformation. Let $\mathbf{u}$ and $\mathbf{v}$ be two orthogonal tangent vectors of unit length at a sample point $\mathbf{p}$. The first fundamental form at $\mathbf{p}$ is defined by the 2×2 matrix

$$\begin{bmatrix} \mathbf{u}^2 & \mathbf{u} \cdot \mathbf{v} \\ \mathbf{u} \cdot \mathbf{v} & \mathbf{v}^2 \end{bmatrix}. \tag{6.3}$$

The eigenvalues of this matrix yield the minimum and maximum stretch factors and the corresponding eigenvectors define the principal directions into which this stretching occurs. When applying a deformation, the point $\mathbf{p}$ is shifted to a new

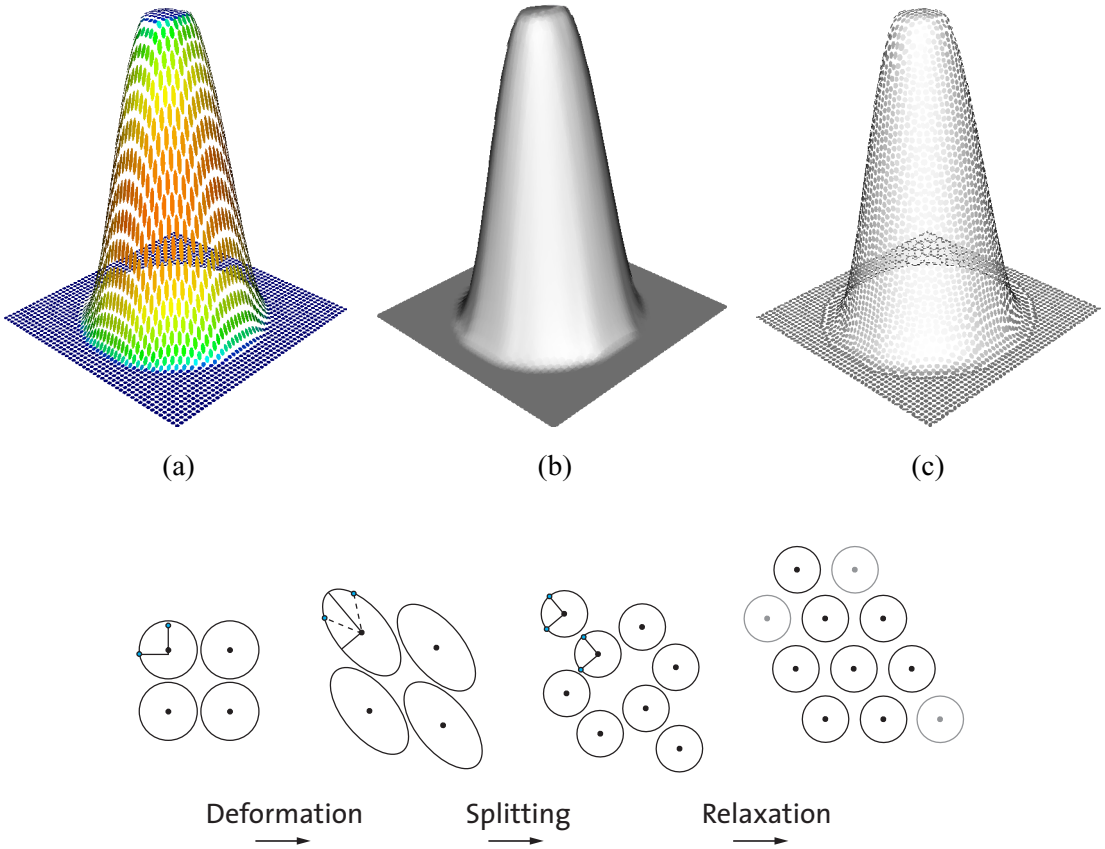


Figure 6.16 Dynamic sampling. Top row: Deformation of a plane. (a) local stretching: blue corresponds to zero stretch, while red indicates maximum stretch, (b) surface after re-sampling, (c) sampling distribution. Bottom row: illustration of point insertion.

position $\mathbf{p}'$ and the two tangent vectors are mapped to new vectors $\mathbf{u}'$ and $\mathbf{v}'$. Local stretching implies that $\mathbf{u}'$ and $\mathbf{v}'$ might no longer be orthogonal to each other nor do they preserve their unit length. The amount of this distortion can be measured by taking the ratio of the two eigenvalues of Equation 6.3 (local anisotropy) or by taking their

product (local change of surface area). When the local distortion becomes too strong, new samples have to be inserted to re-establish the prescribed sampling density. Since Equation 6.3 defines an ellipse in the tangent plane centered at $\mathbf{p}$ with the principal axes defined by the eigenvectors and eigenvalues, $\mathbf{p}$ can be replaced by two new samples $\mathbf{p}_1$ and $\mathbf{p}_2$ that are positioned on the main axis of the ellipse (see Figure 6.16).

6.2.3 Filter Operations

Whenever a splitting operation is applied, both the geometric position and the scalar function values for the newly generated sample points have to be determined. Both these operations can be described as the application of a filtering operator: A *relaxation filter* determines the sample positions while an *interpolation filter* is applied to obtain the function values.

Relaxation

Introducing new sample points through a splitting operation creates local imbalances in the sampling distribution. To obtain a more uniform sampling pattern, a relaxation operator is applied that moves the sample points within the surface (see Figure 6.16). Similar to [111] (see also Section 3.4) a simple point repulsion scheme is used with a repulsion force that drops linearly with distance. This confines the radius of influence of each sample point to its local neighborhood, which allows very efficient computation of the relaxation forces. The resulting displacement vector is then projected into the point's tangent plane to keep the samples on the surface.

Interpolation

Once the position of a new sample point $\mathbf{p}$ is fixed using the relaxation filter, the associated function values need to be determined. This can be achieved using an interpolation filter by computing a local average of the function values of neighboring sample points. The relaxation filter potentially moves all points of the neighborhood of $\mathbf{p}$. This tangential drift leads to distortions in the associated scalar functions. To deal with this problem a copy of each point that carries scalar attributes is created and its position is fixed during relaxation. In particular, for each sample that is split a copy is maintained with its original data. These points will only be used for interpolating scalar values, they are not part of the current geometry description. Since these samples are *dead* but their function values still *live*, they are called *zombies*. Zombies will undergo the same transformation during a deformation operation as living points, but their positions will not be altered during relaxation. Thus zombies accurately describe the scalar attributes without distortions. Therefore, zombies are only used for interpolation, while for relaxation only living points are considered. After an editing operation is completed, all zombies will be deleted from the representation.

Figure 6.17 illustrates this dynamic re-sampling method for a very large deformation that leads to a substantial increase in the number of sample points. While the initial plane consists of 40,000 points, the final model contains 432,812 points, clearly demonstrating the robustness and scalability of the method in regions of extreme surface stretch.

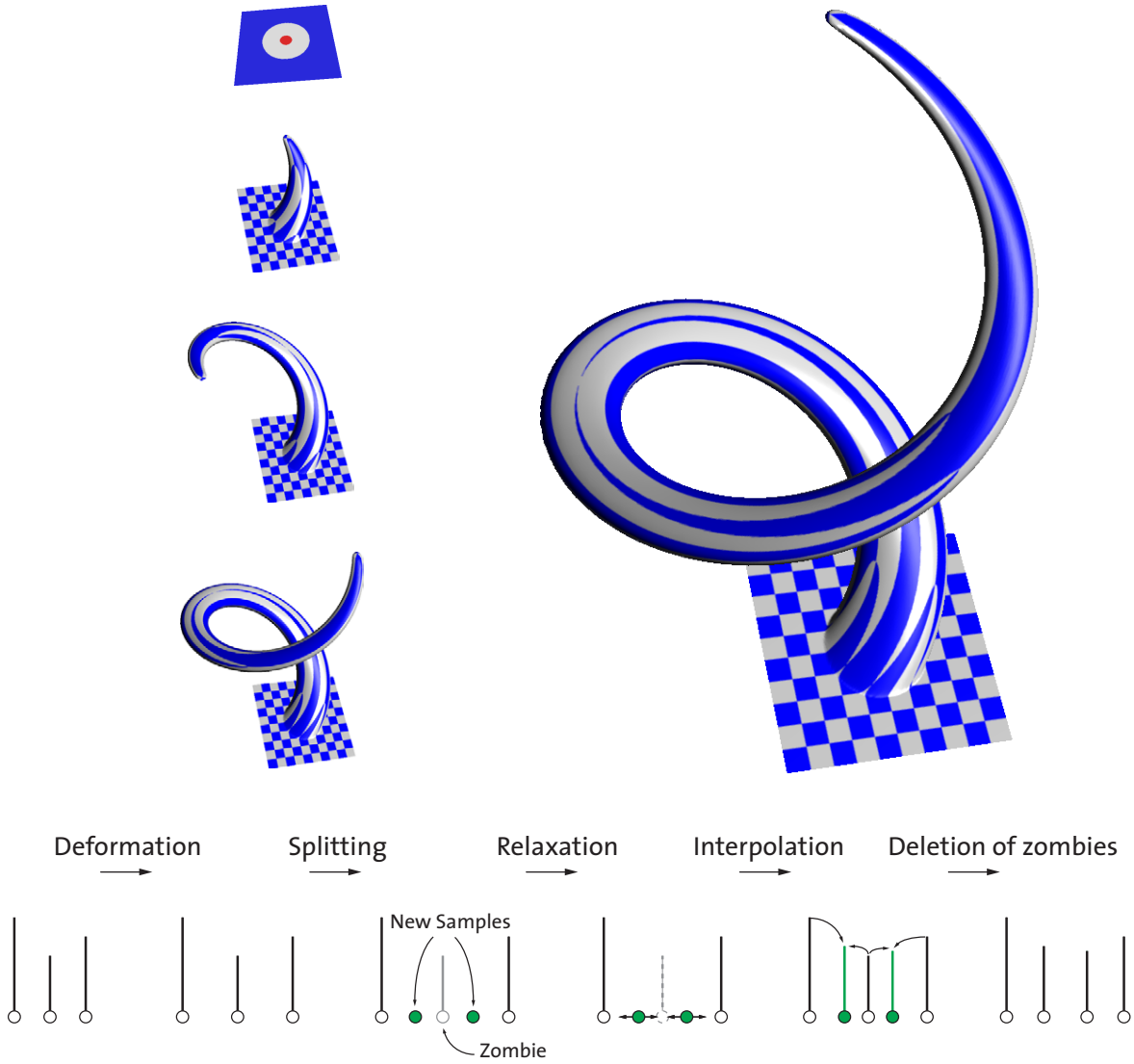


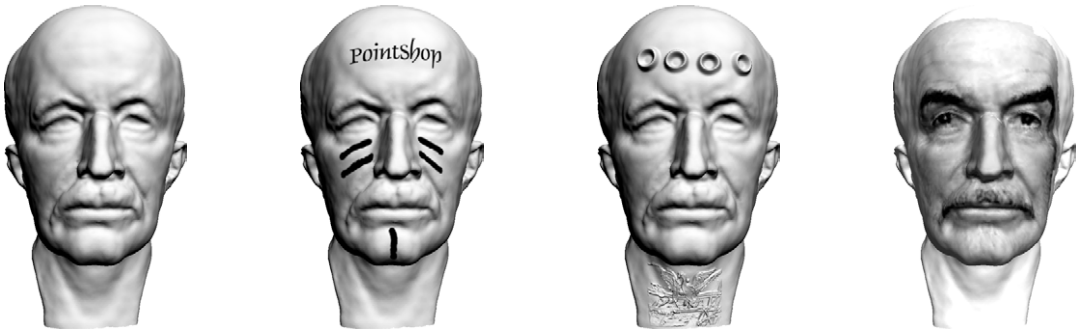
Figure 6.17 A very large deformation using a combination of translatory and rotational motion. The left column shows intermediate steps with the top image indicating zero- and one-regions. Each point of the surface carries texture coordinates, which are interpolated during re-sampling and used for texturing the surface with a checkerboard pattern. The bottom row illustrates this interpolation process, where the function values are indicated by vertical lines.

6.2.4 Down-Sampling

Apart from lower sampling density caused by surface stretching, deformations can also lead to an increase in sampling density, where the surface is squeezed. It might be desirable to eliminate samples in such regions while editing, to keep the overall sampling distribution uniform. However, dynamically removing samples also has some drawbacks. Consider a surface that is first squeezed and then stretched back to its original shape. If samples get removed during squeezing, surface information such as color will be lost, which leads to increased blurring when the surface is re-stretched. Thus instead of dynamic sample deletion, an optional "garbage collection" is performed at the end of the editing operation. To reduce the sampling density, any of the simplification methods of Chapter 3 can be used.

7

APPEARANCE MODELING



The previous chapter has introduced boolean operations and free-form deformation for point-based shape modeling. Apart from the mere geometric shape, which is described by the position of the sample points in space, 3D objects also carry a number of additional attributes, such as color or material properties, that determine the overall appearance of the surface. In this chapter, a number of tools and algorithms for interactively editing surface appearance attributes will be presented [118]. These methods can be understood as a generalization of common photo editing techniques from 2D images to 3D point-sampled surfaces.

7.1 OVERVIEW

This section gives an overview of the appearance modeling functionality for point-sampled models by defining a surface editing operation on an abstract level. A brief analysis of 2D photo editing identifies three fundamental building blocks of an interactive editing operation: parameterization, re-sampling, and editing. It will be shown that these concepts can be extended from 2D photo editing to 3D surface editing. For this purpose an operator notation will be introduced that allows a wide variety of editing operations to be defined in a unified and concise way. The

fundamental differences between functional images and manifold surfaces lead to different implementations of these operators, however.

7.1.1 2D Photo Editing

A 2D image I can be considered as a discrete sample of a continuous image function containing image attributes such as color or transparency. Implicitly, the discrete image I always represents a continuous image, yet image editing operations are typically performed directly on the discrete samples.

A general image editing operation can be described as a function of a given image I and a brush image B. The brush image is used as a generic tool to modify the image I. Depending on the considered operation it may be interpreted as a paint brush or a discrete filter, for example.

The editing operation involves the following steps: First, a parameter mapping Φ has to be found that aligns the image I with the brush B. For example, Φ can be defined as the translation that maps the pixel at the current mouse position to the center of B. Next, a common sampling grid for I and B has to be established, such that there is a one-to-one correspondence between the discrete samples. This requires a re-sampling operator Ψ that first reconstructs the continuous image function and then samples this function on the common grid. Finally, the editing operator Ω combines the image samples with the brush samples using the one-to-one correspondence established before. The resulting discrete image I' is then obtained as a concatenation of the operators described above:

$$I' = \Omega(\Psi(\Phi(I)), \Psi(B)). \tag{7.1}$$

The goal is to generalize the operator framework of Equation 7.1 to irregular point-sampled surfaces, as illustrated in Figure 7.1. Formally, this can be done by replacing the discrete image I by a point cloud P that represents a surface S. The discrete sample points can be considered as a direct extension of image pixels, carrying the attributes shown in Figure 7.4 (a). This motivates the term *surfel*, short for *surface element*, similar to *pixel*, which stands for *picture element* (see also [92]). The transition from image to surface has the following effects on the individual terms of Equation 7.1:

Parameterization Φ

For photo editing, the parameter mapping Φ is usually specified by a simple, global 2D to 2D affine mapping, i.e., a combination of translation, scaling, and rotation. Mapping a manifold surface onto a 2D domain is much more involved, however. Therefore, the user interactively selects subsets, or *patches*, of S that are parameterized individually. In general, such a mapping leads to distortions that cannot be avoided completely. Section 7.2 will introduce two algorithms for computing a parameterization that correspond to two different interaction schemes: Parameterization by orthogonal projection for interactive brush painting, and a method to compute a constrained minimum distortion parameterization. The latter allows the user to control the mapping in an intuitive manner by setting corresponding feature points both on the parameter plane and the surface, respectively.

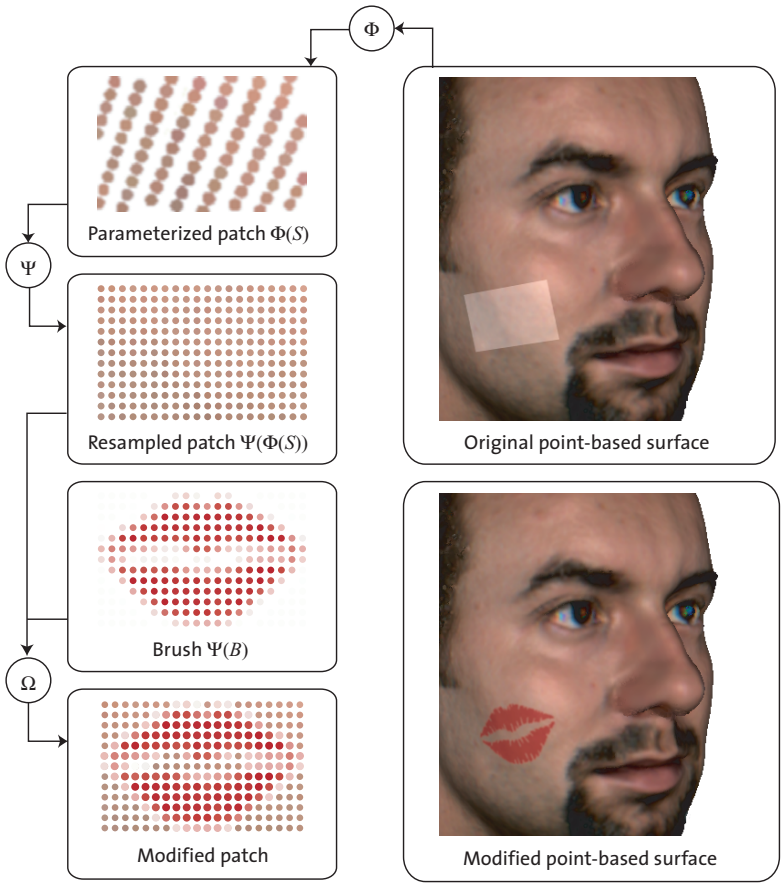


Figure 7.1 Overview of the operator framework for point-based surface editing [118].

Re-sampling Ψ

Images are usually sampled on a *regular* grid, hence signal processing methods can be directly applied for re-sampling. However, the sampling distribution of surfaces is in general *irregular*, requiring alternative methods for reconstruction and sampling. For this purpose a parameterized scattered data approximation is used that reconstructs a continuous function from the samples (see [118]). This continuous function can then be evaluated at the desired sampling positions.

Editing Ω

Once the parameterization is established and re-sampling has been performed, all computations take place on the discrete samples in the 2D parameter domain. Hence the full functionality of photo editing systems can be applied for texturing and texture filtering. Additionally, operations that modify the geometry, e.g., sculpting or geometry filtering, can easily be incorporated into the system. As will be described in Section 7.1.3, all of these tools are based on the same simple interface that specifies an editing tool by a set of bitmaps and few additional parameters. For example, a sculpting tool is defined by a 2D displacement map, an alpha mask and an intrusion depth.

7.1.2 Interaction Modes

The appearance attributes of a point-sampled model can be manipulated using two different interaction schemes:

Brush Interaction

In this interaction mode the user moves a brush device over the surface and continuously triggers editing events, e.g., painting operations (see Figure 7.2). The brush is positioned using the mouse cursor and aligned with the surface normal at the current interaction point. In terms of Equation 7.1, this means that the parameterization is continuously and automatically re-computed and re-sampling is performed for each editing event. A complete editing operation is then performed using a fixed brush image.

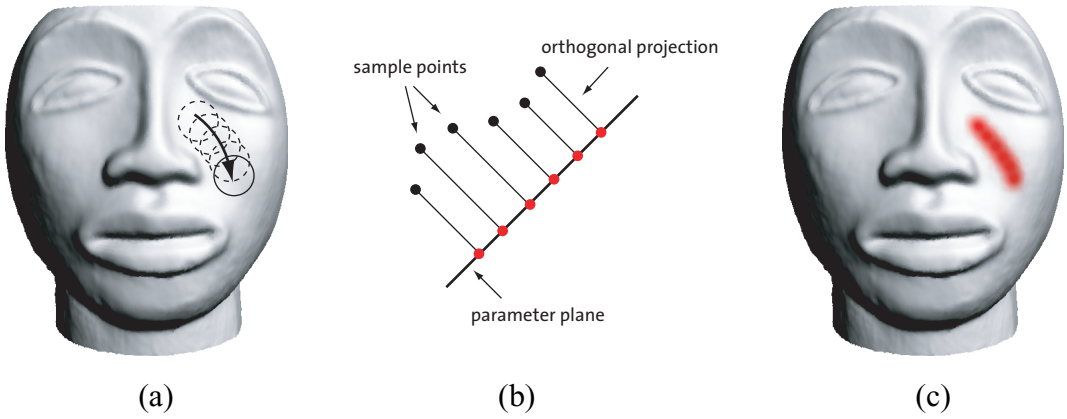


Figure 7.2 Brush interaction: (a) brush cursor movement, (b) parameterization by orthogonal projection onto the brush plane (2D for illustration), (c) painted surface

Selection Interaction

Here the user first selects a subset of the surface called a patch and defines the parameter mapping interactively by imposing point constraints (see Figure 7.3). Based on this fixed parameterization, various editing operations can be applied. Hence parameterization and re-sampling operators in Equation 7.1 are evaluated once, while different editing operators can be applied successively.

7.1.3 Brush Interface

All appearance modeling operations are based on a generic brush interface (see also Figure 7.1). A brush is defined as a $M \times N$ grid B, where each grid point $\mathbf{b}_{mn} = \{\mathbf{c}_{mn}, d_{mn}, \mathbf{s}_{mn}, k_{a,mn}, k_{d,mn}, k_{s,mn}, s_{mn}\}$, $1 \le m \le M$, $1 \le n \le N$ stores all the surface appearance attributes shown in Figure 7.4 (a). Each individual bitmap, e.g., the diffuse color image $\{\mathbf{c}_{mn}\}$, defines a *brush channel* that represents a corresponding continuous attribute function, similar to the surface samples P that represent the continuous surface S. Additionally, each brush channel carries an alpha mask that can be used for blending the brush coefficients with the surface coefficients as described in Section 7.4.1. The channel $\{d_{mn}\}$ defines a bitmap of displacement coefficients that

(a) (b) (c)

Figure 7.3 Selection Interaction: (a) Feature points on parameter plane, (b) feature points on surface, (c) final texture-mapped surface.

can be used for sculpting operations, such as normal displacement or carving (see Section 7.4.2).

Figure 7.4 shows a typical brush that combines texture, geometry and material properties to support flexible editing operations.

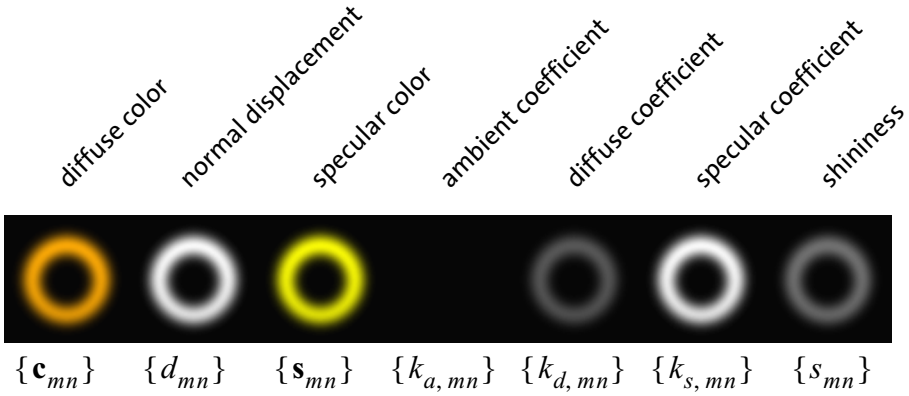
7.2 PARAMETERIZATION

This section describes two different methods to compute a parameterization for a point-sampled surface that correspond to the two interaction schemes defined above. For brush interaction the parameter mapping will be computed by a simple orthogonal projection, while an optimization method is applied for computing a constrained minimum distortion parameterization for selection interactions (see also [117]). To define the parameterization Φ, the user first selects a subset S' of the surface S, described by a point cloud $P' \subseteq P$. A mapping $\Phi: P' \rightarrow [0, 1] \times [0, 1]$ is then computed that assigns parameter coordinates $\mathbf{u}_i$ to each point $\mathbf{p}_i \in P'$.

7.2.1 Orthogonal Projection

A simple method for computing a parameter mapping is dimension reduction. 2D parameter values for a point $\mathbf{p}_i \in P'$ can be obtained by simply discarding one of the three spatial coordinates. With an appropriate prior affine transformation, this amounts to an orthogonal projection of the sample points onto a plane. This plane can either be specified by the user, or computed automatically according to the distribution of the sample points, e.g., as a least-squares fit. Using covariance analysis (see Section 2.1.3), the normal vector of the parameter plane would then be chosen as the eigenvector of the covariance matrix with smallest corresponding eigenvalue.

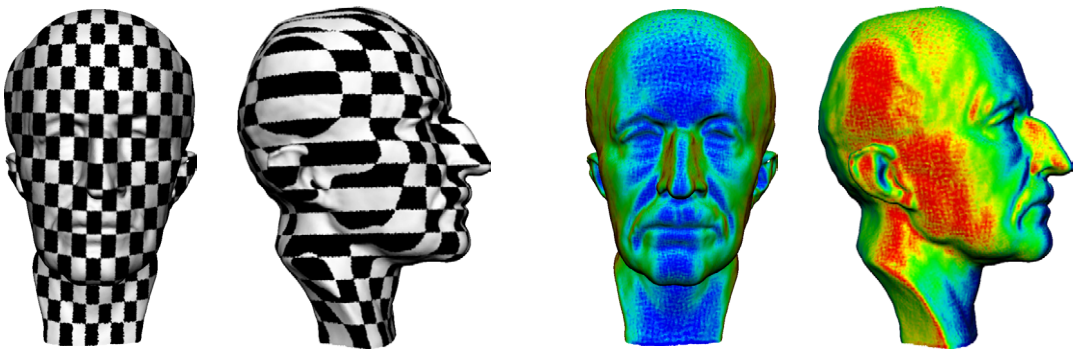
Figure 7.5 shows examples of texture-mapping operations using a parameterization obtained by orthogonal projection. In general, such a mapping will exhibit strong distortions and discontinuities, leading to inferior editing results. However, if the surface patch is sufficiently small, distortions will be small too and no discontinuities will occur. Thus orthogonal projection is a suitable parameterization method for brush interactions, where the parameter plane is defined by the surface normal at the tool



(a)

(b)

Figure 7.4 Brush interface: (a) a brush specified by a set of bitmaps, (b) the brush applied to a surface. The zooms on the right show the surface under different illumination to illustrate how the reflectance properties have been modified.



(a) (b)

Figure 7.5 The Max Planck model parameterized by orthogonal projection from the front. (a) texture mapped surface, (b) the color-coded first derivative of the parameterization measures the stretch of the mapping, where blue corresponds to minimum, red to maximum stretch.

cursor and the surface patch is defined by the projection of the brush onto the surface. as shown in Figure 7.2.

7.2.2 Constrained Minimum Distortion Parameterization

As Figure 7.5 illustrates, orthogonal projection leads to strong distortions in the parameterization. Furthermore, it provides little support for interactive control of the mapping by the user. Consider the typical texture mapping operation shown in Figure 7.3. The goal is to map a 2D image of a human face onto a laser-range scan of a different face. It is certainly desirable to minimize the distortion of the mapping. Equally important, however, is a good correspondence of feature points, e.g., the tip of the nose in the image should be mapped onto the tip of the nose on the surface. Thus some mechanism is needed that allows the user to define corresponding feature points both in the image and on the surface. These point-to-point correspondences are then incorporated as constraints into an optimization that computes the mapping [77].

First, an objective function for a continuous surface patch is defined that penalizes high distortion as well as the approximation error of the feature point constraints. A suitable discretization then yields a system of linear equations that can be solved using conjugate gradient methods [104].

Objective Function

A continuous parameterized surface S_X can be defined by a mapping

$$X\colon [0,1]\times[0,1] \rightarrow S_X \subset \mathbf{IR}^3, \tag{7.2}$$

which for each parameter value

$$\mathbf{u} = (u, v) \in [0,1]\times[0,1]$$

determines a point

$$\mathbf{x} = X(\mathbf{u}) = (x(\mathbf{u}), y(\mathbf{u}), z(\mathbf{u})) \in S_X \tag{7.3}$$

on the surface S_X. The mapping X defines a parameterization of the surface S_X. Let $U = X^{-1}$ be the inverse mapping, i.e., a function that assigns parameter coordinates $\mathbf{u}$ to each point $\mathbf{x} \in S_X$. The distortion of the parameter mapping can be measured using the cost function

$$C_{dist}(X) = \int_H \gamma(\mathbf{u}) d\mathbf{u} \quad , \tag{7.4}$$

where $H = [0,1]\times[0,1]$,

$$\gamma(\mathbf{u}) = \int_\theta \left(\frac{\partial^2}{\partial r^2} X_{\mathbf{u}}(\theta, r) \right)^2 d\theta, \tag{7.5}$$

and

$$X_{\mathbf{u}}(\theta, r) = X\left(\mathbf{u} + r \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} \right). \tag{7.6}$$

$\gamma(\mathbf{u})$ is defined as the integral of the squared second derivative of the parameterization in each radial direction at a parameter value $\mathbf{u}$ using a local polar re-parameterization $X_{\mathbf{u}}(\theta, r)$. If $\gamma(\mathbf{u})$ vanishes, the parameterization is arc length preserving, i.e., defines a polar geodesic map at $\mathbf{u}$.

Additionally, a set M of point-to-point correspondences can be specified such that a point $\mathbf{p}_j$ of the point cloud corresponds to a point $\mathbf{u}_j$ in the parameter domain for $j \in M$. These point pairs serve as constraints that are approximated in a least-squares sense using the cost function

$$C_{fit}(X) = \sum_{j \in M} \{X(\mathbf{u}_j) - \mathbf{p}_j\}^2 . \tag{7.7}$$

The two cost function C_{dist} and C_{fit} can be combined into the objective function

$$C(X) = C_{fit}(X) + \beta \cdot C_{dist}(X), \tag{7.8}$$

where β is an additional parameter that allows to control the relative weight of the fitting error and distortion measure. This derivation of the objective function follows Levy's method for triangle meshes [77]. By replacing the mesh connectivity with a point neighborhood relation as defined in Section 2.1.1, a discrete formulation of the objective function can be derived for point-sampled surfaces. This requires a discretization of the directional derivatives in Equation 7.5, which can be obtained using divided differences on a discrete set of normal sections. For a complete derivation, the reader is referred to [118]. When substituting X for U, the discrete objective function finally has the form

$$\tilde{C}(U) = \sum_j \left(\mathbf{b}_j - \sum_{i=1}^{n} a_{j,i} \mathbf{u}_i \right)^2 = \| \mathbf{b} - A\mathbf{u} \|^2 , \tag{7.9}$$

where $\mathbf{u}$ is the vector of all unknowns $\mathbf{u}_i = (u_i, v_i)^T$. The coefficients $a_{j,i}$ result from the discretization of the second derivatives and the $\mathbf{b}_j$ are derived from the fitting constraints. Using normal equations, the linear least squares problem of Equation 7.9 can be transformed to a sparse system of linear equations, which can be solved with conjugate gradient methods [104].

Nested Iteration

A standard approach for improving the convergence of iterative conjugate gradient solvers is nested iteration. The system is first solved on a coarse representation and this solution is propagated to the finer representation. Since each unknown in Equation 7.9 corresponds to a sample of the model point cloud, a coarser representation can be obtained using the clustering methods presented in Section 3.2. This scheme can be recursively extended to a hierarchy of nested approximations as illustrated in Figure 7.6. Special care has to be taken to define the constraints on the coarser levels, as the original point pair correspondences were defined on the finest level. A simple solution is to just propagate the constraint to the centroid of the cluster it belongs to. If a point carries more than one constraint at a certain level, all but one of the constraints are removed on this level to maintain the injectivity of the mapping.

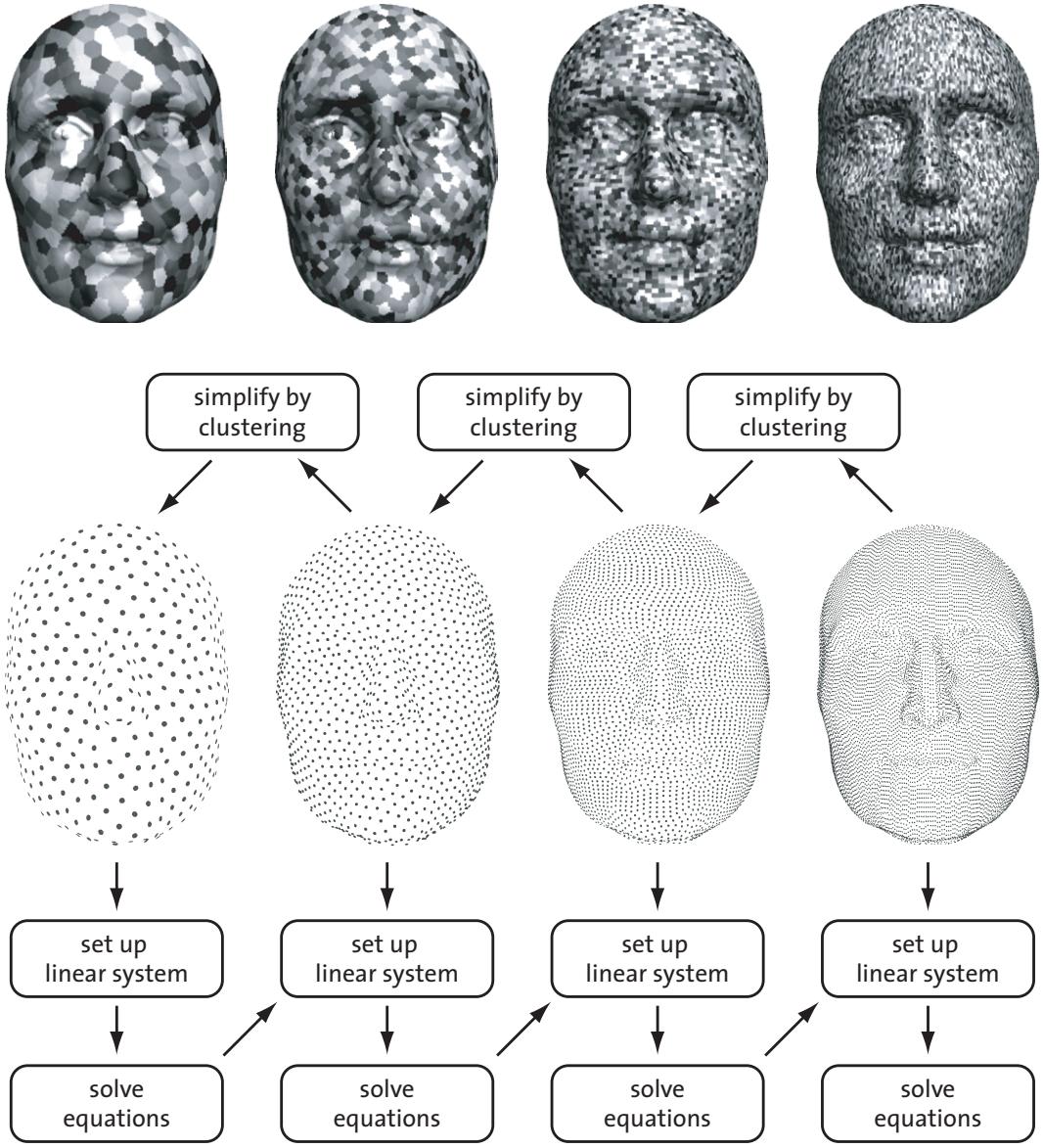


Figure 7.6 Hierarchy used for nested iteration. The top-row shows the clusters color-coded on the original point cloud.

Discussion

Figure 7.7 shows the influence of the parameter β in Equation 7.8. As the images illustrate, it allows the user to define a trade-off between the approximation of the fitting constraints and the smoothness of the mapping.

Floater has presented a different approach to compute a parameterization for point-sampled surfaces using shape preserving weights [34]. In his method the system of equations is obtained by expressing each unknown as a convex combination of its neighbors. This means that the boundary of the surface has to be mapped onto a convex region in parameter space. In the above formulation no such constraints have been imposed, which allows the method to be used as an extrapolator.

Note that at least three point-pair correspondences are required to define the mapping. Also, if the point cloud is a sample of a plane and the fitting constraints can

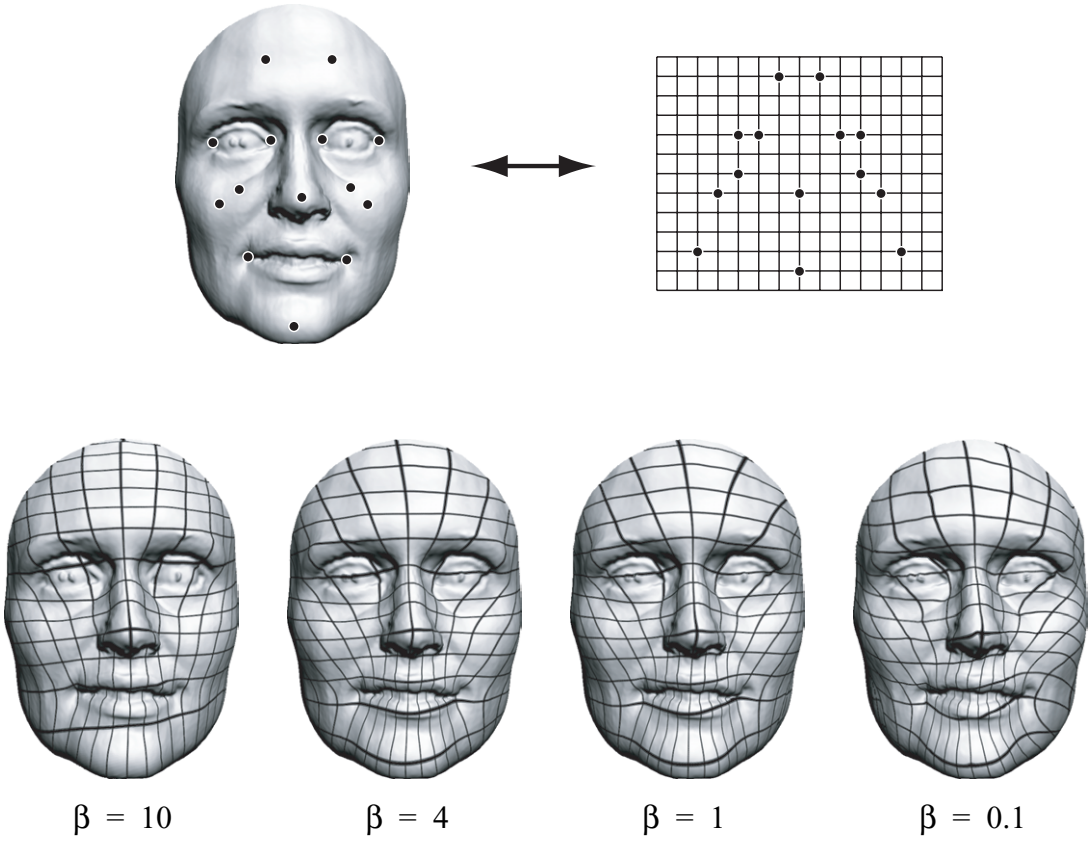


Figure 7.7 Influence of the weighting parameter β of Equation 7.8. The top images illustrate the feature point correspondences, the bottom row show the resulting mapping for different values of β.

be expressed by an affine mapping, then the parameterization will be an affine mapping too (see Figure 7.8 (a)). As illustrated in Figure 7.8 (b), the parameterization is not guaranteed to be bijective. It is rather left to the user to select a suitable patch and appropriate point correspondences to obtain the desired mapping.

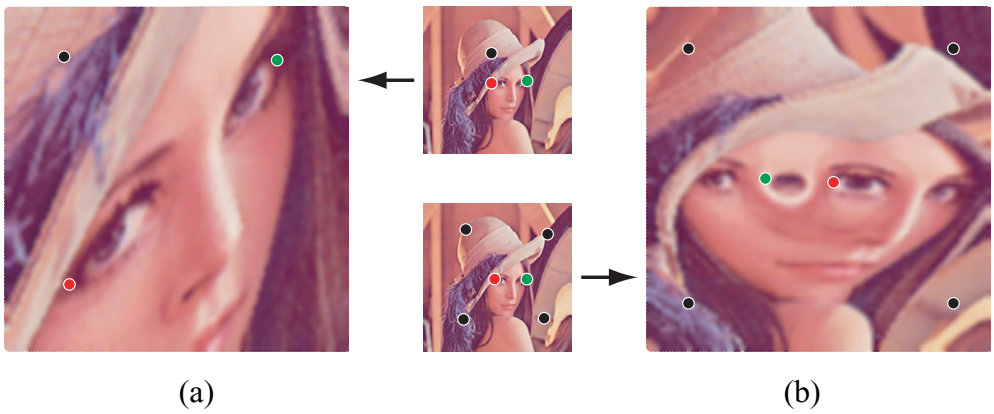


Figure 7.8 Constrained minimum distortion parameterization: (a) reproduction property, (b) the parameterization is not guaranteed to be bijective.

Figure 7.9 shows two examples of a texture mapping operation using the minimum distortion parameterization. As described in Section 7.1.2, this type of operation requires the user to interactively define corresponding feature points both on the 2D image and on the 3D point-sampled surface, as shown on the left. Figure 7.10 illustrates the distortion of these parameterizations by mapping a regular square grid texture onto the surface.

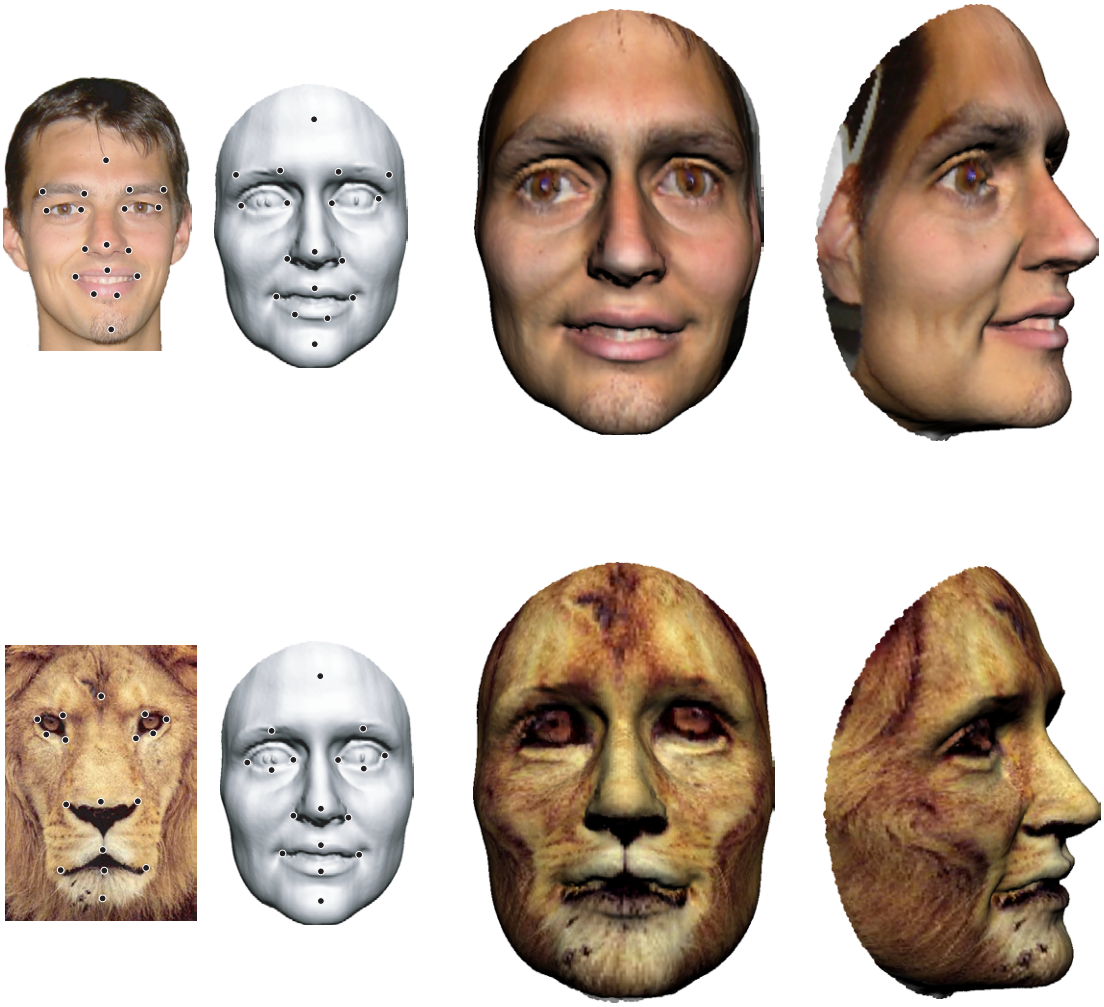


Figure 7.9 Texture mapping using the minimum distortion parameterization. The images on the left show the corresponding feature points. On the right, the final textured surface is shown.

Performance data for the example shown in the top row of Figure 7.9 is given in Table 7.1 (similar results are obtained for the lion example). 18 feature point correspondences have been specified on a patch consisting of 39,194 sample points. During initialization, the cluster hierarchy is created and the system of linear equations is set up. The system of equations is solved in the solving stage using iterative methods as described above. This example shows that significant speed-ups can be achieved using the nested iteration approach. In particular, the time of the solving stage is reduced up to a factor of 7 at a moderate increase in initialization cost. This is especially useful when the user interactively changes the feature point constraints to

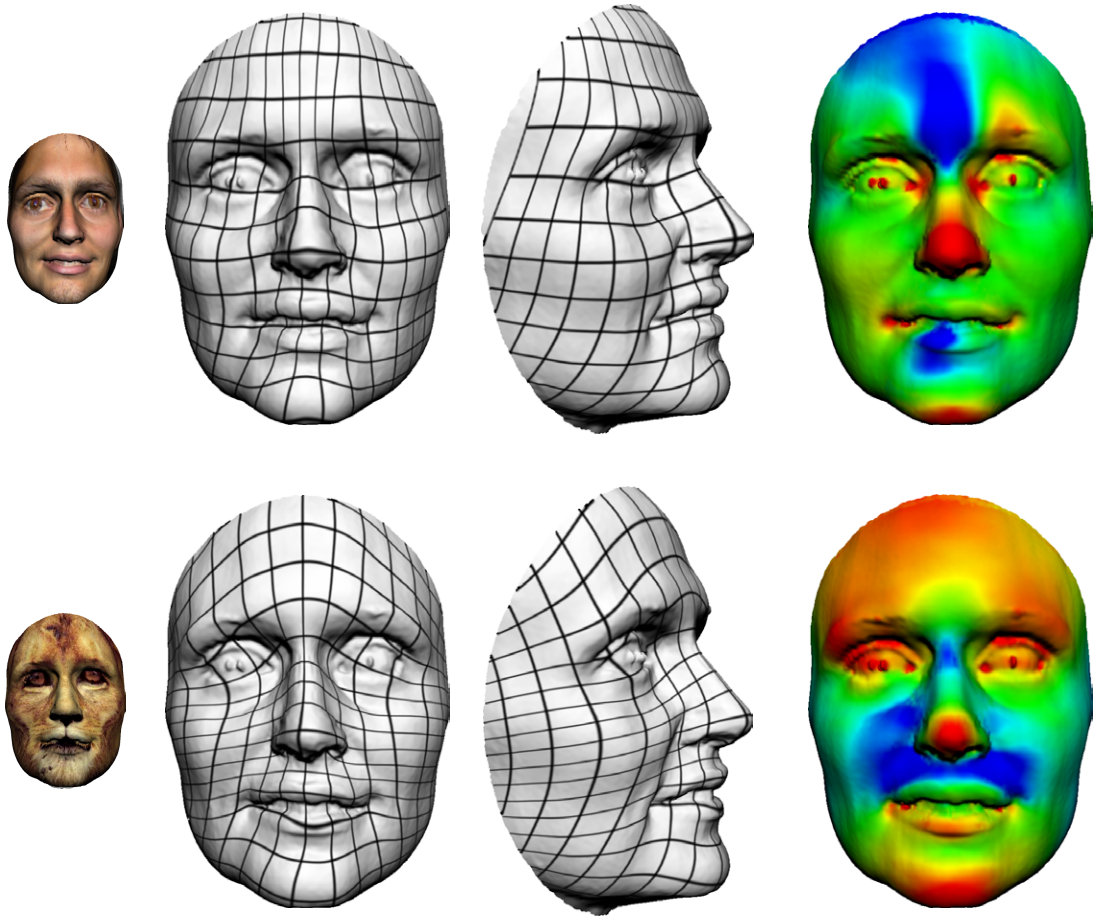


Figure 7.10 Illustration of local stretching for the texture mapping examples of Figure 7.9. The image on the right shows the color-coded first derivative of the parameterization, where blue denotes minimum absolute value and red denotes maximum absolute value.

#Levels	Cluster size	Initialization	Solving	Total
1	-	4.93	27.81	32.74
2	2	6.97	24.32	31.29
2	6	5.71	5.06	10.76
2	10	5.53	4.68	10.21
2	14	5.29	4.59	9.88
4	2	8.37	4.55	12.92
4	6	5.74	3.96	9.70
4	10	5.56	3.81	9.37
6	2	8.64	3.85	12.49
6	4	5.71	3.82	9.53

Table 7.1 Timing data for computing the minimum distortion parameterization in Figure 7.9 using nested iteration with different numbers of hierarchy levels and cluster sizes. All timings are given in seconds on a 2.0 GHz Pentium IV.

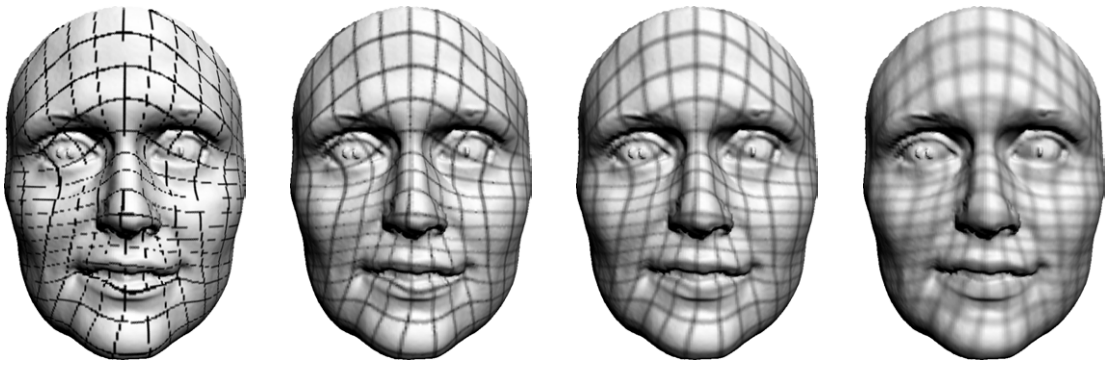
adapt the parameter mapping. The same cluster hierarchy can be re-used and changes of the constraints can be computed efficiently using incremental updating. Thus the computational overhead during interactive editing is dominated by the solving stage, leading to increased overall performance.

7.3 RE-SAMPLING

The mapping function Φ defines a parameter value $\mathbf{u}_i$ for each $\mathbf{p}_i \in P'$ of the selected surface patch S'. To create a one-to-one correspondence between the surface samples $\mathbf{p}_i \in P'$ and the brush $\mathbf{b}_{mn} \in B$, either the surface or the brush needs to be re-sampled. Note that the brush samples are implicitly parameterized, i.e., each $\mathbf{b}_{mn}$ has parameter coordinates $(m/M, n/N) \in [0, 1]$.

7.3.1 Re-sampling the Brush

Re-sampling the brush is performed by evaluating the continuous brush function represented by the discrete brush bitmaps at the parameter values of the sample points of the surface patch. The most simple reconstruction uses piecewise constant basis functions which amounts to nearest neighbor sampling. As Figure 7.11 (a) illustrates, this can lead to severe aliasing artefacts, which can be reduced by applying a Gaussian low-pass filter to the brush function prior to sampling. This filtering is analogous to the elliptical weighted average (EWA) filtering used in point rendering and the reader is referred to [119] for further details. Note that if the model surface is sampled substantially less dense than the brush, Gaussian filtering will lead to significant blurring. If the brush contains high-frequency detail, this information cannot be accurately mapped onto the surface without introducing new samples into the model.



(a) (b) (c) (d)

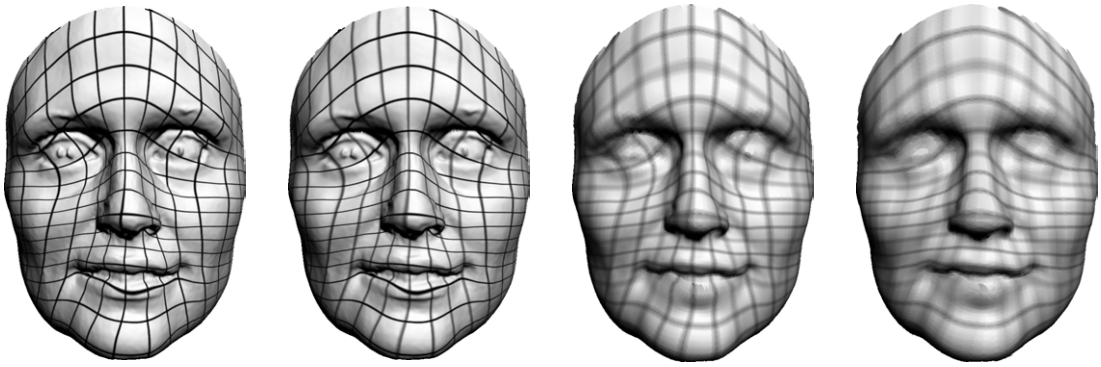
Figure 7.11 Re-sampling the brush on a model consisting of 40,880 points. (a) nearest-neighbor sampling, (b) - (d) Gaussian filtering with increasing filter width.

7.3.2 Re-sampling the Surface

To overcome the problem of information loss when re-sampling the brush, an alternative sampling method re-samples the surface to exactly map the $M \times N$ sampling grid of the brush. For this purpose a parameterized scattered data

approximation is used. This method reconstructs a continuous surface from the discrete sample points, which can then be sampled at the parameter values of the brush samples. The idea is to compute local polynomial fitting functions at each $\mathbf{p}_i \in P'$ that are blended using a mapping from the local parameter plane of the fitting functions into the global parameter space of the surface (see [118] for details).

Figure 7.12 shows various examples of editing operations where the surface has been re-sampled according to the sampling grid of the brush. If the sampling resolution of the brush decreases, surface detail is lost. This is complementary to the situation described above, where brush information was lost due to insufficient sampling density of the model surface.



(a) (b) (c) (d)

Figure 7.12 Re-sampling the surface. The sampling resolution of the brush varies from 500×500 pixels in (a), 250×250 pixels in (b), 100×100 pixels in (c), to 50×50 pixels in (d).

7.4 SURFACE EDITING

The re-sampling method of Section 7.3 provides samples of the surface $S_\Psi = \Psi(\Phi(S))$ and of the brush $B_\Psi = \Psi(B)$ with identical sampling distribution. Thus the two can be combined by applying an editing operator directly on the discrete coefficients. Note that both S_Ψ and B_Ψ represents all the sample attributes shown in Figure 7.4 (a). Depending on the intended functionality, an editing operator will then manipulate a subset of these surface attributes, such as diffuse color or spectral coefficient. In the following some of the editing operators will be described that have been implemented into the Pointshop3D system (see Chapter 8). A prime will denote the manipulated attributes, e.g., $\mathbf{x}_i'$ describes the position of a sample of the edited surface. Quantities that stem from the brush B_Ψ are marked with a bar, e.g., $\bar{\mathbf{c}}_i$ is the diffuse color of a brush sample. All other variables are part of the surface function S_Ψ.

7.4.1 Painting

Painting operations modify surface attributes by alpha-blending corresponding surface and brush coefficients. For example, the diffuse color can be altered by applying the painting operator on the color values, i.e.,

$$\mathbf{c}_i' = \overline{\alpha}_i \cdot \mathbf{c}_i + (1 - \overline{\alpha}_i) \cdot \overline{\mathbf{c}}_i, \tag{7.10}$$

where $\overline{\alpha}_i$ is an alpha value specified in the brush function (see Figure 7.13 (a)). Similarly, painting can be applied to other attributes, e.g., reflectance coefficients (Figure 7.16 (a)).

(a) (b)

Figure 7.13 Editing operations on the Chameleon (101,685 points): (a) texture painting, where the flowers shown on the left have been alpha-blended onto the surface, (b) texture filtering, where an oil-paint filter has been applied to the left half of the model.

7.4.2 Sculpting

The system supports two variations of sculpting operations that modify the geometry of the surface. The first option is to apply *normal displacements* to the sample positions, i.e.,

$$\mathbf{x}_i' = \mathbf{x}_i + \overline{d}_i \cdot \mathbf{n}_i, \tag{7.11}$$

where $\overline{d}_i$ is a displacement coefficient given in the brush function. As illustrated in Figure 7.15 (a), this type of editing operation is particularly suitable for embossing or engraving. On the other hand, a *carving* operation is motivated by the way artists work when sculpting with clay or stone. It implements a "chisel stroke" that removes parts of the surface in the fashion of a boolean intersection (see also Section 6.1). The editing tool is defined with respect to a reference plane that is specified by the surface normal of the touching point and an intrusion depth. The new sample position is then given by

$$\mathbf{x}_i' = \begin{cases} \overline{\mathbf{b}}_i + \overline{d}_i \cdot \overline{\mathbf{n}} & |\mathbf{x}_i - \overline{\mathbf{b}}_i| < \overline{d}_i \\ \mathbf{x}_i & \text{otherwise} \end{cases}, \tag{7.12}$$

where $\overline{\mathbf{b}}_i$ is the base point on the reference plane and $\overline{\mathbf{n}}$ the plane normal (see Figure 7.14). Carving operations can also be applied to rough surfaces (see Figure 7.15 (b)), where normal displacements fail due to the strong variations of the surface normals.

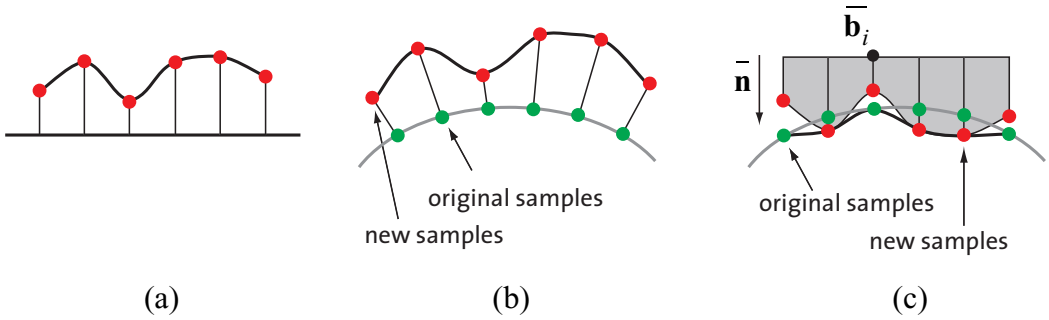


Figure 7.14 Normal displacements vs. carving. (a) brush image, (b) normal displacement, (c) carving.

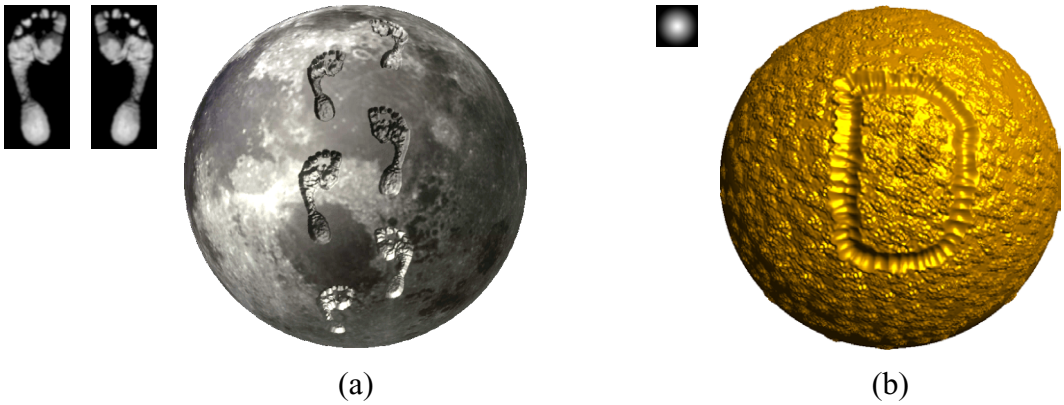


Figure 7.15 Editing operations. (a) normal displacements, (b) carving on a rough surface. The gray-scale images on the upper left show the displacement coefficients of the brush.

7.4.3 Filtering

Filtering is a special kind of editing operation that modifies the samples of the original model using a user-specified filter function f. First, the filter function is applied to S_Ψ yielding $S_\Psi^f = f(S_\Psi)$. Then filtered and original attributes are combined using the brush function for alpha blending. As an example, consider texture filtering, i.e.,

$$\mathbf{c}_i' = \bar{\alpha} \cdot \mathbf{c}_i^f + (1 - \bar{\alpha}) \cdot \mathbf{c}_i, \tag{7.13}$$

where $\mathbf{c}_i^f$ is the filtered color value (illustrated in Figure 7.13 (b)). The filter function is usually implemented as a discrete convolution. Therefore, arbitrary discrete linear filters can be implemented by simply choosing the appropriate kernel grid. Filters can be applied to any attribute associated with a sample, e.g., color, normal, or distance from the reference plane for geometric offset filtering (Figure 7.16 (b)). Note that filtering with large kernels can be implemented efficiently in the spectral domain, similar to [91].

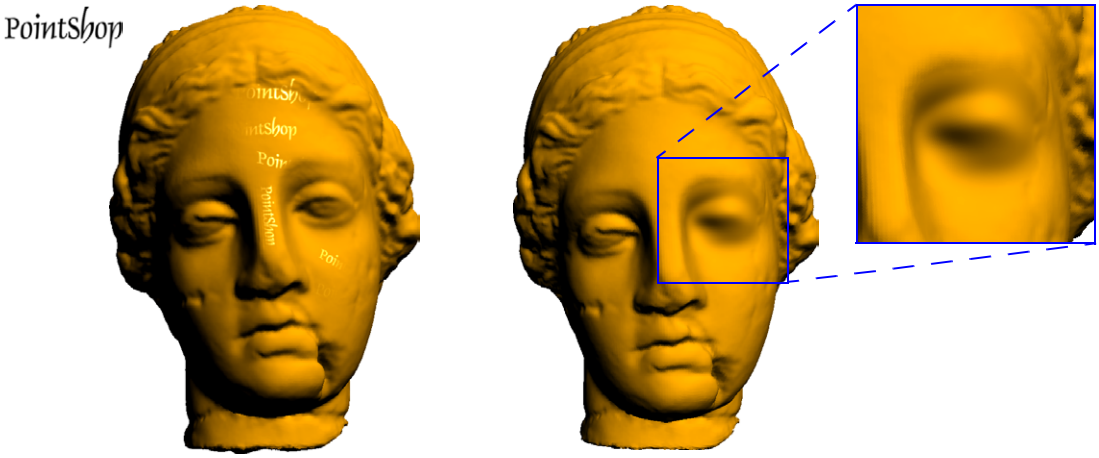
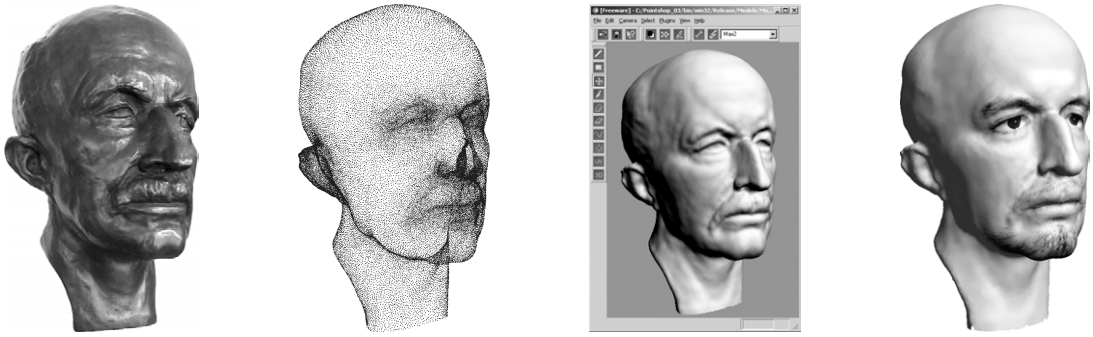


Figure 7.16 Editing operations on the Igea model (134,345 points): (a) modifying the specular coefficient: the diffuse surface has been painted with a specular text, (b) geometry filtering, where the region around the eye has been smoothed by a geometric low-pass filter.

8

POINTSHOP3D



This chapter introduces Pointshop3D, a software platform for point-based shape and appearance modeling [1, 118]. All algorithms presented in this thesis have been implemented and analyzed within this framework. A brief description of the main components is followed by a more detailed discussion on data structures for spatial queries. The chapter concludes with some examples of point-based surfaces that have been created or edited in Pointshop3D, using the algorithms described in the previous chapters.

8.1 SYSTEM OVERVIEW

Pointshop3D is designed as a modular software platform and implementation test-bed for point-based graphics applications. It provides a set of kernel functions for loading, storing, modifying, and rendering point-sampled surfaces. Most of the functionality is implemented in various tools and plug-ins that are briefly discussed below. A more detailed description of the software architecture and user interface can be found in [63] and in the online documentation available at [1].

8.1.1 Tools and Plug-ins

The interaction metaphor of Pointshop3D is similar to common photo editing tools such as Adobe's Photoshop. Figure 8.1 shows a screen shot of the main user interface, where the most important interaction tools are annotated:

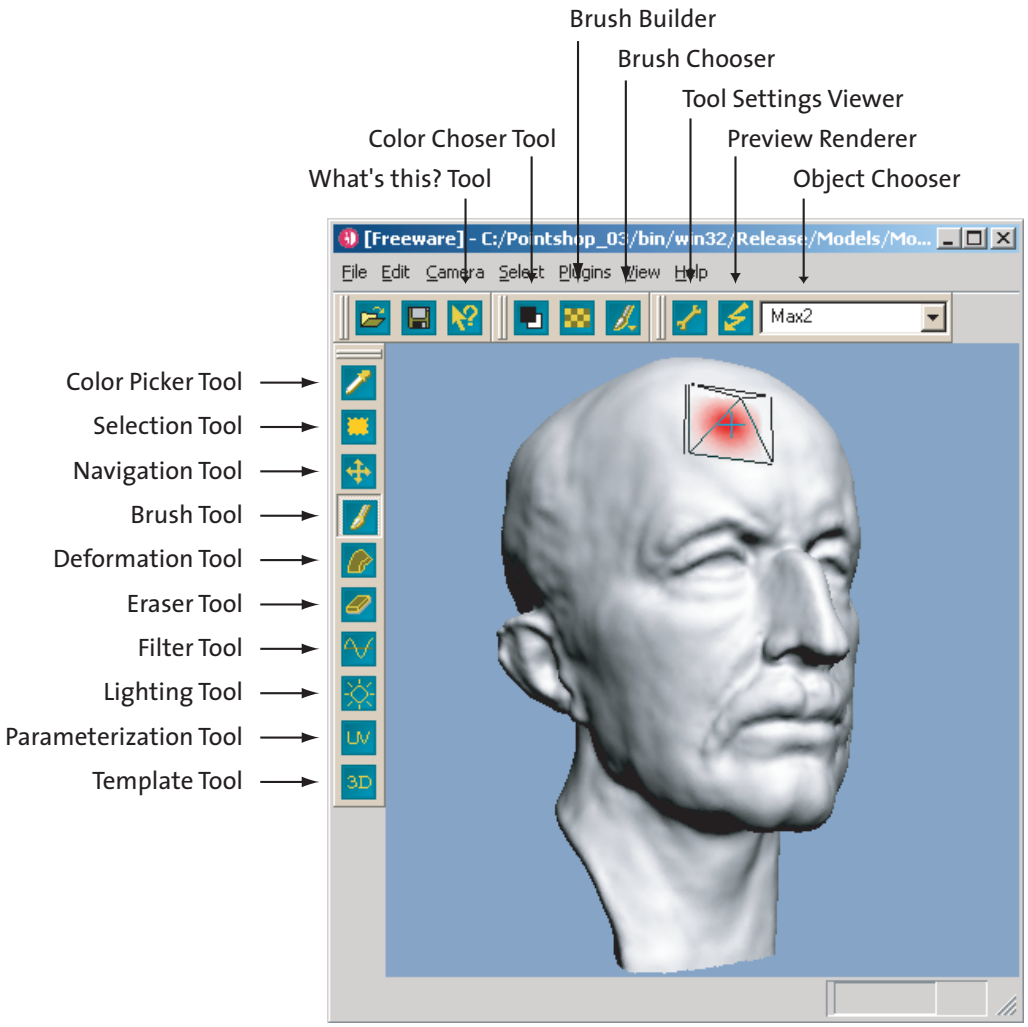


Figure 8.1 The main Pointshop3D interface.

- The **color picker tool** allows to extract sample attributes from the model.
- The **selection tool** is used to select parts of the surface, e.g., to create a patch for selection interactions (see Section 7.1.2).
- The **navigation tool** controls the camera position and the relative orientation of different objects with respect to the world coordinate system.
- The **brush tool** implements brush interactions (see Section 7.1.2).
- The **deformation tool** supports free-form deformation as described in Section 6.2.
- The **eraser tool** allows to remove parts of the surface.
- The **filter tool** implements various filters (see Section 7.4.3).

- The **lighting tool** controls the position of the light source.
- The **parameterization tool** allows the specification of point constraints for computing a minimum distortion parameterization (see Section 7.2.2).
- The **template tool** is a place-holder that supports easy extensions of the system by adding new tools.
- The **What's this? tool** provides an online help.
- The **color chooser tool** allows to select an rgb color from a palette for brush painting.
- The **brush builder** is used to create new brushes (see Figure 7.4) for brush interactions.
- The **brush chooser** stores a list of pre- or user-defined brushes used for brush interactions.
- The **tool settings viewer** displays additional parameters of the currently active tool.
- The **preview renderer icon** activates the preview renderer (see Section 8.1.2).
- The **object chooser** selects a certain object within the scene.

Each of these tools can be configured as described in the online documentation of Pointshop3d [1]. Additional functionality, e.g., the boolean operations defined in Section 6.1, or the multi-scale decomposition of Section 4.2, is implemented as plug-ins that are accessible via the main menu. Tools and plug-ins are implemented as dynamic link libraries and can be loaded dynamically at run-time, which makes the system easily extensible (see [63]).

8.1.2 Rendering

Pointshop3D supports a number of different point-based renderers that can also be dynamically loaded at run-time. Most relevant for this thesis are the following three renderers:

- A purely software-based EWA surface splatting renderer [119] is used for high quality renditions of still images and for off-line computations of animated sequences. Most images in this thesis have been created with this renderer, see for example Figures 8.5 to 8.11.
- A fast OpenGL renderer is used during navigation, e.g. when the object is scaled, rotated or zoomed. The OpenGL renderer uses rounded, rectangular splats, whose size is dynamically computed during rendering using vertex shader programming (see [1]).
- A non-photorealistic renderer to create line drawings of extracted feature curves as described in Section 5.5.

Figure 8.2 shows renditions produced by the three different renderers. In general, the rendering quality of the surface splatting renderer is significantly higher as compared to the OpenGL renderer, in particular for models containing high-frequency textures as shown in Figure 8.2 (d) and (e). A detailed analysis of point-based rendering algorithms can be found in [117].

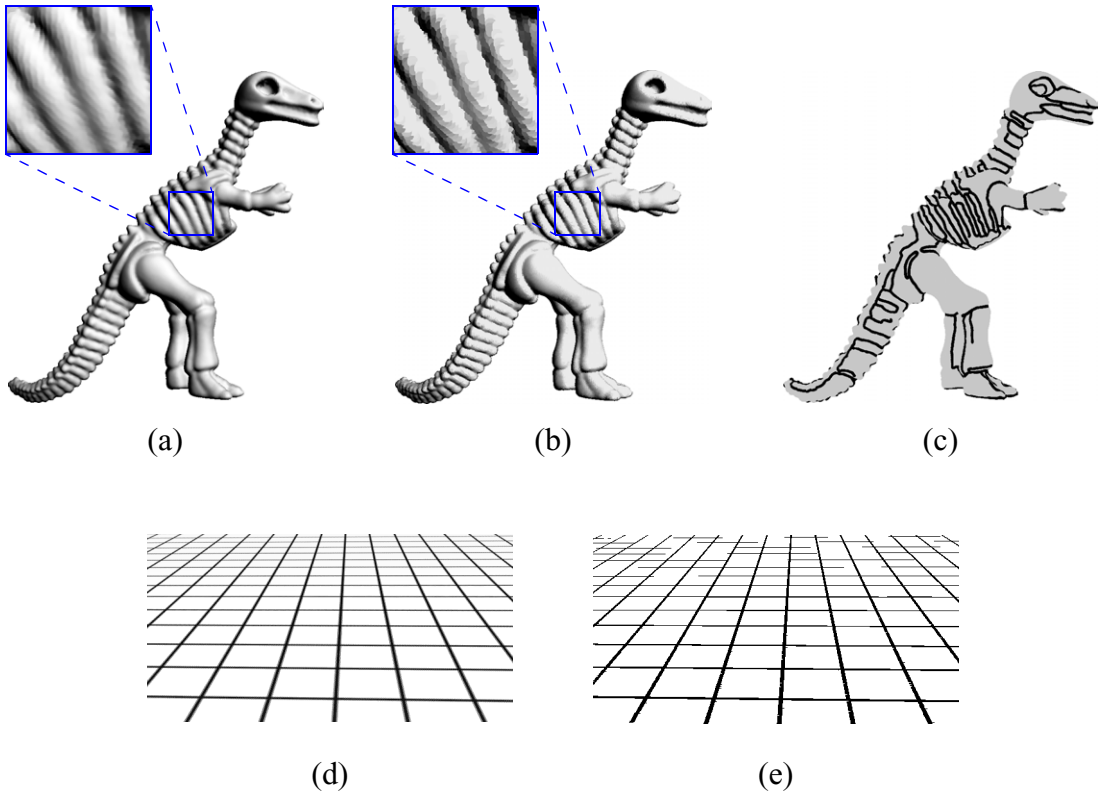


Figure 8.2 Renderers used in Pointshop3D. Top row: Three different renditions of the dinosaur model: (a) surface splatting renderer, (b) OpenGl slpat renderer, (c) feature-based line renderer, bottom row: texture filtering: (d) surface splatting, (e) OpenGL renderer.

8.2 DATA STRUCTURES

The central computational primitive required by the algorithms described in the previous chapters is closest points query: Given a point $\mathbf{x} \in \mathbb{R}^3$ find the point(s) $\mathbf{p}_i \in P$ such that $\|\mathbf{x} - \mathbf{p}_i\|$ is minimal. Closest point queries occur in two different contexts:

- The neighborhood relations defined in Section 2.1.1 are based on the k-closest points of a sample in P. In this case the query point $\mathbf{x}$ is part of the set of point samples, i.e., $\mathbf{x} \in P$. The number of required closest points ranges from 8 to 20 for, e.g., normal estimation (Section 2.1.3), to up to about 200 for multi-scale variation estimation for feature classification (Section 5.3.1).
- Spatial queries are also required for the MLS projection (Section 2.3.2), boolean classification (Section 6.1.1), and collision detection (Section 6.2.1). Here the query point can be arbitrary, i.e., $\mathbf{x} \notin P$ in general, and typically only a single closest point is required.

The performance of the algorithms presented in this thesis critically depends on efficient data structures for such queries. The requirements for these data structures vary considerably, depending on the specific context. For example, multi-scale feature classification (Section 5.3.1) applies nearest neighbor queries with varying neighborhood size for static objects, while particle simulation (Section 3.4) requires

closest point queries for completely dynamic point distributions. Other operations, e.g., the filtering methods applied during free-form deformation (Section 6.2.3) operate on models with dynamic point locations, but mostly constant neighborhood relations (see also the discussion in Section 8.2.4).

In the design of appropriate data structures, one is typically faced with a trade-off between efficient construction/updating of the data structure and fast query response [85]. In this thesis two data structures, kd-trees and dynamic grids, have been implemented. Kd-trees feature fast query response, yet perform poorly when new points need to be inserted or existing points updated or removed. Dynamic grids provide efficient updating at the expense of slower query response times.

8.2.1 Kd-Trees

Bentley introduced kd-trees as a generalization of binary search trees in higher dimensions [11]. A number of improvements have been proposed since then, see for example [36, 105]. Kd-trees are binary trees, where each node is associated with a cubical cell and a plane orthogonal to one of the three coordinate axes. The plane splits the cell into two sub-cells, which correspond to two child nodes in the tree. The cells of the leaf nodes are called buckets and represent the whole space.

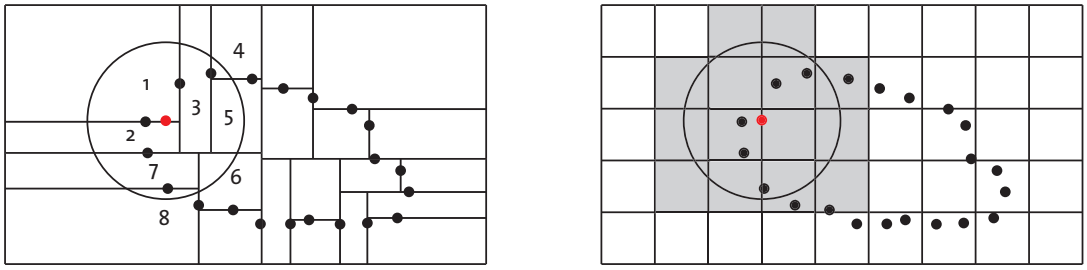
In Pointshop3D, kd-trees are built in a recursive top-down approach, where the sliding-midpoint rule is used to determine the split axis (see [9]). Average time complexity for the construction of the tree is $O(n\log n)$ for a point cloud consisting of n sample points. To compute the k-closest points from a given location $\mathbf{x} \in \mathbb{R}^3$, the tree is traversed from the root until the bucket containing $\mathbf{x}$ is found. From this leaf node, the k-closest points are determined by back-trapping towards the root. Finding the k-closest points takes $O(k + \log n)$ time on average, insertions and updates of single points require $O(\log n)$ time [99]. A detailed analysis of the kd-tree implementation of Pointshop3D can be found in [61].

8.2.2 Dynamic Grids

Dynamic grids have been implemented as a data structure for closest point queries in dynamic settings. The idea is to subdivide space into a regular grid of cubical cells, each of which stores a list of all points that fall within its cell [50]. Range queries can be performed by scan-converting the query sphere to determine all cells that potentially contain samples located within the query range. Each of these cells is then processed by testing all points of the cell against the query sphere. To accommodate variations of point density over time, the grid can be re-structured dynamically as described in [50].

Finding the correct grid cell for a point $\mathbf{x} \in \mathbb{R}^3$ that lies within the object's bounding box takes $O(1)$ time. Thus a dynamic grid for n samples can be constructed in $O(n)$ time and the insertion and updating of a single point takes $O(1)$ time. Since each grid cell stores a linear list of points, the average time complexity of a query is $O(m)$, where m is the average number of points per cell. Note that a linear list of all sample points is a special case of a grid data structure consisting of only one grid cell.

Figure 8.3 shows a 2D illustration of the kd-tree and grid data structures. Table 8.1 compares the average time complexity of these data structures for typical operations used in the algorithms of this thesis.



(a) (b)

Figure 8.3 Spatial data structures for closest point queries, where the red dot indicates the query location and the black circle shows the query sphere. (a) kd-tree, where the numbers indicate the order in which the buckets are traversed, (b) dynamic grid, where the gray cells mark the intersection with the query sphere.

Data Structure	Construction	Insertion	Update	Query
List	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Grid	$O(n)$	$O(1)$	$O(1)$	$O(m)$
Kd-tree	$O(n\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$

Table 8.1 Average time complexity for spatial data structures used for closest point queries. n is the number of points in P, m is the average number of points in each grid cell. Update refers to a change of sample position or a sample point deletion.

8.2.3 Neighborhood Caching

For many processing methods the k-closest points of a sample point $\mathbf{p} \in P$ have to be determined to establish local neighborhood relations (see Section 2.1.1). Often the point cloud is static or semi-dynamic, i.e., point locations vary but neighborhood relations remain (mostly) constant. If neighborhood relations have to be used multiple times, the efficiency of the computations can be improved considerably by caching these neighborhoods. Each sample point explicitly stores a list of its k-nearest neighbors computed using a kd-tree or a dynamic grid. This defines an adjacency graph that is comparable to a connectivity graph for triangle meshes. Unlike the latter, the adjacency graph does not define a consistent manifold surface, however. Thus the construction of the adjacency graph is significantly more efficient than the reconstruction of a triangle mesh from a given point cloud.

8.2.4 Spatial Queries in Pointshop3D

This section discusses the use of the data structures described above in the algorithms presented in the previous chapters.

Surface Simplification

Spatial data structures for surface simplification have already been discussed in Section 3.5.4. Incremental clustering (Section 3.2.1) uses kd-trees for nearest neighbor

queries, since the point cloud is static. Hierarchical clustering (Section 3.2.2) builds a BSP-tree and does not require closest point queries. Iterative simplification (Section 3.3) builds an explicit adjacency graph using kd-trees during initialization, which is maintained during simplification. Particle simulation (Section 3.4) makes use of dynamic grids, since individual particles experience significant drifts along the surface, which leads to dynamically changing point neighborhoods.

Boolean Classification

For boolean operations (Section 6.1), two static objects are positioned relative to each other by the user. Since the resulting affine transformation can be directly incorporated into the query, no updates of point locations are required. Thus closest point queries for classification (see Section 6.1.1) are implemented using kd-trees.

Free-Form Deformation

For free-form deformation (Section 6.2) closest point queries are required to evaluate the distance functions d_j (see Equation 6.2) for computing the scale factor t. This is done once at the beginning of the modeling session, so no dynamic updates are required while the user interactively deforms the surface. Thus a kd-tree is most suitable for scale factor computation.

Collision Detection

For collision detection (Section 6.2.1) the classification predicate Ω (see Equation 6.1) has to be evaluated for dynamically varying point locations. Since performance is most critical, the algorithm only detects collisions of the deformable region with the zero-region. Since the latter is described by a static point cloud, kd-trees can be used for efficient closest point queries. Thus the collision detection algorithm in its current version cannot detect collisions of the deformable region with itself.

Particle-Based Blending

The blending method based on oriented particles (Section 6.2.1) uses the dynamic grid data structure. Since the particle simulation is mostly used to smooth out the sharp creases created by boolean operations (see Figure 6.9), particles experience a small spatial displacement during the simulation. Thus the neighborhood relations typically remain constant for a certain number of iterations. This can be exploited to improve the performance by caching the neighborhoods as discussed above.

Dynamic Re-sampling

When dynamically sampling the surface during deformation (Section 6.2.2), the relaxation and interpolation filters (Section 6.2.3) require nearest neighbor information for samples in the deformable region. Since these samples vary in position and new samples are inserted dynamically, kd-trees cannot be used efficiently. The dynamic grid data structure has also been found to be unsuitable, due to the relatively high cost of a spatial query.

The relaxation filter shifts the point locations only slightly, which typically does not change the neighborhood relation significantly. Therefore, neighborhoods are computed at the beginning of the interaction and cached during deformation (see above). When new points are inserted, neighborhoods have to be updated, however.

Since new samples are created by splitting an existing sample into two, the corresponding neighborhood relations can be propagated from the parent sample to its child samples. To maintain neighborhood consistency, each sample stores bi-directional neighborhood relations, i.e., a list of its neighbors and a list of samples of which itself is a neighbor (note that with the k-nearest neighbor relation (see Section 2.1.1), $j \in N_{p_i}$ does not imply $i \in N_{p_j}$).

A different dynamic sampling method has been used in Section 7.3.2 for appearance modeling. Here a whole patch of a surface is replaced by a set of new points that are sampled according to the grid of the currently active brush. Since the regular lattice of the brush defines an implicit neighborhood relation of the sample points, no additional spatial data structure is required for nearest neighbor queries.

8.3 EXAMPLES

This sections shows examples of surfaces that have been created or edited using the shape and appearance modeling functionality introduced in the previous chapters (see Figure 8.4).

Model	#points in final model	MLS smoothing	multi-scale	boolean operations	free-form deformation	particle blending	texturing	sculpting	time of modeling session in minutes
Male	148,138		●		●				15
Igea	152,819		●	●	●				15
Max Planck	25,020			●	●				10
Octopus	295,220				●		●	●	35
Spiral	69,268			●	●				10
Coffee Mug	222,955			●	●	●	●	●	25
Gnome	100,269	●		●	●		●		30

Figure 8.4 Statistics of the example models of Figures 8.5 to 8.11.

Figures 8.5 and 8.7 show shape modeling operations using the multi-scale representation introduced in Chapter 4. The deformations have been applied on the smooth base shape and high-frequency detail has been reconstructed after the deformation. In Figure 8.7, the detail coefficients have also been scaled to achieve an enhancement effect as described in Section 4.4.

In Figure 8.6 the handle of the coffee mug has been created using the deformation tool in connection with a boolean union operation and particle-based blending as described in Section 6.2.1. The interior of the mug has been cut out with a boolean

difference and the dragon head has been added using a union operation. Finally, the Siggraph logo has been embossed and the model has been textured.

Figure 8.8 shows boolean operations with two sparsely and non-uniformly sampled models, illustrating that the methods work well for a wide class of input models. First the skull has been deformed to better match the shape of the head. Then a boolean difference of the head with a plane, and a subsequent union and difference with the skull have been performed.

Figure 8.9 shows the model of the Octopus, whose shape has been created from a single sphere entirely using the free-form deformation tool (see Section 6.2). First a global deformation has been applied to create an ellipsoid, then the tentacles have been pulled out using a similar deformation as the one shown in Figure 6.17. The eyes and suckers have been added using displacement mapping (Section 7.4.2) and the model has finally been textured using the paint tool of Pointshop3D (Section 7.4.1). This example illustrates that dynamic re-sampling is essential when dealing with large deformations. The initial sphere contains 69,706 points, while the final model contains 295,220 points.

Figure 8.10 illustrates that the sharp intersection curves created by a boolean difference operation are well preserved during a subsequent deformation.

Figure 8.11 demonstrates that the point-based shape and appearance modeling framework is well suited for scanned surfaces. The smoothing effect of the MLS projection (see Section 2.3.2) has been used to de-noise the input data set. After this minimal pre-processing, free-form deformations and boolean operations can be directly applied to the acquired point cloud. This example also illustrates a typical cut-and-paste operation. The ear has been extracted from the Max Planck model and pasted onto the scanned surface using a boolean union. After modeling the shape of the object, a texture mapping operation using the constrained minimum distortion parameterization (see Section 7.2.2) has been applied.

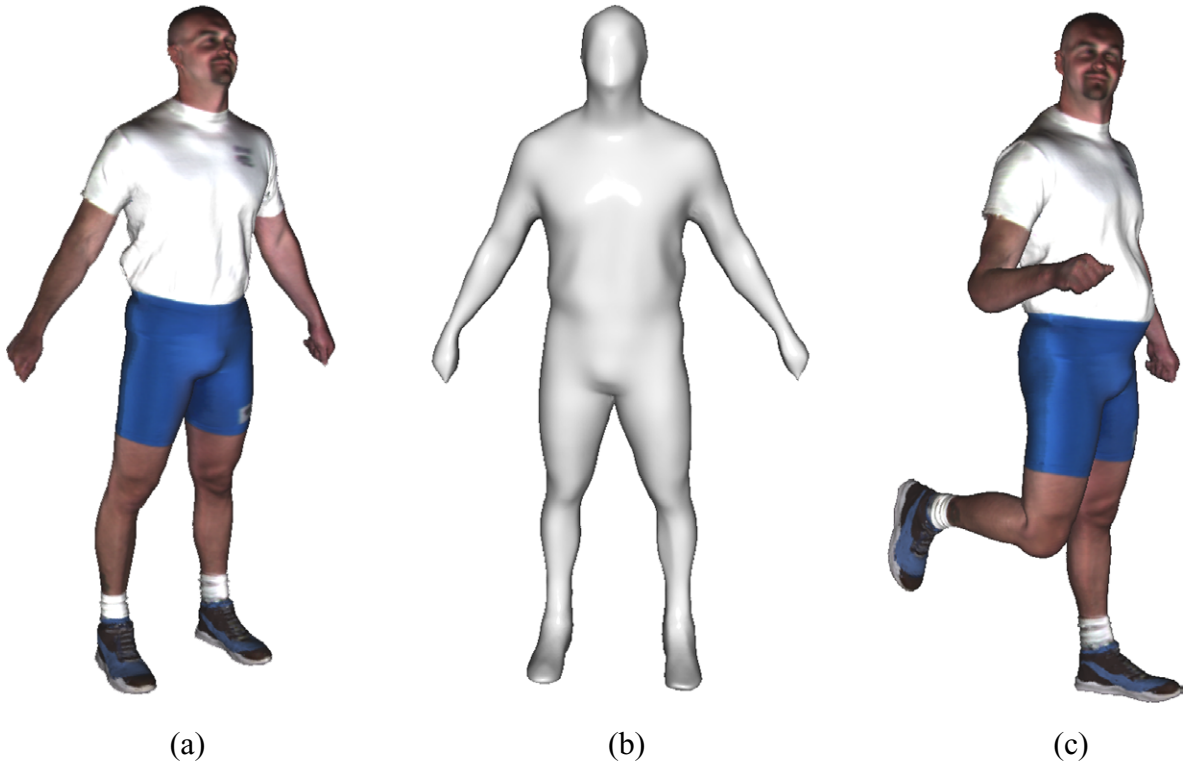


Figure 8.5 Multi-scale modeling on a human body. (a) original, (b) smooth base domain, (c) final deformed surface.

Figure 8.6 Creating a coffee mug using boolean operations, deformation, collision detection and particle-based blending.

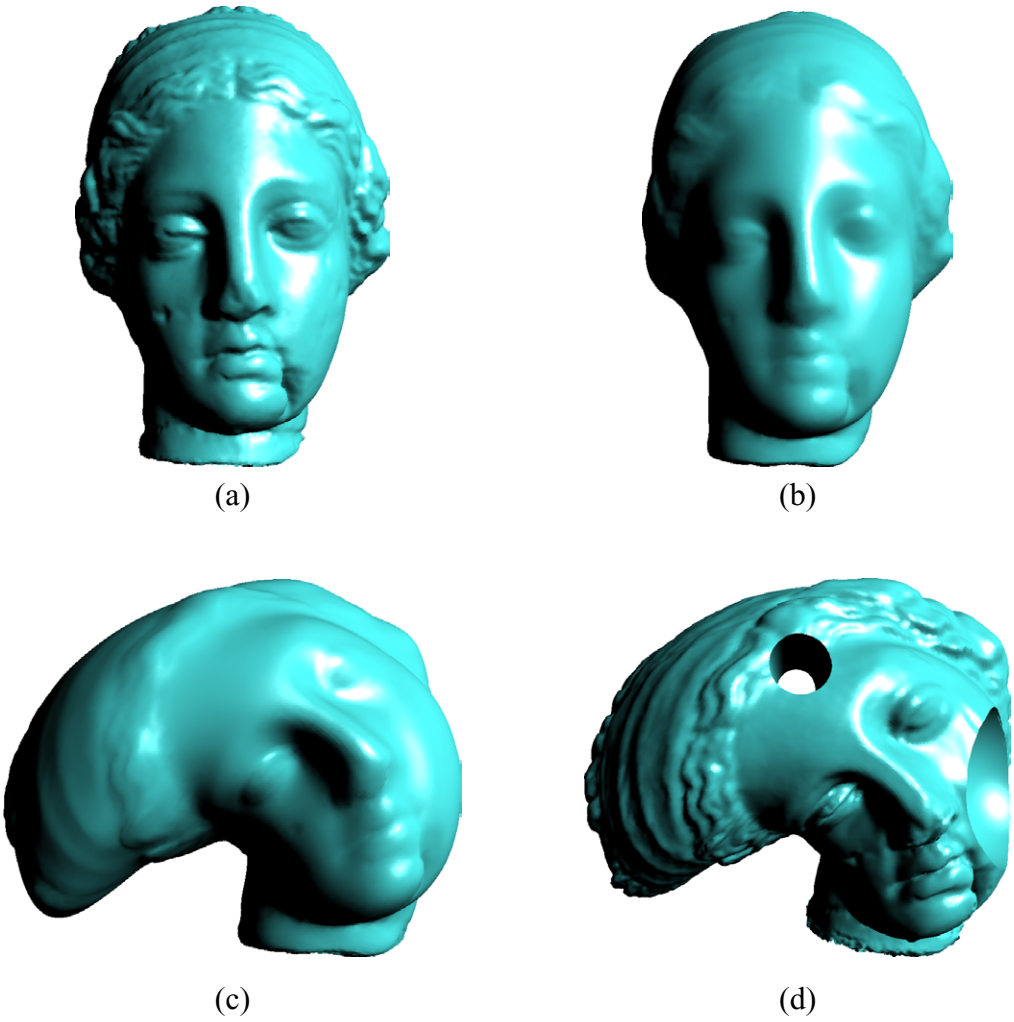


Figure 8.7 Multi-scale modeling on the Igea model. (a) original model, (b) smooth base domain, (c) deformed base domain, (d) final model, where the displacement coefficients have been scaled by a factor of two to achieve an enhancement effect. Additionally, two boolean difference operations have been applied.

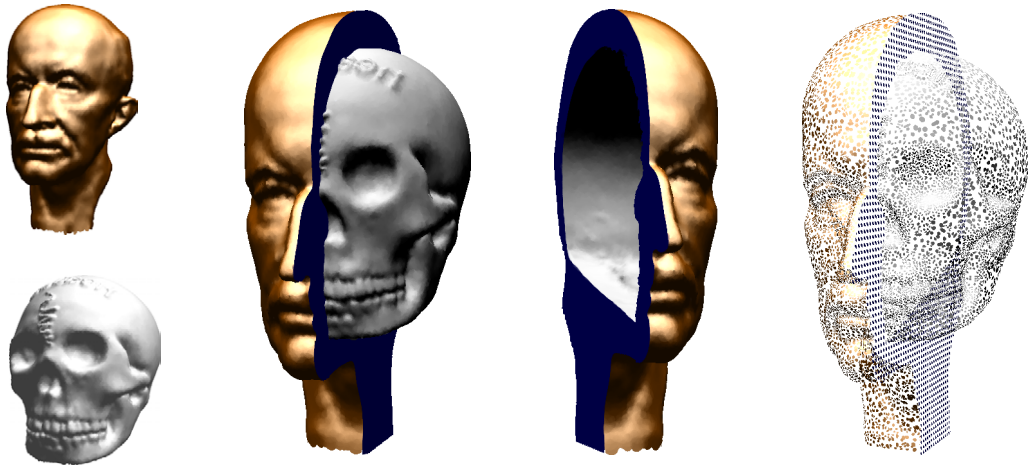


Figure 8.8 Boolean operations of the Max Planck bust, a plane and the skull model.

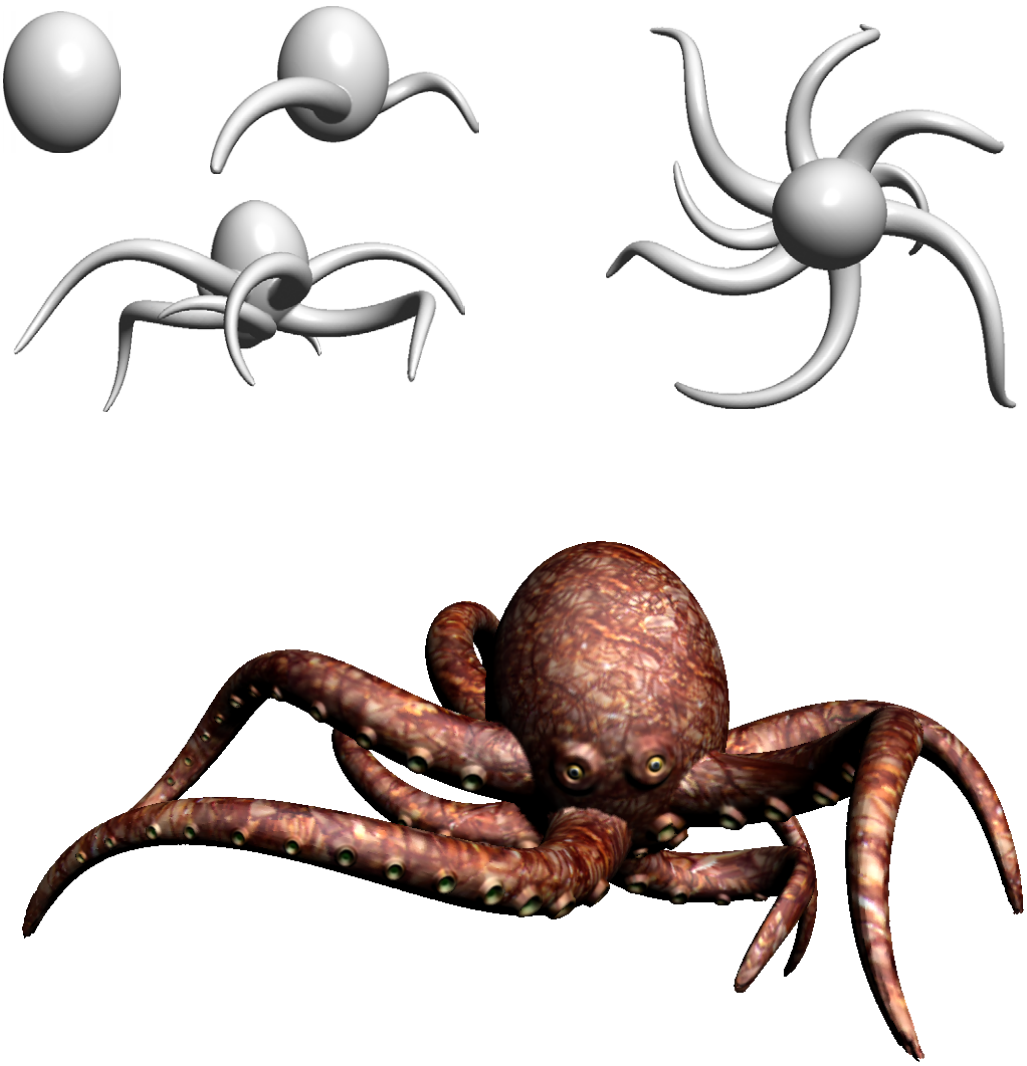


Figure 8.9 Creating an Octopus from a sphere using the deformation tool with dynamic sampling.

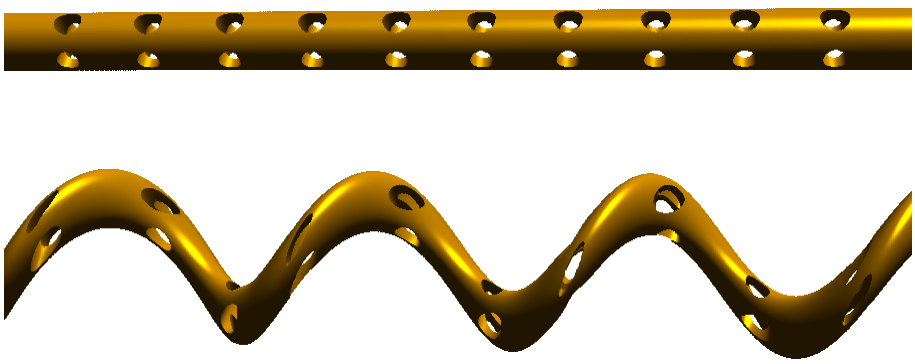


Figure 8.10 Combination of boolean operations and subsequent deformation.

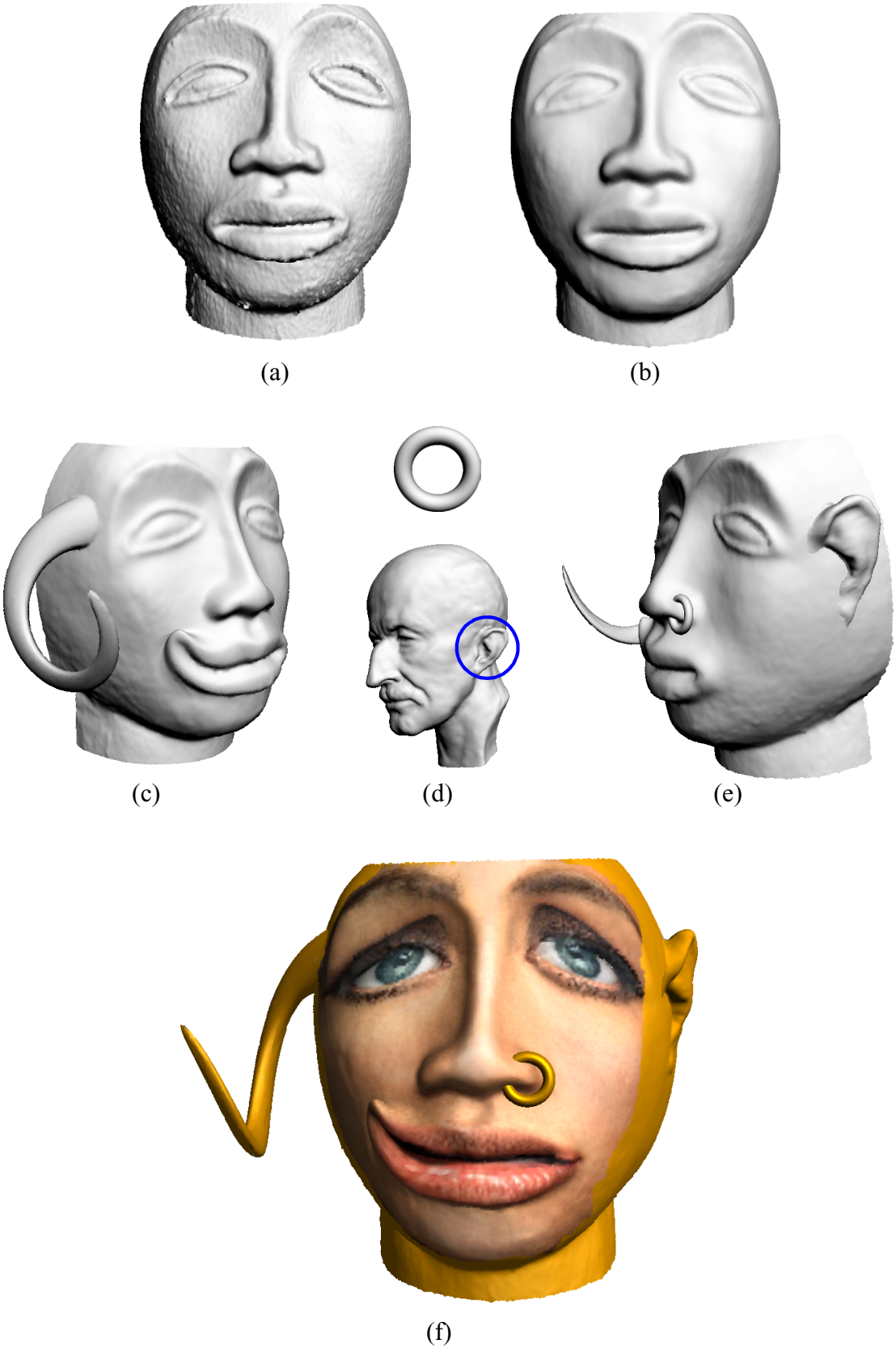


Figure 8.11 Boolean operations and deformations on scanned data: (a) noisy range scan, (b) surface smoothed by MLS projection, (c) surface after local deformations, (d) objects used for boolean union, (e) surface after boolean union, (f) final textured surface.

9

CONCLUSIONS

This chapter concludes the dissertation with a summary of the main contributions, a discussion of advantages and drawbacks of the chosen approach, and some ideas for future research.

9.1 PRINCIPAL CONTRIBUTIONS

The main focus of this thesis is the use of point primitives for digital geometry processing. In this context, the following contributions have been made:

- Methods for **local surface analysis** of point-sampled models based on k-nearest neighbors have been discussed and analyzed. A theoretical analysis of sampling criteria has been given and suitable bounds for k have been formulated. An extension of the moving least squares surface model has been presented that adapts to the local sampling density.

- Various **simplification algorithms** for reducing the complexity of point-sampled surfaces have been introduced. Incremental and hierarchical clustering, iterative point contraction, and particle simulation methods have been implemented and analyzed.

- Based on the moving least squares surface model, a **multi-scale surface representation** has been presented that supports multi-scale modeling and discrete spectral filtering. To build such a representation, point-based surface fairing methods have introduced, as well as a multi-level decomposition operator for efficient detail encoding.

- The concept of scale-space has also been applied to implement a **feature extraction pipeline** for point-sampled surfaces. It has been shown that feature classification at multiple scales yields a robust feature detection method that faithfully recovers salient line-type features even for noisy surfaces.

- Two fundamental **shape modeling** approaches, boolean operations and free-form deformation, have been integrated into a unified geometric modeling system. This required a method to represent, sample and render sharp creases and corners in point-sampled surfaces and a dynamic sampling strategy for large model deformations. To support and control changes in topology during editing, a point-based collision detection algorithm has been incorporated into the system.
- A generic framework for **appearance modeling** has been defined that supports surface editing operations such as painting, texturing, sculpting and filtering. Core components of this framework are a new method for computing a constrained minimum distortion parameterization, a parameterized surface model for dynamic re-sampling, and a set of editing operators for point-sampled surfaces.

9.2 DISCUSSION

It has been demonstrated that point primitives are useful for advanced shape and appearance modeling operations. The structural simplicity of point-sampled surfaces leads to a number of advantages:

- No global combinatorial structure, such as a mesh connectivity graph, needs to be maintained. This allows efficient re-structuring of the surface during interactive modeling. Thus very large model deformations can be supported by dynamically inserting new sample points into the model.
- The moving least squares approximation provides a meshless, continuous surface model and gives access to the signed distance function. This enables efficient classification for boolean operations and accurate sampling of surface intersection curves. Additionally, the MLS projection operator can be utilized to determine point-to-surface distances for error measurement and detail encoding.
- Point primitives are a compact representation for discrete surfaces. This reduces memory traffic and leads to space and time efficient implementations.
- Many of the presented techniques can be directly applied to scanned data. In particular, local surface analysis allows to extract semantic information about the model from the point cloud data. Early processing methods such as simplification, segmentation, or filtering benefit from this information, which can lead to significant improvements in processing performance.
- Point samples provide a unified representation primitive for acquisition, processing and rendering of 3D objects. Thus the whole 3D content creation pipeline (see Figure 1.1) can be implemented using point primitives, eliminating the need for any conversion algorithms between different surface representations.

On the other hand, point-sampled surfaces have a number of limitations that need to be considered when evaluating their use for a specific geometric modeling application:

- Point-sampled surfaces need to observe a certain sampling density. When the spacing between adjacent samples becomes too large, topologically correct reconstruction might no longer be feasible without explicit connectivity information. For example, the surface simplification algorithms described in Chapter 3 give no guarantees that the simplified point cloud is still sampled adequately dense.

- The minimum sampling density is not only dependent on local surface properties, such as curvature, but also needs to take into account the distance to the medial axis, which is a global concept. In particular, the separation of two distinct sheets of the surface that are close in space requires a dense sampling, even though the curvature might be low.
- Many advanced geometry processing algorithms are based on consistent connectivity information. For example, subdivision surfaces are defined by an up-sampling operator and a geometric filtering operator, which requires consistent neighborhood relations. Thus many of the processing methods based on semi-regular meshes [106] are currently not available for point-sampled geometry.
- Graphics hardware is very much optimized for triangles as a rendering primitive. Even though point primitives are simpler to render, the performance of triangle-based renderers is still higher for comparable image quality.

In conclusion it can be stated that point-based representations are mostly suitable for densely sampled models stemming from 3D acquisition, iso-surface extraction or sampling of implicit functions. They are less suited for surfaces that have been carefully designed in a particular surface representation, such as low-resolution polygonal CAD data. Thus for industrial design or manufacturing applications, where mostly static surfaces are considered and geometric accuracy is of primary importance, point-based representations are inferior to existing methods such as NURBS or polygonal meshes. On the other hand, if appearance is more important than accuracy, or the shape of models needs to be modified significantly, point primitives are a viable alternative. This makes them most attractive for low cost 3D content creation, e-commerce and entertainment applications, rapid prototyping tools, or educational systems.

9.3 FUTURE WORK

Digital geometry processing comprises many other topics that have not been addressed in this thesis. For example, compression and progressive transmission of geometric data are of paramount importance for many applications [62]. With the increasing proliferation of geometric models, questions of determining ownership will also be of interest, requiring watermarking methods for authentication [93].

I believe that point primitives can be useful in other related fields, such as physical simulation. Particle systems are already widely used to model natural phenomena such as liquids [17], fire, or smoke [96]. Deformable objects are also gaining increasing attention, e.g., in medical simulation [109]. Point clouds could be an interesting representation for such models, in particular since volumetric aspects can easily be modeled using point primitives.

Special attention in computer graphics has always been given to complex real-world models, such as furry objects [42], or plants, such as trees or fern [95]. The methods presented in this thesis are not suitable for such models, since they are based on a smoothness assumption of the underlying continuous surface. For complex natural objects, a more “fuzzy” approach using statistical methods seems more appropriate. This could also incorporate procedural modeling algorithms or fractal methods.

Another promising idea is to combine point samples with other surface representations, e.g., implicit surfaces like level sets, to define hybrid surface models

[31]. With a careful design of such representations it should be possible to bring together the strengths and advantages of each component, while avoiding many of their drawbacks.

An interesting field that has not been addressed in this thesis is the design of user interfaces for 3D shape modeling. In particular, haptic devices in connection with physical simulation of material properties could provide a very powerful and intuitive mechanism for shape deformation.

The simplicity and compactness of point-based representation should allow for efficient hardware implementations. One could even imagine a customized point processing chip that integrates basic algorithms from the different stages of the 3D content creation pipeline.

In my opinion, the greatest challenge that still remains is the formulation of a rigorous sampling theory for discrete manifold surfaces. Ideally, one would want a generalization of Fourier theory to discrete geometry. The approach taken in Section 2.2 considers the intrinsic structure of discrete surfaces using concepts from computational geometry such as the Voronoi diagram or the medial axis. The derived sampling criteria give some first quantitative results, yet the gap between theory and practice is still significant. First provable theoretic bounds are much too pessimistic compared to empirical evidence obtained by practical experiments. Additionally, important aspects, such as entropy or noise, are not considered in this approach.

Maybe a combination of concepts from signal processing, approximation theory, and computational geometry yields a more general sampling theory for discrete manifolds.

A

REFERENCES

[1] Pointshop3D, http://graphics.ethz.ch/pointshop3d/

[2] A. Adamson and M. Alexa. Ray Tracing Point Set Surfaces. In *Proceedings of Shape Modeling International*, 2003.

[3] M. Alexa. Recent Advances in Mesh Morphing, In *Computer Graphics Forum*, 21(2), pages 173-196, 2002.

[4] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Point Set Surfaces. In *Proceedings of IEEE Visualization 2001*, pages 21–28. IEEE Computer Society Technical Committee on Computer Graphics, 2001.

[5] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Computing and Rendering Point Set Surfaces. In *IEEE Transactions on Visualization and Computer Graphics,* Vol. 9, Nr. 1, 2003.

[6] N. Amenta, M. Bern, and D. Eppstein. The crust and the beta-skeletion: combinatorial curve reconstruction. In *Graphical Models and Image Processing*, 60/2, pages 125-135, 1998.

[7] N. Amenta, M. Bern, and M. Kamvysselis. A new Voronoi-based surface reconstruction algorithm. In M. F. Cohen, editor, *SIGGRAPH 98 Conference proceedings, July 19–24, 1998*, pages 415-421, New York, 1998.

[8] N. Amenta, S. Choi, T. K. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. In *Proceedings of the 16th ACM Symposium on Computational Geometry*, pages 213-222, 2000.

[9] S. Arya and H. A. Fu. Expected-case Complexity for Approximate Nearest Neighbor Searching. In *Symposium on Discrete Algorithms*, pages 379-388, 2000.

[10] R. Baule. *Moving Least Squares Approximation mit parameterabhaengigen Gewichtsfunktionen*. Diploma thesis, Institut for numerical and applied mathematics, University of Goettingen, 2000.

[11] J. L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. In *Communications of the ACM*, pages 509-517, September 1985.

[12] P. J. Besl and N. D. McKay. A Pyramid Approach to Subpixel Registration Based on Intensity. In *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14(2), pages 239-256, Feb. 1992.

[13] R. N. Bracewell. *The Fourier Transform and Its Applications, 2nd revised edition*. McGraw-Hill, 1986.

[14] D. Brodsky and B. A. Watson. Model simplification through refinement. In *Proceedings of Graphics Interface*, pages 221-228, 2000.

[15] F. J. Canny. A computational approach to edge detection. *IEEE Trans PAMI*, 8(6):679–698, 1986.

[16] J. Carr, R. Beatson, J. Cherrie, T. Mitchell, W. Fright, B. McCallum, and T. Evans. Reconstruction and representation of 3d objects with radial basis functions. In E. Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, Annual Conference Series, pages 67–76. ACM Press / ACM SIGGRAPH, 2001.

[17] M. Carlson, P. Mucha, B. van Horn III, and G. Turk. Melting and Flowing. In *ACM SIGGRAPH Symposium on Computer Animation* San Antonio, Texas, July 21-22, 2002.

[18] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2), June 1998.

[19] U. Clarenz, U. Diewald, and M. Rumpf. Anisotropic geometric diffusion in surface processing. In T. Ertl, B. Hamann, and A. Varshney, editors, *Proceedings Visualization 2000*, pages 397–405. IEEE Computer Society Technical Committee on Computer Graphics, 2000.

[20] B. Curless and M. Levoy. A Volumetric Method for Building Complex Models from Range Images. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 303-312. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[21] J. Davis, S. Marschner, M. Garr, and M. Levoy. Filling holes in complex surfaces using volumetric diffusion. In *First International Symposium on 3D Data Processing, Visualization, Transmission*, June, 2002.

[22] T. DeRose, M. Kass, and T. Truong. Subdivision surfaces in character animation. In M. F. Cohen, editor, *SIGGRAPH 98 Conference proceedings, July 19–24, 1998*, pages 85–94, New York, 1998.

[23] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In A. Rockwood, editor, *Siggraph 1999*, Annual Conference Series, pages 317–324, Los Angeles, 1999. ACM Siggraph, Addison Wesley Longman.

[24] T. K. Dey, J. Giesen, S. Goswami, and W. Zhao. Shape dimension and approximation from samples. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms* (SODA), pages 772-780, 2002.

[25] M. P. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, Upper Saddle River, New Jersey 07458, 1976.

[26] N. Dyn. Interpolation and approximation by radial and related functions.. In C. K. Chui, L. L. Schumaker, J. D. Ward, editors, *Approximation Theory VI*, volume 1, pages 211-234, Academic Press, New York, 1989.

[27] N. Dyn and S. Rippa. Data-dependent triangulations for scattered data interpolation and finite element approximation. In *Applied Numerical Mathematics*, 12, pages 89-105, 1993.

[28] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In R. Cook, editor, *SIGGRAPH 95 Conference Proceedings*, pages 173–182. ACM SIGGRAPH, Aug. 1995.

[29] M. Eck and H. Hoppe. Automatic reconstruction of B-Spline surfaces of arbitrary topological type. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 325–334. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[30] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. In *Proceedings of the 41st IEEE Symposium on the Foundations of Computer Science 2000*, pages 454-463, 2002.

[31] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. A Hybrid Particle Level Set Method for Improved Interface Capturing. *Journal of Computational Physics 183*, pages 83-116, 2002.

[32] G. Farin. *Curves and Surface for CAGD*, 5th edition, Morgan-Kaufmann, 2001.

[33] S. Fleishman, D. Cohen-Or, M. Alexa, and C. T. Silva. Progressive Point Set Surfaces, *ACM Transactions on Graphics*, to appear.

[34] M. S. Floater and M. Reimers. Meshless parameterization and surface reconstruction In *Computer Aided Geometric Design*, 18, pages 77-92, 2001.

[35] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics, Principles and Practice*. Addison-Wesley, Reading, Massachusetts, second edition, 1990.

[36] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An Algorithm for Finding Best Matches in Logarithmic Expected Time. In *ACM Transactions on Mathematical Software*, pages 209-226, September 1977.

[37] S. Funke and E. A. Ramos. Smooth-surface reconstruction in near-linear time. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms* (SODA), pages 781-790, 2002.

[38] M. Garland. *Quadric-Based Polygonal Surface Simplification*. Ph.D. thesis, Computer Science Department, Carnegie Mellon University, CMU-CS-99-105, May 1999.

[39] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 209–216. ACM SIGGRAPH, Addison Wesley, Aug. 1997.

[40] J. Giesen and M. John. Surface Reconstruction Based on a Dynamical System. In *Proceedings of the 23rd Annual Conference of the European Association of Computer Graphics (Eurographics)*, Computer Graphics Forum 21, pages 363-371, 2002.

[41] J. Giesen and U. Wagner. Shape dimension and intrinsic metric from samples of manifolds with high co-dimension. In *Proceedings of the 19th annual symposium on Computational Geometry*, 2003.

[42] D. G. Goldman. Fake fur rendering. In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 127-134. ACM SIGGRAPH, Addison Wesley, Aug. 1997.

[43] M. H. Gross and A. Hubeli. Eigenmeshes. *Technical report*, ETH Zurich, Mar. 2000.

[44] J. P. Grossman and W. J. Dally. Point sample rendering. In G. Drettakis and N. Max, editors, *Rendering Techniques '98*, Eurographics, pages 181–192. Springer-Verlag Wien New York, 1998. Held in Vienna, Austria, July.

[45] A. Gruss, S. Tada, and T. Kanade. A VLSI Smart Sensor for Fast Range Imaging. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems*, 1992.

[46] S. Gumhold, X. Wang, and R. McLeod. Feature Extraction from Point Clouds. In *Proceedings of the 10th International Meshing Roundtable*, Newport Beach, 2001.

[47] I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. In A. Rockwood, editor, *Siggraph 1999, Computer Graphics Proceedings*, Annual Conference Series, pages 325–334, Los Angeles, 1999.

[48] I. Guskov, K. Vidimce, W. Sweldens, and P. Schröder. Normal meshes. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 95–102. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[49] P. M. Hall, A. D. Marshall, and R. R. Martin. Incremental Eigenanalysis for Classification. In *Proceedings of the British Machine Vision Conference 1998*, Volume 1, 1998.

[50] P. S. Heckbert. Fast Surface Particle Repulsion. *CMU Computer Science Tech. Report* CMU-CS-97-130, 1997.

[51] C. M. Hoffmann. *Geometric and solid modeling: An Introduction*. Morgan Kaufmann, 1989.

[52] H. Hoppe. Progressive meshes. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.

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

[54] B. K. P. Horn and M. J. Brooks. *Shape from Shading,* MIT Press, Cambridge, Massachusetts, 1989.

[55] A. Hubeli. *Multiresolution Techniques for Non-Manifolds*. PhD thesis, Department of Computer Science, ETH Zurich, 2002.

[56] A. Hubeli and M. Gross. Fairing of non-manifolds for visualization. In *Proceedings Visualization 2000*, pages 407–414. IEEE Computer Society Technical Committee on Computer Graphics, 2000.

[57] A. Hubeli and M. Gross. Multiresolution feature extraction from unstructured meshes. In *Proceedings Visualization 2001*, pages 287–294. IEEE Computer Society Technical Committee on Computer Graphics, 2001.

[58] I. Jolliffe. *Principal Component Analysis*, Springer Verlag, 1986.

[59] A. Kalaiah and A. Varshney. Differential Point Rendering. In S. J. Gortler, K. Myszkowski, editors, *Rendering Techniques '01*, Springer Verlag, pages 139-150, 2001.

[60] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. In *First International Conference on Computer Vision, (London, England, June 8–11, 1987)*, pages 259–268, Washington, DC., 1987. IEEE Computer Society Press.

[61] R. Keiser. *Collision Detection and Response for Interactive Editing of Point-Sampled Models*. Diploma thesis, Department of Computer Science, ETH Zurich, 2003.

[62] A. Khodakovsky, P. Schröder, and W. Sweldens. Progressive geometry compression. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 271–278. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[63] O. Knoll. *Interaktive Bearbeitung von Point-Sample basierten Objekten*. Diploma thesis (in German), Computer Science Department, ETH Zurich, 2001.

[64] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive multi-resolution modeling on arbitrary meshes. In M. Cohen, editor, *Proceedings of SIGGRAPH 98*, Annual Conference Series, Addison Wesley, pages 105–114. Addison Wesley, 1998.

[65] L. Kobbelt, M. Botsch, U. Schwanecke, and H. Seidel. Feature Sensitive Surface Extraction from Volume Data. In E. Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, Annual Conference Series, pages 57–66. ACM Press / ACM SIGGRAPH, 2001.

[66] J. J. Koenderink. The structure of images. *Biological Cybernetics*, Volume 50, pages 363-370, 1984.

[67] V. Krishnamurthy and M. Levoy. Fitting Smooth Surface to Dense Polygon Meshes. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 313-324. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[68] S. Krishnan and D. Manocha. An efficient surface intersection algorithm based on lower-dimensional formulation. In *ACM Transactions on Graphics 16(1)*, 1997.

[69] P. Lancaster and K. Salkauskas. Surfaces generated by moving least squares methods. *Math. Comp.,* 37, pages 141-159, 1981.

[70] A. Lee, D. Dobkin, W. Sweldens, and P. Schröder. Multiresolution mesh morphing. In A. Rockwood, editor, *Siggraph 1999, Computer Graphics Proceedings*, Annual Conference Series, pages 343–350, Los Angeles, 1999. ACM Siggraph, Addison Wesley Longman.

[71] I. Lee. Curve Reconstruction from Unorganized Points. In *Computer Aided Geometric Design*, Vol 17, pages 161-177, 2000.

[72] Y. Lee and S. Lee. Geometric Snakes for Triangle Meshes. In *Proceedings of the 23rd Annual Conference of the European Association of Computer Graphics (Eurographics)*, Computer Graphics Forum 21, pages 229-238, 2002

[73] D. Levin. The approximation power of moving least-squares. In *Math. Comp. Vol 67, No. 224*, pages 1517-1531, 1998.

[74] D. Levin. Mesh-independent surface interpolation, *Advances in Computational Mathematics*, submitted, 2001.

[75] M. Levoy, K. Pulli, B. Curless, S. Rusinnkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The Digital Michelangelo Project: 3D Scanning of Large Statues. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 131-144. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[76] M. Levoy and T. Whitted. The use of points as display primitives. *Technical Report TR-85-022,* The University of North Carolina at Chappel Hill, Department of Computer Science, 1985.

[77] B. Levy. Constrained texture mapping for polygonal meshes. In E. Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, Annual Conference Series, pages 417–425. ACM Press / ACM SIGGRAPH, 2001.

[78] T. Lindeberg. *Scale-Space Theory in Computer Vision*. Kluwer Academic Publishers, ISBN 0-7923-9418-6, 1994

[79] T. Lindeberg. Feature Detection with Automatic Scale Selection. In *International Journal of Computer Vision, Volume 30, no. 2*, 1998.

[80] L. Linsen. Point Cloud Representation. *Technical Report*, Faculty of Computer Science, University of Karlsruhe, 2001.

[81] L. Linsen and H. Prautzsch. Global Versus Local Triangulations. In *Eurographics 2001 Proceedings, Short Presentations*, 2001.

[82] L. Linsen and H. Prautzsch,. Fan Clouds - An Alternative To Meshes. In T. Asano, R. Klette, Ch. Ronse, editors, *Proceedings of Dagstuhl Seminar 02151 on Theoretical Foundations of Computer Vision - Geometry, Morphology, and Computational Imaging*, IEEE Computer Society Press, 2002.

[83] W. E. Lorensen and H. E. Cline. Marching Cubes: a high resolution 3D surface reconstruction algorithm. In *Computer Graphics, Vol. 21, No. 4*, (Proceedings of ACM SIGGRAPH), pages 163-169, 1987.

[84] K. Museth, D. E. Breen, R. T. Whitaker, and A. H. Barr. Level set surface editing operators. In J. F. Hughes, editor, *SIGGRAPH 2002, Computer Graphics Proceedings*, Annual Conference Series. ACM Press / ACM SIGGRAPH, 2002.

[85] J. Nievergelt and K. H. Hinrichs. *Algorithms & Data Structures*. vdf Hochschulverlag AG, ETH Zurich, 1999.

[86] Y. Parish and P. Mueller. Procedural Modeling of Cities. In E. Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, Annual Conference Series, pages 301–308. ACM Press / ACM SIGGRAPH, 2001.

[87] M. Pauly, R. Keiser, and M. Gross. Multi-scale Feature Extraction on Point-Sampled Models. In *Proceedings of the 24rd Annual Conference of the European Association of Computer Graphics (Eurographics)*, Computer Graphics Forum, to appear, 2003.

[88] M. Pauly, R. Keiser, L. Kobbelt, and M. Gross. Shape Modeling with Point-Sampled Geometry. In *SIGGRAPH 2003, Computer Graphics Proceedings*, Annual Conference Series. ACM Press / ACM SIGGRAPH, to appear, 2003.

[89] M. Pauly, L. Kobbelt, and M. Gross. Multiresolution Modeling of Point-Sampled Geometry. *Technical report*, ETH Zurich, 2002.

[90] M. Pauly, M. Gross, and L. Kobbelt. Efficient Simplification of Point-Sampled Surfaces. In *Proceedings Visualization 2002*, pages 163–170. IEEE Computer Society Technical Committee on Computer Graphics, 2002.

[91] M. Pauly and M. H. Gross. Spectral processing of point-sampled geometry. In E. Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, Annual Conference Series, pages 379–386. ACM Press / ACM SIGGRAPH, 2001.

[92] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 335–342. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[93] E. Praun, H. Hoppe, and A. Finkelstein. Robust Mesh Watermarking. In A. Rockwood, editor, *Siggraph 1999, Annual Conference Series*, pages 49-56, Los Angeles, 1999. ACM Siggraph, Addison Wesley Longman.

[94] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer Verlag, New York, 1985.

[95] P. Prusinkiewicz, L. Mündermann, R. Karwowski, and B. Lane. The use of positional information in the modeling of plants. n E. Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, Annual Conference Series, pages 289–300. ACM Press / ACM SIGGRAPH, 2001.

[96] W. Reeves. Particle systems - a technique for modeling a class of fuzzy objects. In *Computer Graphics*, 17(3), pages 359-376, July 1983.

[97] J. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering complex scenes. In *Geometric Modeling in Computer Graphics*, B. Falcidieno and T. Kunii, eds., pp. 455-465, Springer-Verlag, 1993.

[98] S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 343–352. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[99] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1990.

[100] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In *International Journal of Computer Vision*, 47(1), pages 7-42, May 2002.

[101] R. SedgeWick. *Algorithms in C++ (3rd edition)*. Addision Wesley, 1998.

[102] J. A. Sethian and S. Osher. Fronts Propagating with Curvature-Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations. *Journal of Computational Physics*, 79, pages 12-49, 1988.

[103] E. Shaffer and M. Garland. Efficient adaptive simplification of massive meshes. In *Proceedings Visualization 2001*, pages 127–134. IEEE Computer Society Technical Committee on Computer Graphics, 2001.

[104] J. R. Shewchuk. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. available at http://www-2.cs.cmu.edu/~jrs/jrspapers.html, August 1994.

[105] R. L. Sproull. Refinements of Nearest-Neighbour Searching in k-dimensional Trees. In *Algorithmica*, Vol. 6, pages 579-589, 1991.

[106] W. Sweldens and P. Schröder. Digital Geometry Processing. In *Course notes of Siggraph 2000,* ACM SIGGRAPH, 2001.

[107] R. Szeliski and D. Tonnesen. Surface modeling with oriented particle systems. In E. E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, July 1992.

[108] G. Taubin. A signal processing approach to fair surface design. In R. Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 351–358. ACM SIGGRAPH, Addison Wesley, Aug. 1995. held in Los Angeles, California, 06-11 August 1995.

[109] M. Teschner. *Direct Computation of Soft-Tissue Deformation in Craniofacial Surgery Simulation*. Ph.D. thesis, Shaker Verlag, Aachen, Germany, ISBN 3-8265-8317-5, January 2001.

[110] D. Tonnesen, *Dynamically Coupled Particle Systems for Geometric Modeling, Reconstruction, and Animation*. Ph.D. thesis, University of Toronto, Toronto, Canada, 1998.

[111] G. Turk. Re-Tiling Polygonal Surfaces. In E. E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 55–64, July 1992.

[112] G. Turk. Texture Synthesis on Surfaces. In E. Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, Annual Conference Series, pages 347–354. ACM Press / ACM SIGGRAPH, 2001.

[113] L. Velho, J. Gomes, and L. Figueiredo. *Implicit Object in Computer Graphics*. Springer Verlag, New York, 2002.

[114] A. P. Witkin and P. S. Heckbert. Using particles to sample and control implicit surfaces. In *SIGGRAPH 1994, Computer Graphics Proceedings*, Annual Conference Series, pages 269–278, 1994.

[115] D. Zorin, P. Schröder, T. DeRose, L. Kobbelt, A. Levin, and W. Sweldens. Subdivision for modeling and animation. In *Course notes of Siggraph 2000, ACM SIGGRAPH*, 2000.

[116] D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 259–268. ACM SIGGRAPH, Addison Wesley, Aug. 1997. ISBN 0-89791-896-7.

[117] M. Zwicker. *Continuous Reconstruction, Rendering, and Editing of Point-Sampled Surfaces*. Ph.D. thesis, Computer Graphics Laboratory, Computer Science Department, ETH Zurich, 2003.

[118] M. Zwicker, M. Pauly, O. Knoll, and M. Gross. Pointshop3d: An interactive system for point-based surface editing. In J. F. Hughes, editor, *SIGGRAPH 2002, Computer Graphics Proceedings*, Annual Conference Series. ACM Press / ACM SIGGRAPH, 2002.

[119] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Surface splatting. In E. Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, Annual Conference Series, pages 371–378. ACM Press / ACM SIGGRAPH, 2001.

B

GLOSSARY OF NOTATIONS

Point-Sampled Surfaces

P . . . . . A set of point samples

$\mathbf{p}$ . . . . . A point in $\mathbb{R}^3$

S . . . . . A manifold surface

N_p . . . . . The local neighborhood of a point $\mathbf{p}$

N_p^k . . . . . The set of k-nearest neighbors

N_p^B . . . . . The set of BSP neighbors

N_p^V . . . . . The set of Voronoi neighbors

H . . . . . The MLS reference plane

ρ_i . . . . . The discrete local sampling density at the i^{th} point of a point cloud

r_i . . . . . The radius of the local neighborhood of the i^{th} point of a point cloud

$\rho(\mathbf{x})$ . . . . . The continuous local sampling density at a point $\mathbf{x}$

$\eta(\mathbf{x})$ . . . . . The average local sample spacing

$\bar{\mathbf{p}}$ . . . . . The centroid of a local neighborhood

$\mathbf{C}$ . . . . . The covariance matrix of the sample positions of a local neighborhood

$\sigma_n(\mathbf{p})$ . . . . . The surface variation

$\mathbf{C}'$ . . . . . The covariance matrix of the normals of a local neighborhood

$\sigma_n'(\mathbf{p})$ . . . . . The normal variation

$M(S)$ . . . . . The medial axis of a surface S

$f(\mathbf{x})$ . . . . . The local feature size

$V_{P,S}$ . . . . . The restricted Voronoi diagram

$D_{P,S}$ . . . . . The restricted Delaunay triangulation
N_p^D . . . . . The set of restricted Delaunay neighbors
Π_m^d . . . . . The space of polynomials of degree m in $\mathbb{R}^d$
$\Phi(\mathbf{x}_i, \mathbf{x}_j)$ . . . The MLS kernel function for two points $\mathbf{x}_i$ and $\mathbf{x}_j$
$\phi(r)$ . . . . . The radially symmetric MLS kernel function for a distance r
h . . . . . The MLS scale parameter
Ψ_P . . . . . The MLS projection operator
$d(\mathbf{x})$ . . . . . The distance function
$d_+(\mathbf{x})$ . . . . . The signed distance function
$h_{\mathbf{x}}$ . . . . . The adaptive MLS scale parameter

Surface Simplification

C_i . . . . . A cluster of point samples
$n_{\max}$ . . . . . The maximum cluster size
$\sigma_{\max}$ . . . . . The maximum cluster variation
$Q_{\mathbf{p}}$ . . . . . The error quadric associated with a point $\mathbf{p}$
E_i . . . . . The tangent plane associated with an edge i.
$F(\mathbf{p})$ . . . . . The repulsion force exerted on particle $\mathbf{p}$
$d(\mathbf{q}, S)$ . . . . . The distance of a point $\mathbf{q}$ from a surface S
$\Delta_{\max}(S, S')$. The maximum distance between two surfaces S and S'
$\Delta_{\text{avg}}(S, S')$ . . The average distance between two surfaces S and S'

Multi-Scale Representation

L . . . . . A scale-space approximation of a function f
$g(\mathbf{x}, t)$ . . . . . A Gaussian filter kernel
t . . . . . The scale parameter
ΔL . . . . . The Laplacian operator of a function L
λ . . . . . The diffusion constant
$f_{i,j}$ . . . . . A discrete 2D signal
$\Delta\mathbf{x}$ . . . . . The Laplace-Beltrami operator at a point $\mathbf{x}$
$\Delta\mathbf{v}$ . . . . . The approximation of the Laplace-Beltrami operator at a vertex $\mathbf{v}$
ω_i . . . . . The weights in the discrete approximation of the Laplacian
$\mathbf{P}$ . . . . . A discrete multi-scale representation
P^l . . . . . The point cloud of level l of a discrete multi-scale representation

$\mathbf{D}$ . . . . . The sequence of sets of detail coefficients

D^l . . . . . The set of detail coefficients of level l of multi-scale representation

Feature Extraction

ω_i . . . . . The feature weights

$p(\lambda)$ . . . . . The characteristic polynomial of the covariance matrix

$\Omega(\mathbf{p}_i, n)$ . . . . The thresholding function

n_b . . . . . The size of the neighborhood for boundary detection

α_b . . . . . The maximum angle for boundary detection

Q . . . . . The set of feature nodes

$\omega_{\min}$ . . . . . The minimum variation bound

$\omega_{\max}$ . . . . . The maximum variation bound

$c(\mathbf{q}, \mathbf{q}_i)$ . . . . The edge cost between two points of the MST

$\mathbf{v}(s)$ . . . . . A parametric feature curve

$\mathbf{E}$ . . . . . The energy functional for active contour models

$\mathbf{E}_{int}$ . . . . . The internal energy

$\mathbf{E}_{ext}$ . . . . . The external energy

$\tilde{\sigma}(\mathbf{v}(s))$ . . . . The interpolated maximum variation

Shape Modeling

$\Omega_P(\mathbf{x})$ . . . . . The boolean classification predicate

χ_d . . . . . The deformable region

t . . . . . The deformation scale parameter

χ_1 . . . . . The one-region

χ_0 . . . . . The zero-region

$d_j(\mathbf{p})$ . . . . . The distance from the zero- resp. one-region

β . . . . . The deformation blending function

$F(\mathbf{p}, t)$ . . . . . The deformation function

$F_T(\mathbf{p}, t)$ . . . . The translatory deformation function

$F_R(\mathbf{p}, t)$ . . . . The rotational deformation function

s_p . . . . . The temporal coherence sphere centered at $\mathbf{p}$

$\Phi(\mathbf{p}_i, \mathbf{p}_j)$ . . . The inter-particle potential between $\mathbf{p}_i$ and $\mathbf{p}_j$

E_i . . . . . The potential of particle $\mathbf{p}_i$

Appearance Modeling

I . . . . . A 2D discrete image

B . . . . . A brush image

Φ . . . . . A parameterization

Ψ . . . . . The re-sampling operator

Ω . . . . . An editing operator

$\mathbf{b}_{mn}$ . . . . . An entry in the brush grid

$\mathbf{c}_{mn}$ . . . . . The diffuse color

d_{mn} . . . . . The displacement coefficient

$\mathbf{s}_{mn}$ . . . . . The specular color

$k_{a,\,mn}$ . . . . . The ambient coefficient

$k_{d,\,mn}$ . . . . . The diffuse coefficient

$k_{s,\,mn}$ . . . . . The specular coefficient

s_{mn} . . . . . The shininess coefficient

C_{dist} . . . . . The cost function for parameter distortion

C_{fit} . . . . . The cost function for the fitting constraints

$\tilde{C}$ . . . . . The discrete objective function

$\phi_i(\mathbf{u})$ . . . . . The local fitting functions

C

SAMPLING REQUIREMENTS

This section analyses sampling requirements for a point cloud P sampled from a smooth, twice-differentiable manifold surface S. The goal is to define sampling criteria such that the methods for local surface analysis based on k-nearest neighbors (Section 2.1.3) are provably accurate. The approach taken here follows recent work in surface reconstruction [7, 8]. First some definitions and previous results are stated:

Definition C.1: The *medial axis* $M(S)$ of a surface S in $\mathbb{R}^3$ is defined as the set of all points $\mathbf{x} \in \mathbb{R}^3$ such that there exist two points $\mathbf{x}_1, \mathbf{x}_2 \in S$ with $\mathbf{x}_1 \neq \mathbf{x}_2$ such that $\|\mathbf{x} - \mathbf{x}_1\| = \|\mathbf{x} - \mathbf{x}_2\|$ and no other point $\mathbf{x}_3 \in S$ with $\|\mathbf{x} - \mathbf{x}_3\| < \|\mathbf{x} - \mathbf{x}_1\|$. In other words, the medial axis is the closure of the set of points which have more than one closest point on S.

Definition C.2: The *local feature size* $f(\mathbf{x})$ of a point $\mathbf{x} \in S$ is a function $f: S \rightarrow \mathbb{R}$ that measures the minimum distance of $\mathbf{x}$ to the medial axis $M(S)$, i.e., $f(\mathbf{x}) = \min_{\mathbf{y} \in M(S)} \{\|\mathbf{x} - \mathbf{y}\|\}$.

Definition C.3: A point cloud P is an ε*-sample* of a surface S if for every $\mathbf{x} \in S$ there exists a $\mathbf{p} \in P$ such that $\|\mathbf{x} - \mathbf{p}\| \leq \varepsilon \cdot f(\mathbf{x})$.

An interesting property of the local feature size f is Lipschitz continuity, i.e., $f(\mathbf{x}) \leq f(\mathbf{y}) + \|\mathbf{x} - \mathbf{y}\|$ [8]. The ε-sampling criterion guarantees that a surface S is sampled densely in areas of high curvature and where two separate sheets of the surface come close together (see Figure 2.4). For suitable ε, this allows topologically exact reconstruction of the surface S from an ε-sample P (see [7]). For example, [8] presents an algorithm that for $\varepsilon \leq 0.08$ computes a piecewise-linear 2-manifold T homeomorphic to S such that for each $\mathbf{y} \in T$ there exists an $\mathbf{x} \in S$ with

$$\|\mathbf{x} - \mathbf{y}\| \leq \frac{1.3\varepsilon f(\mathbf{x})}{1 - \varepsilon}.$$

The ε-sampling criterion alone is not sufficient for local surface analysis based on k-nearest neighbors, however. Consider the example shown in Figure C.1 (a), where a cylindrical surface is sampled densely along the circumference, but sparsely in the direction of its axis. For any given ε and k, one can always find an ε-sample $P_{\varepsilon,k}$ of the cylinder such that

- the distance d between neighboring rows is less than $\varepsilon \cdot r$, where r is the cylinder radius which is equal to the local feature size, and
- the k-nearest neighbors all lie on a single row, e.g., when the samples are uniformly sampled along the circumference with a local sample spacing of d/k.

With such a sample, the covariance analysis of Section 2.1.3 will fail. For example, the normal estimation at a point $\mathbf{p} \in P_{\varepsilon,k}$ based on the k-nearest neighbors at $\mathbf{p}$, will yield a normal vector that is perpendicular to the true surface normal. As Figure C.1 (b) illustrates, such point clouds are quite common in practice, since the resolution of 3D scanners often varies depending on the spatial direction.

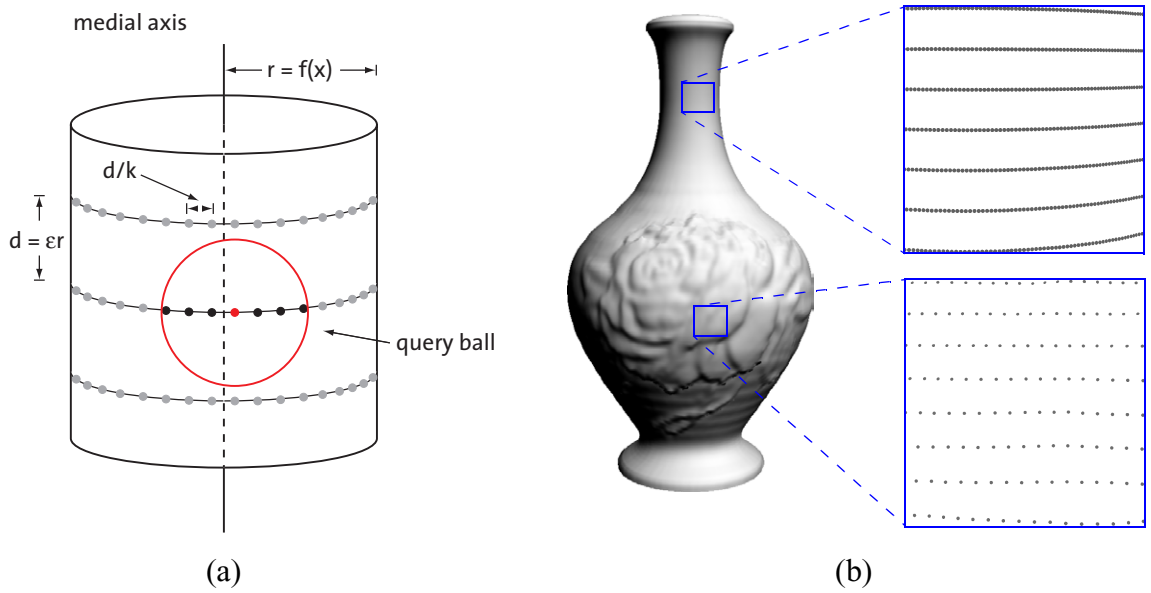


Figure C.1 (a) ε-sample of a cylinder, where k-nearest neighbors do not adequately represent the surface patch intersected by the query ball, (b) rotational laser scan of a vase that exhibits a similar sampling pattern.

This example demonstrates that additional sampling criteria are required for the local surface analysis based on k-nearest neighbors to be provably accurate. In [24, 37] the notion of an (ε, δ)-sample has been introduced:

Definition C.4: A sample P of a surface S is called (ε, δ)-*sample* of S for $\varepsilon/2 \leq \delta < \varepsilon < 1$, if P is an ε-sample, and $\|\mathbf{p}_i - \mathbf{p}_j\| \geq \delta f(\mathbf{p}_i)$ for all $\mathbf{p}_i, \mathbf{p}_j \in P$.

The ε-criterion gives a lower bound on the sampling density, while the δ-criterion provides a corresponding upper bound and controls the positions of the samples to ensure a certain uniformity of the sampling distribution. Note that from a theoretical point of view, the δ-criterion is not a severe limitation, since Funke and Ramos

presented an algorithm that computes an (ε, δ)-sample from a given ε-sample using point removal [37]. Points that are not essential for the description of the surface S are discarded, i.e., the redundancy of an ε-sample is diminished.

Given an (ε, δ)-sample P of a surface S the goal is to find a k such that the k-nearest neighbors of a sample point $\mathbf{p} \in P$ are an adequate sample of the surface patch $S_{p,k}$. A more formal definition of "adequate" makes use of the concept of Voronoi diagram and its dual, the Delaunay triangulation:

Definition C.5: Let V_P be the Voronoi diagram and D_P the Delaunay triangulation of P [94]. The *restricted Voronoi diagram* $V_{P,S}$ is the restriction of V_P to S, i.e., consists of the *restricted Voronoi cells* $V_{i,S} = V_i \cap S$, where V_i is the Voronoi cell of a point $\mathbf{p}_i \in P$. The dual of the restricted Voronoi diagram is the *restricted Delaunay triangulation* $D_{P,S}$.

Definition C.6: An edge $\overline{\mathbf{p}_i\mathbf{p}_j}$, $\mathbf{p}_i, \mathbf{p}_j \in P$ is called a *restricted Delaunay edge*, if $\overline{\mathbf{p}_i\mathbf{p}_j} \in D_{P,S}$, i.e., if and only if $V_{i,S} \cap V_{j,S} \neq \emptyset$. A point $\mathbf{p}_j \in P$ is called a *restricted Delaunay neighbor* of a point $\mathbf{p}_i \in P$ if $\mathbf{p}_i$ and $\mathbf{p}_j$ are connected by a restricted Delaunay edge.

Note that for a sufficiently dense sample of a closed surface S, the collection of restricted Delaunay edges incident at a point $\mathbf{p}_i \in P$, defines a piecewise linear surface (triangle fan) that is homeomorphic to a disc. In [6] it has been shown that the set N_p^D of restricted Delaunay neighbors of a point $\mathbf{p} \in P$ allows stable estimation of local surface properties such as the tangent plane. Thus k should be chosen such that the restricted Delaunay neighbors are a subset of the k-nearest neighbors, i.e., $N_p^D \subseteq N_p^k$.

The following lemma states that a restricted Delaunay edge of an ε-sample cannot be longer than a constant times the distance to the medial axis [41]:

Lemma C.1: Let P be an ε-sample of a surface S with $\varepsilon < 1$. If $\overline{\mathbf{p}_i\mathbf{p}_j}$, $\mathbf{p}_i, \mathbf{p}_j \in P$ is a restricted Delaunay edge, then

$$\|\mathbf{p}_i - \mathbf{p}_j\| \leq \frac{2\varepsilon}{1-\varepsilon} f(\mathbf{p}_i).$$

Thus all Delaunay neighbors of a point $\mathbf{p}_i$ are located within a sphere of radius $f(\mathbf{p}_i)2\varepsilon/(1-\varepsilon)$ around $\mathbf{p}_i$. The maximum number of points that can be fitted in this sphere under the (ε, δ) sampling criterion gives a lower bound for k. To compute this bound the following two lemmas are required [6, 24, 41]:

Lemma C.2: Assume that $\|\mathbf{x} - \mathbf{y}\| \leq \theta \cdot f(\mathbf{x})$ for $\mathbf{x}, \mathbf{y} \in S$ and some $\theta > 0$. Then the angle α between the line segment $\overline{\mathbf{x}\mathbf{y}}$ and the tangent space $T_{\mathbf{x}}$ of S at $\mathbf{x}$ satisfies

$$\sin\alpha \leq \frac{\theta}{2}.$$

Lemma C.3: If $\mathbf{x}, \mathbf{y} \in S$ with $\|\mathbf{x} - \mathbf{y}\| \leq \varepsilon \cdot f(\mathbf{x})$, then $f(\mathbf{x}) \leq \frac{1}{1-\varepsilon} f(\mathbf{y})$.

Given these two lemmas, the desired lower bound for k can be estimated using a packing argument. The idea is to count the number of balls of radius $\delta f(\mathbf{x})$ that can be packed without intersection onto the surface patch that contains all the restricted Delaunay neighbors. The following lemma formalizes this idea:

Lemma C.4: Let P be an (ε, δ)-sample of a surface S with $\varepsilon < 0.15$, $\varepsilon/2 \le \delta < \varepsilon < 1$ and

$$k \ge \frac{16(\varepsilon - 1)^2}{(\varepsilon^3 - 3\varepsilon^2 + 7\varepsilon - 1)(\varepsilon^3 - 3\varepsilon^2 - \varepsilon - 1)}.$$

Then the k-nearest neighbors of a sample point $\mathbf{p} \in P$ include the set of restricted Delaunay neighbors of $\mathbf{p}$.

Proof: Let $S' = S \cap s_p$ be the intersection of the surface S with the ball s_p centered at $\mathbf{p} \in P$ with radius $R = f(\mathbf{p})2\varepsilon/(1-\varepsilon)$. The goal is to determine the number M of δ-balls of radius $r = \delta f_{min}$ with centers on S than can be packed onto S', where $f_{min} = \min\{f(\mathbf{p}_i) \,|\, \mathbf{p}_i \in s_p\}$. This problem can be simplified by considering the intersection of these δ-balls with the tangent space T_p at $\mathbf{p}$ (see Figure C.2). From Lemma C.2 it follows that the maximum distance d of a point $\mathbf{x} \in S'$ from T_p is bounded by $d \le f(\mathbf{p})R^2/2$. Thus the smallest radius r' of the intersection of any of the δ-balls with T_p is bounded by $r'^2 \ge r^2 - f(\mathbf{p})R^2/2$, where $r \ge \delta f(\mathbf{p})(1-\varepsilon)$ (see Lemma C.3). A conservative estimate on the number of circles of radius r' that can be packed into a circle of radius R is $M \le M' = R^2/r'^2$. Combining these results yields the following upper bound on M:

$$M \le \frac{16(\varepsilon - 1)^2}{(\varepsilon^3 - 3\varepsilon^2 + 7\varepsilon - 1)(\varepsilon^3 - 3\varepsilon^2 - \varepsilon - 1)}.$$

□

For example, given an (ε, δ)-sample with $\varepsilon < 0.1$ it is guaranteed that for $k > 35$ all restricted Delaunay neighbors are among the k-nearest neighbors. For $\varepsilon \to 0$ and $\delta = \varepsilon/2$, the lower bound on k converges to 16. A similar argument yields an upper bound for the size of the query ball of the k-nearest neighbors with respect to the feature size. Here one tries to pack as many ε-balls, i.e., balls with radius $\varepsilon f(\mathbf{x})$ into the surface patch S'.

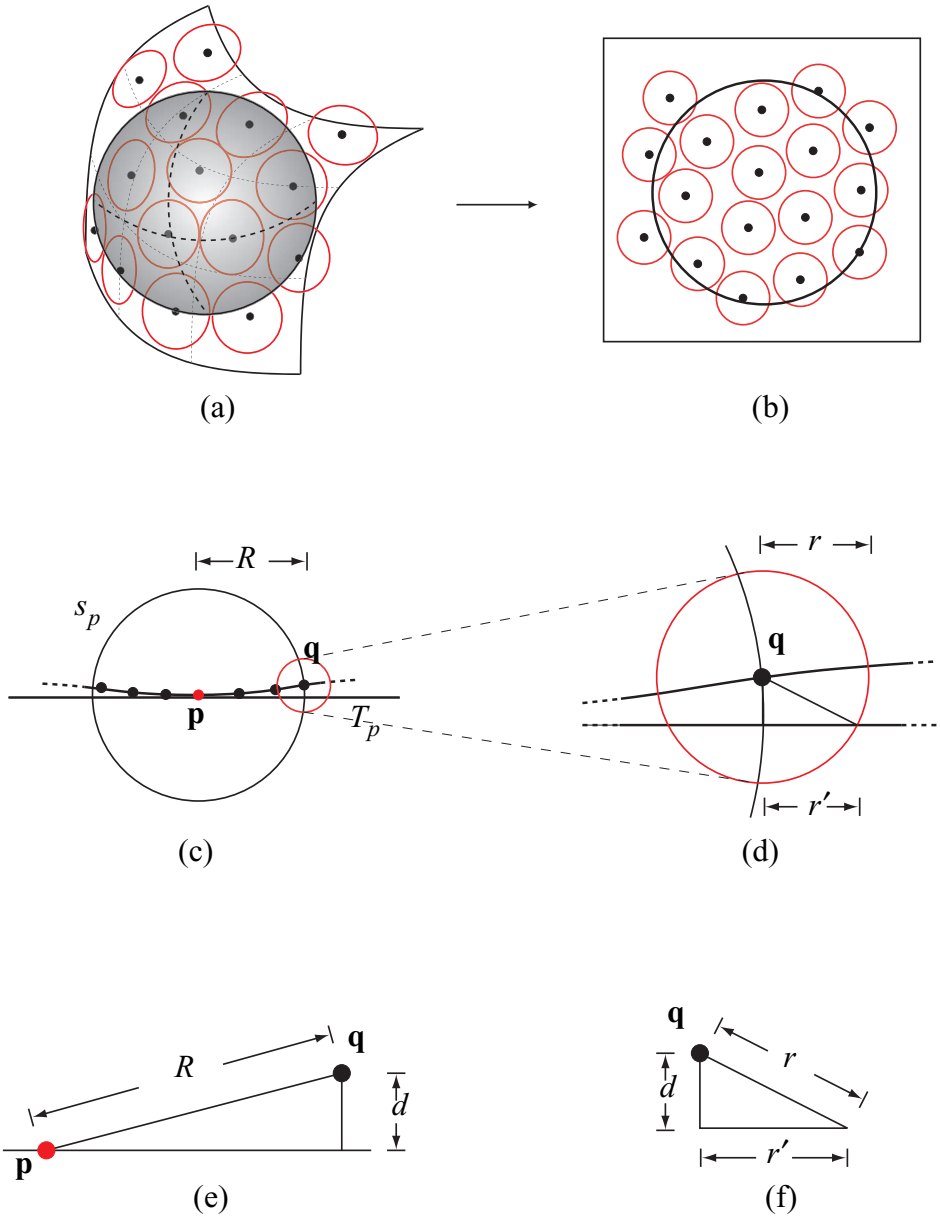


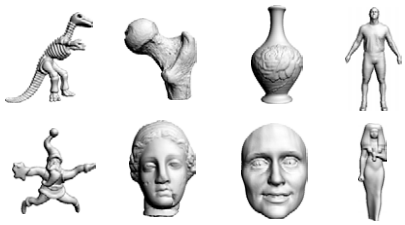
Figure C.2 Illustration of the proof of Lemma C.4. (a) Packing δ-balls into the intersection of the surface with a sphere of radius R, (b) reduction to sphere packing in the tangent space, (c-f) illustration of the relations of the different variables used in the proof.

D

DATA SOURCES

The following list specifies the sources of origin of all models used in this thesis that have not been scanned or created by the author.

- The dinosaur, ball joint, vase, male, santa, Igea, face, and Isis models are courtesy of Cyberware Inc., USA.



- The bunny and dragon models are courtesy of the Stanford 3D Scanning Repository, Stanford University, USA.

- The David and St. Matthew models are courtesy of Marc Levoy, Digital Michelangelo Project, Stanford University, USA.

- The cat model is courtesy of Hugues Hoppe, Microsoft Research, USA.

- The Max Planck model is courtesy of MPI Saarbrücken, Germany

E

CURRICULUM VITAE

Personal Data

Mark Pauly
Address:
Swiss Federal Institute of Technology (ETH)
Computer Graphics Laboratory
ETH Zentrum, IFW C25.2
8092 Zurich, Switzerland
++41-1-6320906
pauly@inf.ethz.ch
http://graphics.ethz.ch/~pauly/

18. Feb. 1974	born in Bernkastel - Kues, Germany

Education

Oct. 1992 - Dec. 1999	study of computer science at the University of Kaiserslautern, Germany
Oct. 1994 - Jul. 1995	study of computer science at the University of Edinburgh, Scotland
Apr. 1997 - Jul. 1997	research visit at the Imager Computer Graphics Laboratory at the University of British Columbia, Vancouver, Canada,
Oct. 1997 - Apr. 1998	research assitantship at the University of New South Wales, Sydney, Australia

Dec. 1999	computer science diploma (with honours)
since Apr. 2000	Ph.D. student at the Swiss Federal Institute of Technology, Zürich, Switzerland
Oct. 2002 - Nov. 2002	research visit at the multires modeling group, Caltech, Pasadena, USA

Scholarships

Mar. 1993 - Dec. 1999	full-time scholarship of the German National Merit Foundation
Apr. 2000 - Mar. 2003	Ph.D. scholarship of the European Graduate Program on Combinatorics, Geometry & Computation

Publications

EUROGRAPHICS 2003	Mark Pauly, Richard Keiser, Markus Gross: *Multi-scale Feature Extraction on Point-sampled Surfaces*
SIGGRAPH 2003	Mark Pauly, Richard Keiser, Leif Kobbelt, Markus Gross: *Shape Modeling with Point-sampled Geometry*
IEEE Visualization 2002	Mark Pauly, Markus Gross, Leif Kobbelt: *Efficient Simplification of Point-sampled Surfaces*
SIGGRAPH 2002	Matthias Zwicker, Mark Pauly, Oliver Knoll, Markus Gross: *Pointshop3D: An Interactive System for Point-based Surface Editing*
ETH Technical Report 2002	Mark Pauly, Leif Kobbelt, Markus Gross: *Multiresolution Modeling of Point-Sampled Geometry*
SIGGRAPH 2001	Mark Pauly, Markus Gross: *Spectral Processing of Point-sampled Geometry*
Eurographics Workshop on Rendering 2000	Mark Pauly, Thomas Kollig, Alexander Keller: *Metropolis Light Transport for Participating Media*
University of Kaiserlautern, diploma thesis 1999	Mark Pauly: *Robust Monte Carlo Methods for Photorealistic Rendering of Volumetric Effects*