Capturing the Essence of Shape of Polygonal Meshes

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg
von

Dipl.-Inf. Tobias Isenberg

geboren am 14. Dezember 1974 in Gardelegen

Gutachter: Prof. Dr. Thomas Strothotte
Prof. Dr. Kees van Overveld
Prof. Dr. Hans Hagen

Ort und Datum des Promotionskolloquiums:
Magdeburg, 16. April 2004

© 2004 by Tobias Isenberg. All rights reserved.

Für Johanna Jellen.

In Erinnerung an
Käthe und Wilhelm Isenberg
und Herbert Jellen.

Abstract/Zusammenfassung

Geometric models are the basis of computer graphics. Due to the growing computing power and hardware support more and increasingly complex models are created. In order to efficiently store, evaluate, manipulate, and match these, it is necessary to capture and extract the essence of shape. In particular for polygonal models, a concept is developed that allows adaptation of the notion of importance to the specific application without changing the extraction algorithm itself. For the use with this concept, a variety of criteria are described and conceived that capture what commonly is considered to be important in a polygonal mesh. In addition, an algorithm is developed within the concept that allows extraction of external skeletons as one aspect of essence of shape. Moreover, it is demonstrated that the concept also covers different notions of shape such as silhouettes. Since these have different application-specific requirements, a different algorithm is presented for fulfilling them. Finally, the methods that were presented are discussed with respect to potential application areas and a number of examples are shown.

Geometrische Modelle bilden die Basis der Computergraphik. Aufgrund ständig wachsender Rechenkapazität und steigender Hardwareunterstützung werden immer mehr und zunehmend komplexere Modelle erzeugt. Um diese effizient speichern, evaluieren, manipulieren und miteinander vergleichen zu können ist es notwendig, das Wesentliche der Form von Objekten zu finden und zu extrahieren. Insbesondere für polygonale Modelle wird ein Konzept entwickelt, das es erlaubt, die Notation von Wichtigkeit an die spezifische Anwendung anzupassen, ohne den Extraktionsalgorithmus selbst zu verändern. Es werden eine Reihe von Kriterien für die Verwendung innerhalb dieses Konzeptes beschrieben und neu entworfen. Diese werden verwendet, um Strukturen entsprechend der üblichen Auffassung von wichtigen Merkmalen in polygonalen Modellen zu finden. Des weiteren wird entsprechend des Konzeptes ein Algorithmus entwickelt, der externe Skelette als einen Aspekt von wesentlichen Formmerkmalen extrahiert. Außerdem wird gezeigt, daß das Konzept auch andere Formmerkmale wie beispielsweise Silhouetten beinhaltet. Da diese andere anwendungsbezogene Anforderungen haben, wird ein entsprechender Algorithmus präsentiert, der diese erfüllt. Abschließend werden die vorgestellten Methoden in Bezug auf ihre möglichen Anwendungsgebiete untersucht und eine Anzahl von Beispielanwendungen vorgestellt.

Preface

This dissertation is the result of my research work at the Department of Simulation and Graphics at the Faculty of Computer Science of the Otto-von-Guericke University of Magdeburg. I was advised by Prof. Strothotte to whom I am very grateful for giving me this opportunity and from whom I learned a lot.

I also thank my external reviewers Prof. Hans Hagen and Prof. Kees van Overveld for unhesitantly taking on the review as well as for their constructive criticism and helpful comments.

I would like to thank my parents and my grandmother for the support and encouragement they have always provided me. Without them this thesis would not have been possible. I thank my girlfriend Petra first of all for being a great friend and support, but also for the joint work on funny elephants, for her contribution to OpenNPAR, and for the beautiful model of the pitcher of a *Nepenthes alata* plant.

I am very grateful to Stefan Schlechtweg for many discussions throughout the whole time I was working on the dissertation and for giving a lot of valuable advice during the writing process. In addition, I would like to thank Henry König for sharing the office with me and always offering his help and his tremendous knowledge of C++ programming and debugging. Nick Halper, Roland Jesse, Henry Sonnet, Felix Ritter, and Marcel Götze deserve my thanks for our joint work on non-photorealistic rendering and the OpenNPAR system. In this context I must not forget to name my students Angela Brennecke, Johannes Zander, and Bernd Nettelbeck and thank them for their contributions to OpenNPAR.

Stefan Schirra helped me by answering many questions on theoretical concepts, computational complexities, and notation as well as Michiel Smid with notation problems and with proofreading the dissertation. I also thank Klaus Sachs-Hombach for the interesting discussion on the notion of the term ‘shape’. For taking the time to proofread the whole or parts of the dissertation I am very thankful to Lissa Janoski, Andreas Raab, Wallace Chigona, and Niklas Röber.

Last but certainly not least I am very grateful for the great environment provided by the Department of Simulation and Graphics. In particular, thanks for the many questions answered and the many problems solved by Petra Janka, Petra Specht, Beate Traoré, and Sylvia Zabel in the department’s office.

The work and parts of this thesis including the framework OpenNPAR were developed in collaboration with the following people: Angela Brennecke, Sheelagh Carpendale, Marcel Götze, Nick Halper, Roland Jesse, Bernd Nettelbeck, Petra Neumann, Felix Ritter, Henry Sonnet, Thomas Strothotte, and Johannes Zander (for details see Table A.1). Parts of this thesis have have been published or submitted for publication as [IHS02], [HIR$^+$03], [IFH$^+$03], [IHK03], [JI03], [SIDS03], [BI04], [GNIS04], [JINS04], and [ZISS04].

Table of Contents

Abstract i

Preface iii

1 Introduction 1
- 1.1 Motivation and Goals 2
- 1.2 Results 3
- 1.3 Structure of the Dissertation 3

2 Essence of Shape 7
- 2.1 Shape 7
- 2.2 Motivation and Definitions 8
- 2.3 Requirements of an Essential Shape Representation 10
- 2.4 Essential Shape Representations for Polygonal Models 12
- 2.5 ESR Criteria 13
  - 2.5.1 Geometric vs. Non-Geometric Criteria 14
  - 2.5.2 Geometric Criteria for Abstraction of Form 15
  - 2.5.3 Geometric Criteria for Surface Details 15
- 2.6 Summary 16

3 Approaches to Capturing Geometric Features of Shapes 17
- 3.1 Mathematical Concepts for Shape Abstraction 17
  - 3.1.1 Voronoi Diagrams and Medial Axes 18
  - 3.1.2 Reeb Graphs 24
- 3.2 Feature Line Extraction from Discrete Data Representations 27
  - 3.2.1 Skeletons from 2D Pixel Images 28
  - 3.2.2 Skeletons from Volumetric Data 29
- 3.3 Extracting Internal Features from Boundary Representations 33
  - 3.3.1 Medial Surface or Voronoi Diagram Based Algorithms 34
  - 3.3.2 Edge Collapse Based Algorithms 35
  - 3.3.3 Radial Basis Function Approach 39
  - 3.3.4 Skeletons From Point Clouds 40
- 3.4 External Feature Extraction from Boundary Representations 42
  - 3.4.1 Feature Lines Based on Approximated Curvature 42
  - 3.4.2 External Feature Lines From Internal Skeletons 45

3.4.3 Approach Requiring User Interaction 46
3.5 Summary 47

4 Degree of Interest Functions 49
4.1 Concept and Framework 49
4.1.1 Degree of Interest and Extreme Ridges 49
4.1.2 Discrete DoI Functions 52
4.1.3 Determining the Discrete Curvature of Polygonal Meshes 53
4.1.4 Simple DoIs Based on Principal Curvature 55
4.2 Heuristic DoI Function 56
4.3 Modified Shape Index 58
4.3.1 Shape Index 58
4.3.2 Modifications to Emphasize Ridges and Ruts 58
4.4 Fold Index 60
4.5 Singularity Index 63
4.6 DoI Functions for Adaptive Subdivision 65
4.6.1 DoI Functions for Silhouette Generation 66
4.6.2 DoI Functions for Improving Haptic Rendering 67
4.6.3 DoI Functions Based on User Interaction 68
4.7 Summary 69
4.7.1 Classification and Hierarchy of DoI Functions 69
4.7.2 Comparison of Geometric DoI Functions 70

5 An Essential Shape Representation in 3D: the External Skeleton 75
5.1 Analysis of Internal Skeletons 75
5.1.1 Suitability for Different Classes of Models 75
5.1.2 Correlation of the Skeleton Graph with the Original Model . . . 76
5.1.3 Sensitivity to the Tessellation of Shapes 77
5.1.4 Sensitivity to Errors in the Mesh 79
5.1.5 Skeletons of Segmented Meshes 80
5.1.6 Necessity for a New Concept 82
5.2 External Skeletonization using Wave Propagation 82
5.2.1 Definition and Algorithm Outline 82
5.2.2 Rationale of Wave Propagation 84
5.2.3 Finding Local Maxima 85
5.2.4 Wave Propagation on Polygonal Surfaces 86
5.2.5 Regularization 88
5.2.6 Following Local Maxima Curves On Wavefronts 90
5.2.7 External Skeleton Graph 92
5.3 Algorithmic Considerations 93
5.3.1 Efficient Computation 93
5.3.2 Winged Edge Data Structure 93
5.3.3 Modifications of the Algorithm 95
5.4 Analysis of the External Skeleton 96
5.4.1 Scale 96

5.4.2 Reconstruction 98
5.4.3 Robustness of the Method and Potential Problems 103
5.5 Summary 105

6 Essential Shape Representations in 2D: Silhouettes 107
6.1 Definition of Silhouette Concept 107
6.2 Silhouettes as ESRs in 2D 108
6.3 Approaches for Detecting the Silhouette of Polygonal Meshes 111
6.3.1 Image-Space Algorithms 112
6.3.2 Hybrid Algorithms 113
6.3.3 Object-Space Algorithms 116
6.3.4 Comparison of Silhouette Detection Algorithms 122
6.4 From Silhouette Edges to Visible Strokes 123
6.4.1 Visibility Culling of Silhouette Edges 123
6.4.2 A New Hybrid Approach for Determining Visibility 126
6.4.3 Finding Silhouette Strokes 129
6.4.4 Removing 2D Artifacts 129
6.5 Analysis of the Approach 133
6.5.1 Examples 134
6.5.2 Computation Times 134
6.5.3 Summary 136

7 Applications and Case Study 139
7.1 Progressive Coding of Shape Information 139
7.1.1 Progressive Meshes and Compression 140
7.1.2 Feature Driven Mesh Decimation 140
7.1.3 Linearizing the Mesh Data for Network Transfer 141
7.2 Shape Matching and Database Search 142
7.2.1 Similarity Measure Based on Shape Abstractions 142
7.2.2 Shape Matching with Edge Based Skeletons 143
7.2.3 Shape Matching with Silhouettes 145
7.2.4 Model Database Search Using Silhouettes or Skeletons 145
7.3 Non-Photorealistic Rendering 146
7.3.1 Silhouettes and Skeletons in Non-Photorealistic Techniques . . . 147
7.3.2 Stroke Manipulation Based Example Applications 150
7.3.3 Combination with Alternative Stroke Generation 152
7.4 Summary 154

8 Concluding Considerations 155
8.1 Summary 155
8.2 Criticism 158
8.3 Future Work 158

References 161

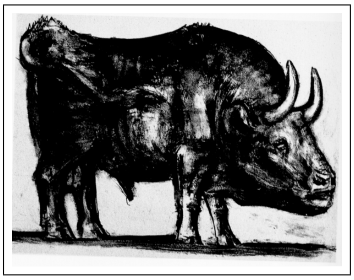
List of Figures 177

List of Tables 185

A Implementations 187

B A Unifying Implementation: OpenNPAR 189
B.1 Winged Edge Data Structure . . . 189
B.2 Stylization and Visualizing Strokes . . . 190

Résumé 193



Chapter 1

Introduction

In 1945/46, Picasso created the lithography series “The Bull” that is shown in Figure 1.1. It depicts the step by step abstraction from fairly detailed and almost realistic depiction of a bull to a highly stylized one with only a few curves. This seems to be a very good and intuitive example for capturing the essence of shape. As the level of abstraction increases, Picasso concentrates on the curves on the surface of the object that are characteristic for its shape. Those are curves that indicate, for example, places of a high local curvature.

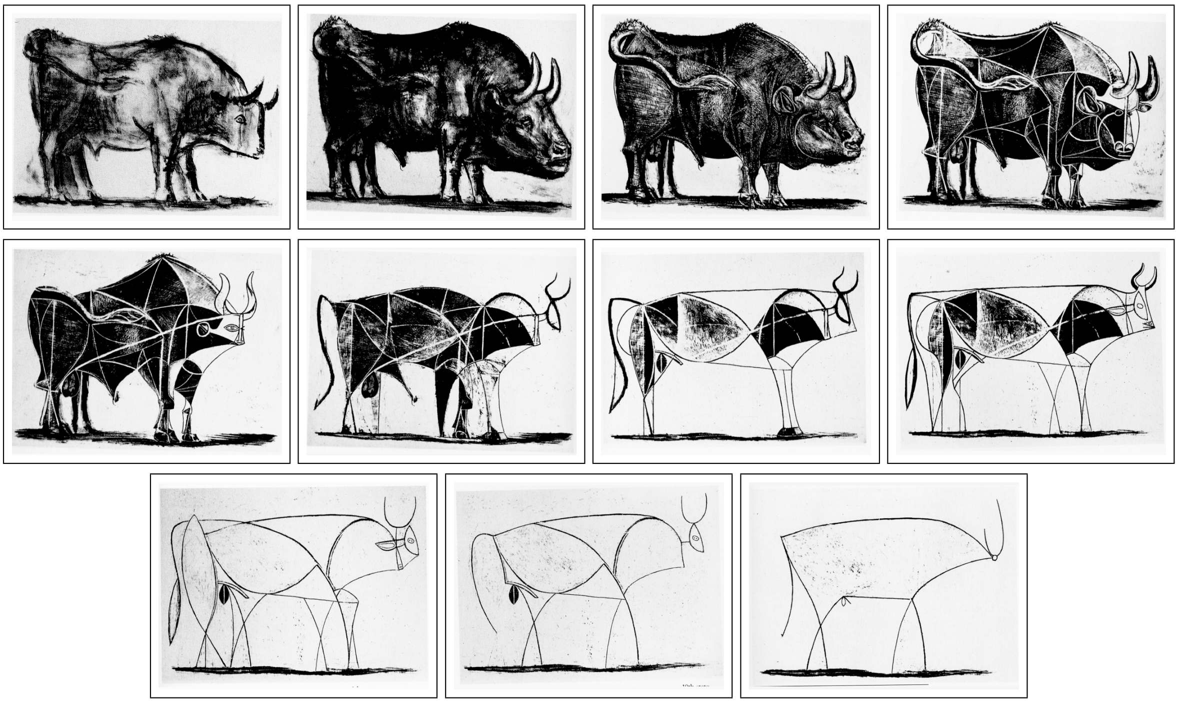


Figure 1.1: “The Bull.” Part of the lithography series by Picasso. From [Lév91].

However, Picasso not only draws the curves that are relevant in terms of the static aspect of shape but also uses silhouette lines that border the object from the background. This illustrates that silhouettes are important shape descriptors as well. In this context, the figure very nicely links feature detection or skeleton extraction, feature driven mesh decimation (as another type of abstraction), and silhouette computation.

This example illustrates that abstractions are used as a medium of art and as a way to detect the essence in a shape. In addition, these abstractions are also often used as a tool of art. For example, when an artist draws or paints a human figure he or she has an abstraction of a human in mind (see, e. g., [Jen01]). This is used to make sure that the depiction of the human has, e. g., the right proportions before the artist goes on to add individual features to the drawing or painting.

1.1 Motivation and Goals

These two artistic examples illustrate that the knowledge of essential features of shape is of great importance for the depiction of objects. However, it is not only important in art but also in all areas where shapes are created, processed, or depicted. Therefore, this especially applies to the area of computer graphics that deals with creating images of shapes.

In the computer graphics domain shapes are created, represented, processed, and depicted in a great variety of ways. In particular in three-dimensional computer graphics, shape representation as geometric models are the basis of the image generation process. Due to the growing demand for three-dimensional visualizations not only in "serious" applications such as CAD/CAM but also in computer games or computer-animated movies the hardware support for the processing of geometric models has strongly advanced in recent years [Rub03]. The shape representation supported by these *hardware accelerators* is almost exclusively polygonal meshes, specifically triangle meshes. This is one of the reasons for using polygonal meshes in many applications in contemporary computer graphics.

There are many ways to create or acquire models in form of meshes. For example, there is a variety of modeling tools such as 3D Studio MAX that support the incremental construction of models using a set of primitives and manipulations. Alternatively, three-dimensional meshes can also be acquired using 3D scanning hardware. Using a physical process (e. g., reflection of a laser or ultrasound) the surface of a physical object is sampled and a point cloud is generated. The point cloud, in turn, is triangulated in order to create a triangle mesh (for an overview of techniques see, e. g., [KSK98]).

The increasing demand for models and computing power as well as the number of ways for creating or acquiring them leads to ever growing collections of 3D meshes that can be used in applications (such as the VIEWPOINT Model Catalog [Vie99]). Therefore, there is a great demand for automatic ways to evaluate, recognize, compare, search for, simplify, and animate meshes. This means that methods are necessary that automatically compute abstractions of models or detect important features, i. e., to *capture the essence of shape.*

Many different algorithms have been introduced previously that capture certain aspects of shape (e. g., [DHR$^+$99, RKS00, WB01, HBK02]). All of them are very specific in what they consider to be important in a shape. However, the wide variety of applications also means that there is a variety of different concepts of what is considered to be important. Hence, a general concept that allows users to determine and exchange what is meant by “essential” is necessary. This would facilitate an easy adaptation of the method to new applications by only exchanging the criterion for importance.

1.2 Results

Starting from this problem statement the following results have been achieved:

- The concept of *essential shape representation (ESR)* is devised. The concept allows the distinction between algorithms that compute the essence of shape (*ESR scheme*) and criteria that determine what is considered to be important (*ESR criterion*).
- Specifically for polygonal meshes, a group of mostly geometric ESR criteria (*DoI functions*) that allows the capturing of common notions of importance is developed and evaluated.
- A specific ESR scheme based on wave propagation on meshes that extracts a curve based essential shape representation called *external skeleton* using a DoI function is conceived.
- The silhouette of objects was examined as an essential shape representation and due to the special requirements of potential applications (e. g., interactive or real-time frame rates, analytic representation, and visibility culling) an object-space algorithm that allows for an efficient silhouette computation is presented.
- Potential applications in different domains are discussed for the presented methods and a variety of examples are given.

1.3 Structure of the Dissertation

The remainder of this dissertation is structured as follows. First, the foundation for the following chapters is laid in Chapter 2. The chapter starts with a discussion of the concept of shape and the notion of shape used in the context of this dissertation. Afterwards, basic definitions regarding the goal of an *essential shape representation* are derived from a more detailed motivation. Based on these definitions and a number of desired applications, a set of requirements for essential shape representations are stated. Next, the notion of shape is specified in more detail with respect to polygonal meshes. The chapter concludes with a classification of the criteria that will later be used to denote importance in a shape.

The foundations laid out in this chapter will be referred to throughout the remainder of the dissertation. In particular, the distinction between an algorithm for computing the essence of shape (ESR scheme) and the criterion that allows to evaluate the importance of features (ESR criterion) is essential for the discussion in the following chapters. In addition, the requirements derived here will be used to evaluate not only previous work but also the newly developed algorithms.

Chapter 3 gives an overview of previously presented techniques for capturing features in shapes. The chapter focuses on methods for geometric features because these are by far the most common. The chapter starts with an overview of important mathematical concepts because these are later used in many other algorithms as well. This is followed by a discussion of feature extraction from discrete data representations since these are a second important group of shape representations in addition to polygonal meshes. The latter are examined afterwards. In this review of feature extraction from polygonal models it distinguished between internal and external structures that are detected.

By performing this extensive review of previous work it could be shown that most methods extract some type of graph representation from the data. However, it was found that what is considered to be essential in a shape varies significantly between algorithms and data types from which features are extracted. Even algorithms that operate on the same data type and look for the same category of features typically produce different results.

In order to systematize the notion of essence in a shape, Chapter 4 develops a concept for denoting importance. Specifically for determining the relevance of points on the surface of an object the notion of a degree of interest function (DoI function) is defined. Since many of the DoI functions described and conceived later are based on principal curvatures, a method for approximating these for polygonal meshes is reviewed. Using this mathematical background, a variety of known and newly developed DoI functions are examined and evaluated. The chapter finishes with a comparison of geometric DoI functions that were developed.

With the concept of a DoI function, a user is now able to create importance criteria that fulfill the requirements of specific applications. The criteria discussed in Chapter 4 can serve as a collection for users to select from. Many of these DoI functions emphasize geometric features such as ridges and ruts since most users are interested in geometric properties of surfaces. Alternatively, the discussed DoI functions can be used as starting points to derive new criteria if an application calls for specific adaptations.

Based on the concept of DoI functions, a new algorithm for computing an essential shape representation is introduced in Chapter 5. Referring back to the review of techniques in Chapter 3, it starts with a discussion of methods that extract internal structures from polygonal models and derives the necessity for a new concept for capturing the essence of shape. Afterwards, the wave propagation based algorithm for the extraction of external skeletons is described and explained. In addition, algorithmic considerations as well as an analysis of the method in terms of scale and reconstruction are given.

The wave propagation that is performed by the algorithms serves two major objectives. On the one hand, it is a means for traversing the mesh. On the other hand, it provides an easy way to identify the branching features on the surface. Because the new algorithm does not use a predefined notion of importance, any of the previously introduced DoI functions can be used as importance criteria. This makes it more flexible than previous algorithms.

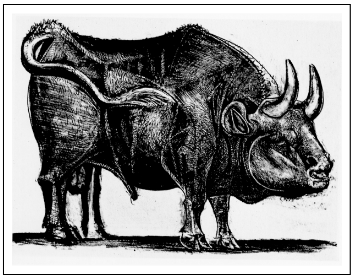
Chapter 6 turns to a different aspect of shape—the silhouette of objects. The chapter first shows that silhouettes as well are in fact essential shape representations as these were defined in Chapter 2. Thus, they can be computed with the wave propagation based algorithm using the appropriate DoI function. However, based on the requirements of typical applications that use silhouettes the need for a different silhouette algorithm is argued. By performing a comprehensive review of silhouette detection algorithms a group of methods that facilitates a flexible use of the curves is identified. Following this, an algorithm that performs the necessary visibility culling efficiently is presented. The combined approach is analyzed in terms of results and performance.

Only the combination of a fast silhouette detection algorithm with a similarly fast silhouette visibility culling method can serve as basis for an efficient silhouette rendering program. Otherwise it is not possible to achieve interactive or real-time frame-rates on the one side and be able to apply further manipulations such as stylization to the extracted strokes on the other side. The presented method can fulfill these requirements as it is demonstrated by examples and experimental results.

In order to evaluate the developed concept and algorithms according to their practical use, Chapter 7 reviews a number of potential applications. Three major domains are covered: progressive shape coding, shape matching and database search, and non-photorealistic rendering. This discussion is illustrated with a number of examples generated by a prototypical application. This tool, OpenNPAR, is also introduced and a number of its main characteristics are presented.

The great variety of possible application areas demonstrates that the concept of the essential shape representation is very powerful and flexible. In particular, the linear character of the extracted information (i. e., a graph or strokes) gives rise to many ways of using ESR algorithms as it is shown by a selection of examples.

Chapter 8 concludes the dissertation by reviewing the achieved results based on a number of propositions. In addition, some points of criticism are given and potential areas of future work are discussed.



Chapter 2

Essence of Shape

Before it is possible to go into detail about specific algorithms or discuss related work, it is necessary to clarify the vocabulary that is used. In addition, the rationale of the approach has to be defined and explained based on the motivation, potential applications, and the type of shapes that are used.

This chapter starts with a discussion of the notion of shape since this is a frequently used concept. In order to be able to work with this concept, it must first be specified what is to be understood when referring to shape in the remainder of the dissertation. Afterwards, the motivation given in Section 1.1 is extended. Some fundamental definitions that are necessary for the future discussion are given that follow from this motivation. Next, a number of requirements for shape abstractions and shape are examined that have been given by other authors. Based on the fundamental definitions, the motivation, and desired application areas a number of more adequate requirements are derived that are being referred to throughout the dissertation.

In the next step the concept developed so far is analyzed and specified with respect to the type of data the work aims to treat—polygonal meshes. Following from these considerations the concept is extended with respect to the criteria that will specify the notion of importance. A classification of these criteria is given and discussed in a summary and relations to the rest of the concept are examined.

2.1 Shape

The shape of an object typically refers to the form of its surface or silhouette. In this respect properties of objects such as texture, color, material, internal structure, etc. are not of importance. Sometimes the shape of an object is associated with the concept of gestalt. In this context the shape of an object can allow a person to recognize it because certain shape properties are typical for specific objects and familiar to this person. For example, a compact object with a thin and long structure as well as four short but thicker structures emerging may be recognized as an elephant having four legs and a trunk. However, the gestalt aspect of shape and the recognition of objects is not the topic or intention of this dissertation.

Instead, in this dissertation the shape of objects is analyzed mainly in terms of its geometry. However, also non-geometric properties of objects can be considered because these may also carry important information that is relevant for the underlying geometry. In particular, algorithms and applications in the area of computer graphics will be examined. With respect to this domain, the discussion will be restricted to two- and three-dimensional objects and their shapes.

Various attempts have been made previously to represent, classify, categorize, or evaluate shape. BLUM introduces his *medial axis transformation* for two-dimensional curve based objects [Blu67] in order to derive a new shape descriptor, KOENDERINK tries to conclude the shape of three-dimensional objects from their silhouette lines [Koe84] and introduces a new measure for analyzing local shape aspects of three-dimensional objects [Koe90, KvD92], KUNII discusses how to represent shape in a computationally valid model [Kun99], and EDELSBRUNNER and MÜCKE show how to use α*-shapes* to approximate the shape of a point cloud [EM94] to name just a few examples. A selection of approaches related to the topic of this dissertation will be reviewed and discussed in more detail in the following chapters.

The shape of objects as understood in this context can be represented in computer graphics in a variety of different ways. For example, a shape representation can be discrete (e. g., volumetric data) or analytic (e. g., boundary representation), explicit (e. g., spline surface) or implicit (e. g., implicit surfaces), and continuous (e. g., smooth subdivision surfaces) or piecewise linear (e. g., polygonal meshes). Other shape representations are, for example, *Supershape* in 2D and in 3D, *Superellipse*, and *Superellipsoid* [Gie03] that represent shapes entirely by parameters of a formula and are specifically used to represent natural shapes.

Due to the recent advance of graphics hardware, research on shapes in computer graphics has concentrated on three-dimensional shape representations, lately. As already mentioned in Section 1.1, particularly polygonal or more specifically triangular meshes are used to approximate 3D shapes because they can easily be processed by graphics hardware. In addition, most other three-dimensional representations can be converted into 3D polytopes. The algorithms introduced below operate specifically on these polygonal shape representations.

The shape analysis that will be performed by the algorithms introduced in the following has the goal to extract the essence of shapes. This means that the extracted shape information should facilitate, among other, feature extraction and shape abstraction. The following section discusses some motivation for the approach presented later as well as gives necessary definitions.

2.2 Motivation and Definitions

One of the biggest problems of shape processing and shape representations used in computer science is that most of them represent the data/information sampled at a very low level. This sampling means that the original, continuous data, such as the color distribution in an image or the surface form of a three-dimensional object is not represented by its original characteristics. Instead, it is represented by a number of sample points, often on a regular or almost regular grid, such as a pixel matrix or a regular triangulation, respectively.

It is obvious that most processes of acquiring the shape data cannot be changed very easily. Image scanners or the optical chips used in digital photography cannot easily be replaced by a more advanced technology. Due to the limitations of data acquisition hardware, the way volumetric three-dimensional objects are digitalized to 3D voxel data sets by, e. g., MRI or CT scans will not be simple to modify. Similarly, the acquisition methods of 3D boundary representations such as polygonal meshes through, for example, laser scans are not easily exchangeable.

Even worse, with the growing processing power of computer hardware the sizes of shape representations used in computer graphics is growing as well. Hence, for being able to classify the shapes, index them for a database search, compress them for storage or network transfer it is necessary to have efficient tools for telling important characteristics from unimportant ones. Thus, algorithms for feature detection are continuously becoming more important to find these significant parts of a shape.

Many algorithms have been presented which are able to find features in shape representations such as 2D pixel matrices, 3D voxel data sets, or polygonal geometric models, some of which will be reviewed and discussed in Chapter 3. Most of these algorithms, however, concentrate on a specific criterion for importance depending on the application that was envisioned. On the other hand, the concept of what is important in a geometric model and what is not differs from one application to another. For some applications, such as lossy compression of geometric models, it might be important to find ridges and creases on the surface of the model. Other applications may need to detect edges of an additional property, such as the border between two texture maps, to ensure the correct storage of those borders. A third group of algorithms could be interested in preserving the general shape of the objects to allow for shape matching. Hence, algorithms which have an inherent criterion are not flexible enough to be used for applications other than the one for which they were created.

Generally speaking, the examples given above show that it is vital to capture the essence of shape of the considered objects in many different application areas. This leads to the following definition:

Definition 2.1 *The data that is generated by an algorithm that captures the* essence of shape *in form of features that are extracted from an object and that make up the essence of its shape according to a certain criterion is called* Essential Shape Representation (ESR)*.*

The concept of essential shape representation as defined above is independent from a specific type of data as long as it complies to a certain set of requirements (see Section 2.3 below). Thus, there can be various specific ESRs and there are many possible ways to construct one.

Definition 2.2 *A specific way to construct an essential shape representation for a certain type of input data leading to a specific representation shall be called* ESR scheme (ESRS)*.*

Starting with these definitions and considering the example application areas given above a set of requirements for ESRs will be derived in the following section.

2.3 Requirements of an Essential Shape Representation

In order to be able to develop a new essential shape representation scheme or to evaluate previously presented ones it is necessary to specifically know what characterizes a good ESR. This means that a number of requirements for ESRs have to be defined.

In previous works on shape abstraction a number of people have given requirements before. For example, SUDHALKAR, GÜRSÖZ, and PRINZ formulate some requirements for skeletons in general [SGP93]. They base their constraints on properties of medial axes (that will be discussed in detail in Section 3.1.1 below) of objects and the application of skeletons in engineering:

- the skeleton should have no interior,
- the skeleton should have homotopic equivalence to the object; that is, number of holes, enclosed voids, etc. should remain the same, and
- the shape of the skeleton should abstract the shape of the object; that is, locations and relative dimensions of features should be the same as in the object.

LAM, LEE, and SUEN discuss requirements for thinning of two-dimensional discrete data [LLS92]. They state that the ideal algorithm should compress the data while retaining significant features. In addition, the algorithm should eliminate local noise without introducing distortion and the extracted structures should have unit width. In [LLS93], LEE, LAM, and SUEN refine this set for application to 2D skeletonization algorithms. They add, for example, that the preservation of connectivity, of topology, and of geometric properties is necessary. Similar requirements are also named by MA, WU, and OUHYOUNG [MWO03, Section 2]. They only add that a skeleton should be invariant under affine transformations. In general, these notions of requirements are very specific to the type of data and the structures extracted from it (i. e., skeletons). Therefore, a more general set of requirements will be defined below. However, it will also be based on the intended area of application of ESR.

In Sections 1.1 and 2.2 a number of motivations for building an essential shape representation were given and discussed. Proposed application areas of essential shape representations are numerous, some of which have already been named:

- feature extraction,

- (progressive) shape storage,
- shape transfer over networks,
- shape compression,
- shape comparison and shape matching,
- search in shape databases,
- shape animation,
- shape property analysis, and more.

From these application areas and their restrictions as well as Definition 2.1 it is now possible to derive a number of requirements that an essential shape representation in the context of this dissertation has to fulfill as follows:

1. In order to enable storage of the data, an essential shape representation has to allow for coding and decoding.

This means it has to be possible to construct an ESR scheme data structure from the shape and also to reconstruct a representation of the shape from that data structure. For example, this is required for (networked) applications where it is necessary to transmit shape information. A progressive nature of the representation facilitates the best possible reconstruction after network transfer with a possibly limited band-width.

2. At the same time, an essential shape representation has to enable compression of shape information, especially lossy compression.

This could be achieved by using a progressive encoding of the data and omitting less significant parts of the model to reduce the amount of information to be transmitted over a network or stored in a file.

The progressive nature of the shape representation can in turn enable matching of two shapes even if they are not fully geometrically identical.

3. Thus, essential shape representations should facilitate the search in shape databases on the basis of previously identified characteristic features of the shapes.

4. Related to these features, an essential shape representation should be a multi-scale representation.

This means that reconstruction can take place at various scales or different levels of detail. One could use it to either reconstruct only the most important features or also include less important ones to refine the reconstruction. In other words, it should be possible to select the size of the geometric details that are needed to be reconstructed. In a way this resembles the different stages of abstraction as shown in Figure 1.1.

5. In addition, an essential shape representation should allow for resampling of shapes, i. e., representing the shape independently of a particular sampling.

For example, a 3D boundary representation should be represented independently from the triangulation given in a specific mesh. This means that different meshes can be used for reconstructing the representation and that different tessellations lead to similar ESRs.

6. As for additional attributes, an essential shape representation should allow the assessment of geometric properties (such as curvature, volume, momenta, etc.) that are all independent from the sampling.

7. An essential shape representation should allow for editing the shape at the feature level, rather than at the sampling level.

This means that by applying a transformation to the essential shape representation the reconstruction of that transformed ESR should also be transformed appropriately compared to the original shape.

8. In addition, an essential shape representation should allow for animation at a more meaningful feature level than at the level of individual sampling points.

For example, 3D meshes would be animated using skeleton-based animation methods rather than specify the animation for each individual triangle vertex coordinate. Indeed, an essential shape representation can be the input for an animation system.

These requirements are independent from a specific ESR scheme. An ideal ESR scheme should fulfill all of these requirements. However, not all schemes deliver solutions for all requirements. Previously presented techniques for capturing the essence of shape are reviewed and compared with these requirements in Chapter 3 to see whether they can be used as essential shape representations.

As already noted, the essential shape representation as a concept is independent from the specific type of data it is applied to. Hence, a specific ESR scheme will be different for different types of input model data. An ESR scheme for a pixel image in 2D or a voxel data set in 3D will be different from an ESR scheme for three-dimensional polygonal model. This in turn means that the methods for constructing an essential shape representation will differ depending on the types of data. An essential shape representation, however, will have to fulfill the requirements stated and discussed above. In the following, the term essential shape representation will mostly be used in the context of 3D polygonal meshes if not otherwise noted. Thus, the following section discusses ESRs and ESR schemes specifically for polygonal meshes.

2.4 Essential Shape Representations for Polygonal Models

In the context of polygonal models, there are a variety of ESR schemes that can be thought of. These schemes can be roughly divided into two major groups:

- schemes that find structures inside of the objects and
- schemes that look for structures on the surface of the mesh.

The first group of algorithms seeks to condense the information that is present in polygonal objects to a graph of lines or curves which generally lies inside the original object. This graph is also commonly referred to as the *skeleton* of the mesh. In addition to the mere structure in the graph, the skeleton might also store additional properties about the object. These additional properties can be used, for example, to guide the later reconstruction of the original model. Although such a internal skeleton graph can capture the general shape of an object and, thus, serves as an abstraction of form of the shape, in most cases it is restricted in its capabilities for reconstructions in terms of important features of the surface. Hence, it fails to meet major requirements for ESRs (in particular Requirements 1 and 5: coding and decoding as well as resampling).

The second principle approach to generate essential shape representations for polygonal meshes is to look for significant structures on the surface of the mesh itself. I. e., the goal in this case is to capture details of the surface rather than the overall form. That means to search for a set of curves on the surface of the object as it is defined by the polygonal mesh. Similar to internal ESR schemes, the curves on the surface might also be associated with additional properties other than their mere geometry. Those properties could include, but are not limited to, curvature and curvature directions.

Of course, there are a variety of other ways to come up with a specific internal or external ESR scheme for polygonal models. However, they will all either construct an internal graph, look for structures on the surface of the object, or do a combination of both.

2.5 ESR Criteria

In order to be able to capture the essence of a shape, the first thing is to define the meaning of *essential*. The concept of what is considered important to be captured as essential differs significantly across applications. Thus, it is necessary to describe the importance of features with a certain criterion. In addition, it is vital that this importance criterion be interchangeable to allow for applying the same ESR scheme to different applications. Also, this allows to apply the same criterion to different ESR schemes. The criterion, thus, has to be kept separate from the ESR scheme algorithm that is used to capture the essence of shape.

Definition 2.3 *The criterion that defines what is important in a shape will be called* ESR criterion. *The choice of a specific ESR criterion is independent from the specific ESR scheme algorithm.*

This separation of ESR scheme and ESR criterion permits the use of virtually any criterion that might be needed for an application. It is always possible to define and implement new criteria that are needed for a specific application and use the new criterion in connection with a specific ESR scheme.

As argued above, many different criteria are possible that can lead to an essential shape representation depending on the application that employs the ESR. In particular, a classification for criteria that can be applied to specific geometric models to capture importance is discussed in more detail in Sections 2.5.1, 2.5.2 and 2.5.3.

2.5.1 Geometric vs. Non-Geometric Criteria

In previous works on feature extraction (see Chapter 3) mostly one type of criterion was used in order to determine what is to be considered as an important feature and what is not: *geometric criteria*. Geometric criteria only take geometric properties of a shape into account such as, for example,

- whether a volumetric point (voxel) is inside or outside the object,
- positions of surface points,
- surface normals,
- neighborhood relations,
- distances from structures such as the surface,
- curvature, or similar properties.

These geometric criteria are used in the literature to extract shape information such as feature lines on the surface of an object, in particular ridges or creases. Other criteria might be used to find object skeletons. An example for an application that uses the features found by applying geometric criteria is lossy compression that guarantees that the reconstruction is visually not or not perceivably affected. In this case the detected features would be stored with a higher priority than areas without features ensuring that the important parts of the model are correctly reconstructed.

There are, however, applications that require other types of criteria to find features of shapes that are not based on geometric properties. An example for such a *non-geometric* criterion is the interest of a user in a particular point and/or user interaction. This data could be derived, e. g., from text-based (such as TextIllustrator [SS00]) or interaction-based illustration applications (for example, ZoomIllustrator [PRS97] and AGILE [HS02]).

The feature extraction using such non-geometric criteria can be equally important for applications. Non-geometric criteria are not only more diverse but also more specific in their application. This is one reason that they are rarely covered in the feature detection literature. Some exemplary non-geometric criteria will be developed and discussed in Sections 4.6.3 and 4.7.1.

Geometric criteria, on the other hand, are more specific in terms of what features they allow to detect, namely geometrically interesting areas. The detection of these locations is important in many algorithms, e. g., shape compression, shape storage, shape transmission, and shape recognition. These algorithms are frequently used in shape processing. For example, mostly geometric criteria are used in previously proposed feature extraction techniques. Thus, geometric criteria are better studied and have a broad range of potential applications.

For the same reason the following sections and chapters will concentrate on geometric features as well. In principle, there are two main directions for capturing important aspects of shape in terms of geometric features: criteria that lead to an abstraction of the general form of the object and criteria that yield details on the surface of a shape. These will be discussed in turn.

2.5.2 Geometric Criteria for Abstraction of Form

The first group of geometric criteria for defining what is important in a shape focuses on large-scale properties. This allows a capturing of the overall shape of the considered objects. This means that the global form rather than local features is considered to be important. Thus, applying such a criterion leads to an *abstraction of form* of the shapes to which it is applied.

In general, abstraction of form criteria are *indirect criteria*, i. e., it is not possible to compute a scalar importance value for features directly. This is due to the fact that taking the whole object into account when evaluating a criterion will not lead to a locally defined importance value. Instead, abstraction of form criteria facilitate the decision for shape modifications which will lead to an ESR in the end.

Thus, abstraction of form criteria are often used for internal ESR schemes (Section 2.4) such as *internal skeleton extraction* methods. Those algorithms capture the general, global shape of objects for, e. g. later analysis of the topology of the objects.

2.5.3 Geometric Criteria for Surface Details

In contrast to abstraction of form criteria, the second group of criteria is used to find local structures that are important for an application. Thus, those criteria concentrate on *surface details* of a shape. This means that they consider a local environment only when evaluating the importance of a point. In fact, for computing an importance value related to a surface detail it is obviously only necessary to consider the local environment. This local character of surface detail criteria facilitates the *direct* computation of scalar importance values for features.

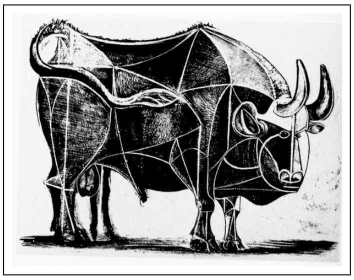
Criteria for finding surface details are typically used for external ESR schemes (Section 2.4), e. g., in so-called *feature extraction* algorithms. These techniques—in contrast to internal skeleton extraction methods—try to find important structures on the surface of shapes on a local scale.

2.6 Summary

In this chapter the concept of essential shape representations (ESRs) was introduced and defined. It was argued that in order to allow for a ESR scheme to find the essence in a shape an additional ESR criterion is necessary for specifying what is considered to be important. Moreover, it was discussed that the specific ESR scheme should be independent from the choice of a specific ESR criterion. This is a major constraint in the ESR concept in order to allow for maximum flexibility of ESR schemes and criteria in applications.

In addition, some requirements for essential shape representations were derived from the list of desired application areas. These requirements are specifically designed for three-dimensional essential shape representations. Since this dissertation focuses on polygonal shape representations in 3D, the two general groups of ESR schemes were distinguished: *internal* and *external* ESR schemes.

The criteria developed below in Chapter 4 mainly focus on direct geometric criteria for use with the external ESR schemes presented in Chapter 5. However, some direct non-geometric criteria will be addressed as well. In addition, in Chapter 3 a number of internal ESR schemes will be reviewed that use indirect criteria.



Chapter 3

Approaches to Capturing Geometric Features of Shapes

In computer graphics, there are numerous methods for capturing the geometric features of shapes. Besides mathematical and topological concepts to shape abstraction which will be discussed first in Section 3.1, the methods for feature extraction presented in this chapter will be reviewed based on the type of shape representation they are applied to. It will be distinguished between discrete shape representations (see Section 3.2) and boundary representations (see Section 3.3). In the former group, there are methods that work on 2D data (i. e., pixel images) and others that were developed for volumetric shapes (i. e., voxel data). While this classification is also possible for boundary representations, in practice primarily algorithms for 3D shapes are used. These can be classified as internal and as external approaches according to where the extracted features are located at.

For the description of extracted geometric features of a shape, in many applications the term *skeleton* is used. In particular for approaches that compute an approximation of a *medial axis* or a *Voronoi diagram* rather than the exact representation, *skeleton* is used synonymously to those two terms. In addition, sometimes the extracted data is referred to as *feature lines.*

3.1 Mathematical Concepts for Shape Abstraction

Although they are not often used in practice for feature detection or shape abstraction applications, mathematical concepts play an important role in many of the algorithms used in practice. In particular, the concepts *Voronoi diagram*, *medial axis*, and *Reeb graph* will be discussed in detail in this section because they are an essential part of many algorithms that will be introduced later.

Mathematically, the skeleton of a shape can be regarded as the set of singularities of the signed distance function from the surface [ABOK94]. Informally this means that it consists of the points that are equidistant to at least two points on the surface. This definition relates closely to Voronoi diagrams and medial axes that are going to be discussed first. A number of algorithms will be introduced for computing Voronoi diagrams and medial axes in two and three dimensions as well as some related concepts.

Afterwards, the Reeb graph being a structure for recording the topology of objects will be introduced. The topology describes objects in terms of properties that are not modified by continuous deformations. In particular, the genus of an object is one of these properties, i. e., how many “holes” the shape possesses. The abstraction provided by the Reeb graph as a topological description of objects leads to a number of algorithms that facilitate the capturing of essential shape information. Two examples for these algorithms will be discussed in the following as well.

3.1.1 Voronoi Diagrams and Medial Axes

Voronoi diagrams are an important and widely used concept in various areas of science, not only in computer graphics and computational geometry. A very comprehensive survey about Voronoi diagrams, their history, areas of application, and algorithmics is given by AURENHAMMER [Aur91]. He shows that Voronoi diagrams have been used in various areas as early as the nineteenth century.

A Voronoi diagram is a partition of n-dimensional space into *Voronoi regions* (see the example in Figure 3.1). Voronoi regions and Voronoi diagram are then typically defined as follows:

Definition 3.1 *Given a set E of* sites *in the n-dimensional Euclidean space with each site being a subset of this space. Then, the* Voronoi region *$VR(e)$ of a site e of E is considered the set of points that are at least as close to e as to any other site in E using the Euclidean distance d:*

$$VR(e) \equiv \{p \in \mathbb{R}^n \mid d(p,e) \leq d(p,e'), \quad \forall e' \neq e,\ e' \in E\}$$

The Voronoi diagram *$VD(E)$ is then defined as the union of the boundaries ∂ of the Voronoi regions:*

$$VD(E) \equiv \bigcup_i \partial VR(e_i)$$

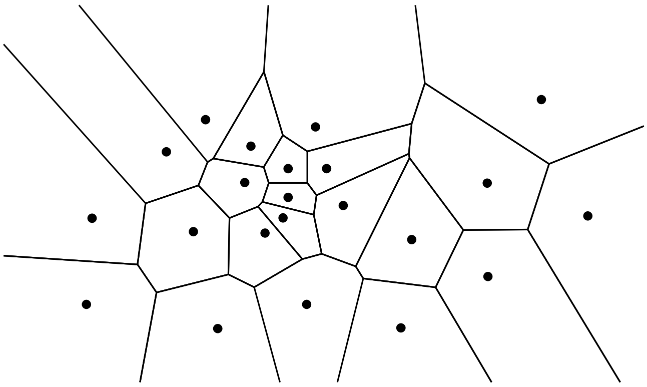


Figure 3.1: Example Voronoi diagram in 2D for a small set of points.

Typically, a Voronoi diagram is computed for a set of individual points in the two-dimensional plane as depicted in Figure 3.1. In this case the Voronoi diagram is a polygonal partition of the plane. Some of the Voronoi regions are open and some are closed convex polygons. Before discussing Voronoi diagrams of more general shapes, the concept of the medial axis will be explored in more detail.

The *medial axis (MA)*—sometimes called the *symmetric axis* or the *(mathematical) skeleton*—of a polygon is a generalization of Voronoi diagrams. It was introduced by Blum [Blu67] for finding and modeling new descriptors of shape. He uses the analogy of burning grassland (*grass fire metaphor*) where an object in the form of a patch of dry grassland is set on fire simultaneously at all points of its outer border. The fire works its way to the inside with the same velocity at all locations (see Figure 3.2). This grass fire metaphor is essentially a wave propagation starting from the border of the shape. The union of all points where two wavefronts meet define the medial axis of the original object (see examples in Figure 3.3).

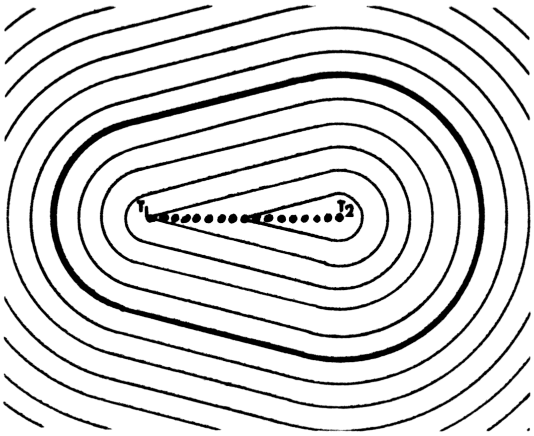
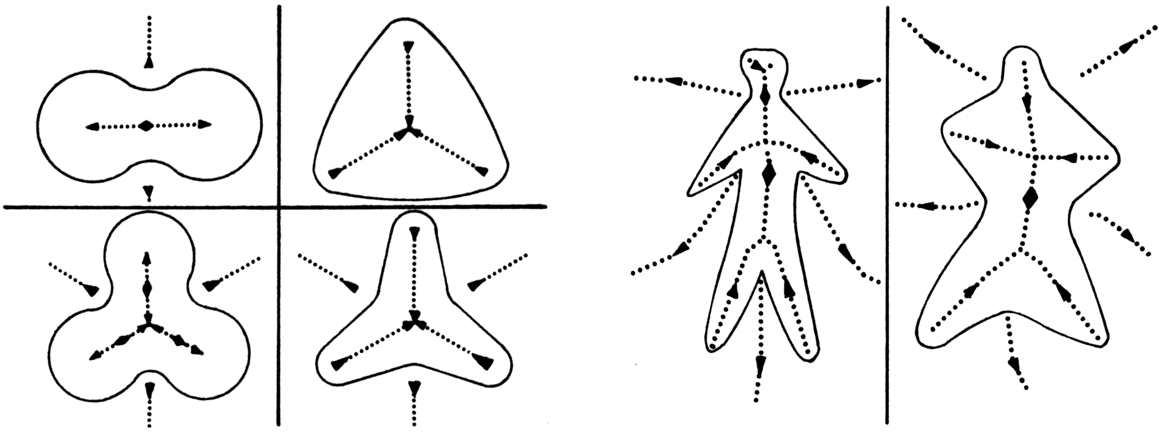


Figure 3.2: Deriving the medial axis of a shape. The medial axis is denoted by the dotted line. From [Blu67].



(a) Examples for simple shapes.

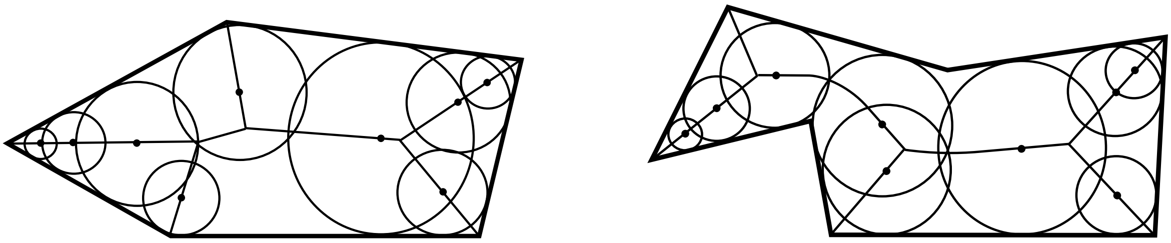
(b) Examples for more complex shapes.

Figure 3.3: Examples for medial axes of shapes. The medial axes are denoted by the dotted lines. From [Blu67].

Rather than using the previously discussed grass fire metaphor, the definition of a medial axis is typically based on *maximal balls* [FEdFC02]:

Definition 3.2 *Given S as a subset of the Euclidean space and its boundary B. A ball inside B that is not entirely contained in another ball inside B is called a* maximal ball*. The union of the centers of all maximal balls form the medial axis $MA(S)$ of the set S.*

The application of this definition yields a graph of connected, one-dimensional skeleton curves for a closed shape in 2D (see examples in Figure 3.4). For polygonal shapes, this graph is connected to every convex vertex on the border of the polygon but not to concave vertices (see example in Figure 3.4(b)). For convex polygons, the medial axis graph is also polygonal (see example in Figure 3.4(a)). For concave polygons, however, the medial axis graph contains parabolic curves (see example in Figure 3.4(b)). Finally, the medial axes of curved objects are usually curved as well. Only in cases where the curved object is (locally) symmetric with respect to a linear symmetry axis will the medial axis be (locally) linear. It must also be noted that the construction of the medial axis is reversible when the radius of every maximal ball is stored since this information is equivalent to the information contained in the original shape [Blu67]. By taking the idea of medial axis even further, this concept can be generalized to external medial axes, i. e., medial axes outside of shapes (see examples in Figure 3.3 where both internal and external medial axes are shown). This is done by looking for maximal balls outside of B. Interestingly enough, those external medial axes are only generated by points of the border of the shape that are not part of its convex hull [Blu67].



(a) Medial axis for a convex polygon.

(b) Medial axis for a concave polygon.

Figure 3.4: Medial axes of polygonal shapes in 2D. Some of the maximal balls are shown together with their respective centers.

To relate medial axes and Voronoi diagrams, one can compute the Voronoi diagram for a polygon and compare it to the medial axis of that polygon. However, considering a polygon as a set of infinitely many individual points[1] and following Definition 3.1, Voronoi regions are found in form of rays emerging from every point perpendicular to the respective polygon edge. Each of these Voronoi regions is also a Voronoi edge at the same time because each Voronoi region only consist of one line. Thus, the Voronoi region is equal to its boundary [FEdFC02].

To overcome this problem, Fabbri, Estrozi, and Costa give a generalized definition for Voronoi diagrams [FEdFC02]:

---

[1] One cannot consider the polygon as a whole to be the site that generates the Voronoi diagram since this would not produce any Voronoi edges at all.

Definition 3.3 *Given a set E of* sites *in the n-dimensional Euclidean space with each site being a subset of this space, then the* Voronoi region *$VR(e)$ of a site e of E is considered the set of points that are strictly closer to e than to any other site in E using the Euclidean distance d:*

$$VR(e) \equiv \{p \in \mathbb{R}^n \mid d(p, e) < d(p, e'), \quad \forall e' \neq e,\ e' \in E\}$$

The Voronoi diagram *$VD(E)$ is then defined as the complement of the union of all Voronoi regions resulting in a set of points where each point is equidistant to at least two Voronoi sites:*

$$VD(E) \equiv \left[\bigcup_i VR(e_i)\right]^c$$

This generalized definition of Voronoi diagrams solves the problem of Voronoi edges emerging from every point of a polygon edge since only points equidistant to at least two sites are considered to be part of the Voronoi diagram.

In addition, FABBRI, ESTROZI, and COSTA also give a generalized definition of medial axes [FEdFC02] that allows to also construct medial axes for open shapes such as open curves:

Definition 3.4 *Given S as a subset of Euclidean space. An open ball B is considered to be a maximal ball of S if it satisfies the following two constraints:*

1. *B belongs to the set G of the open balls that does not contain any points of S.*
2. *B is not a proper subset of any other ball in G.*

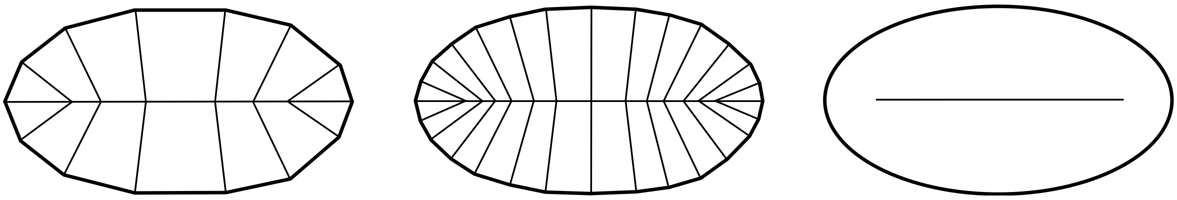
The union of the centers of all maximal balls that touch S in, at least, two points form the medial axis $MA(S)$ of the set S.

The generalized definitions of both Voronoi diagram and medial axis allow FABBRI, ESTROZI, and COSTA to establish a close relationship between both concepts [FEdFC02]. They show that, in fact, the medial axis is a special case of a Voronoi diagram.

Although BLUM also discussed polygonal shapes, he used the medial axis graph mainly for describing the general shape of curved, two-dimensional objects [Blu67]. This is due to the fact that, when computing medial axes (or Voronoi diagrams) for curved objects, the resulting graphs do not touch every convex vertex of the curved shape. Instead, they form a graph of curves which lies completely inside the curved object. This graph is locally symmetric to the shape [FEdFC02]. Hence, they are used as shape descriptions for these types of objects. BLUM also argues that the medial axis defines the natural shape properties of objects while being invertible [Blu67].

However, as discussed above and illustrated in Figure 3.4, both Voronoi diagrams and medial axis graphs connect to convex vertices when they are computed for polygonal shapes. Because polygonal shapes are usually approximations and polytopes approximating curved shapes have a lot of convex vertices, many of the vertices will be connected to branches of the medial axis graph. Even worse—the better the approximation gets the more of those branches will appear in the medial axis (see examples in Figure 3.5). This is caused by the fact that where the polygonal boundary is an approximation the corners are sampling artifacts.

This leads to a very important observation regarding the usability of medial axis representations for shape representation in the context of this work. Although the error introduced by polygonal corners of the shape's border gets smaller with a better approximation, the number of sampling artifacts grows. This, unfortunately, causes the number of branches resulting from the sampling to grow as well. This means that the medial axis representation of the ideal shape—a curved object—will differ significantly from the medial axis representation of very good approximation—a finely tessellated polytope (see Figure 3.5(b) and 3.5(c)). Thus, although a medial axis may be well suited as a shape representation for curved shapes, it is not as well suited for those polygonal shapes that represent approximations. However, most polygonal shapes (in particular 3D meshes) are approximations of smooth, curved shapes.



(a) Medial axis of a coarse polygonal approximation of an ellipse.

(b) Medial axis of a better polygonal approximation of the ellipse.

(c) Medial axis of the exact, not approximated ellipse.

Figure 3.5: Comparison of medial axes of different polygonal approximations of an ellipse and the exact ellipse.

A concept that generalizes the medial axis is the *straight skeleton*. It was first introduced by AICHHOLZER et al. in 1995 [AAAG95a, AAAG95b]. Similar to the medial axis the straight skeleton is a graph inside a polygon. It is defined as the medial axis of a polygon using the L_∞-metric. The straight skeleton is identical to the medial axis when computed for a convex polygon. However, in contrast to the medial axis which can contain curved segments for concave polygons, the straight skeleton only consists of linear edges for any type of polygon it is computed from (see example in Figure 3.6 which contains concave parts). The main advantages of the straight skeleton are its straight-line character and its lower computational complexity [AAAG95a, AAAG95b]. However, it is very sensible to changes of the constructing polygon—small changes can change the structure of the straight skeleton completely.

When generalizing the notion of the medial axis or the Voronoi diagram skeleton to the third dimension, the medial axis "graph" of a three-dimensional boundary representation turns out to be a set of connected surface sections rather than just one-dimensional edges as in the case of two-dimensional shapes (see example in Figure 3.7). Similar to medial axes of two-dimensional shapes, the medial axis of a three-dimensional convex polytope is a set of connected, planar polygons. Also analogous to two-dimensional shapes, the medial axis of polytopes with concave parts contains non-planar (quadratic)

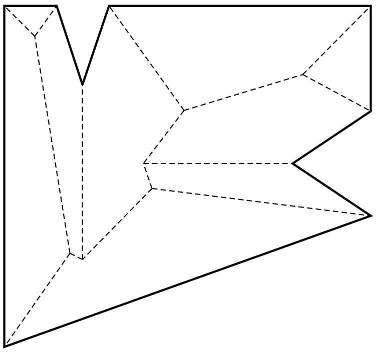


Figure 3.6: The straight skeleton of a concave polygon. Note that there are no curved parts in the skeleton. From [AAAG95b].

surfaces. In addition, the medial axis of a shape with a curved surface will be curved in most cases as well.

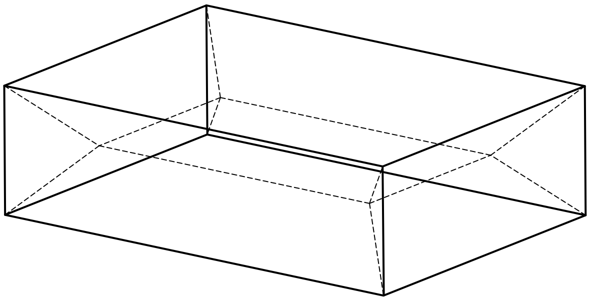


Figure 3.7: Example for the medial axis of a three-dimensional shape.

Sherbrooke, Patrikalakis, and Brisson present an algorithm for the efficient computation of the medial axis of 3D polyhedra of arbitrary genus which may also include non-convex edges and vertices [SPB95]. They classify the different parts of the 3D medial axis into *sheets* (2D elements), *seams* (1D elements, the borders of sheets), and *rims* (those seams that connect to the boundary of the shape) as well as *junction points* (0D elements where seams and sheets meet). To extract the medial axis, they recursively traverse the prospective MA starting at junction points. Each step identifies seams and points or new junction points. In the latter case, the new junction points are used for starting new recursions as long as they have not been visited before.

Etzion and Rappoport solve the problem of computing the 3D Voronoi diagram of piecewise planar 3D polyhedra by determining their symbolic (*Voronoi graph*) and geometric parts separately [ER02]. This algorithm is based on *proximity structure subdivision* which encodes the proximities of the polyhedron's parts and which is used to construct the Voronoi graph. In order to handle degeneracies in the Voronoi diagram, Etzion and Rappoport define and compute the *approximate Voronoi graph* in which degenerate and almost-degenerate parts are simplified. In order to approximate the geometric part of the Voronoi diagram to any needed accuracy, the previously mentioned space subdivision is used to construct the *proximity structure diagram.* This process is very stable and, in addition, can be applied locally. Thus, partial proximity information can easily be derived.

Sudhalkar, Gürsöz, and Prinz generalize the concept of medial axis given in Definition 3.2 in that they use it on general maximal cells based on a certain norm which does not necessarily have to be Euclidean. Using this generalized definition they construct a modified medial axis which they call *box skeleton* [SGP93]. It differs from the medial axis in that the L_∞-metric is used instead of the Euclidean metric. They apply this skeleton to shape abstraction for discrete objects, i. e., voxel based objects. Thus, the algorithm they present works on these kinds of objects only.

One main application area of three-dimensional medial axes or skeletons is engineering as discussed by Quadros et al. [QGRP01]. For example, the planning of forms for injection molding for producing plastic parts is named by several authors (e. g., [SGP93, QGRP01]). In this application area it is necessary to compute an abstraction of the original model (which often contains thin walls) based on two-dimensional primitives. This way the problem of flow of the injected mold can be reduced from three to two dimensions. This significantly simplifies the numerical simulation of the mold filling process while still remaining sufficiently accurate. Other application areas discussed by Quadros et al. in the realm of engineering comprise hexahedral mesh generation, feature recognition, and motion planning for material deposition [QGRP01].

However, for similar reasons as in the two-dimensional case (as discussed above) the 3D medial axes are not as useful for shape abstraction in the sense of the requirements laid out in Section 2.3. One important reason for this, as already mentioned for the 2D case, is that 3D medial axes of smooth objects differ significantly from those of good polygonal approximations. The numerous branches of the medial axis just capture the noise in the approximation rather than the essence of the shape. In addition, these branches caused by noise cannot easily be classified or rated for their significance for the overall shape of the object. Thus, since the medial axis is invertible it is not useful as an abstraction of shape. However, invertible shape representations can qualify as efficient shape abstractions if they have a progressive character that allow for omitting insignificant parts (see also Requirements 1, 2, 3, and 4 in Section 2.3).

Voronoi diagrams and medial axis are two topological concepts for capturing the geometric form of objects. A different topological concept is the *Reeb graph*. In contrast to the previously discussed concepts, Reeb graphs disregard the specific geometric shape of the surface and it are used to denote the pure topology of shapes according to a mapping function. They will be discussed in the following.

3.1.2 Reeb Graphs

Reeb Graphs were introduced by and named after Reeb [Ree46] and are used in homotopy and *Morse theory* [Mil63, SKK91]. They are used to represent the topology of objects as a skeleton graph. The Reeb graph is always associated with a continuous scalar mapping function which is defined on the given shape (often the height function is used). This function is used to identify critical points (maxima, minima, and saddle points) on the shape. These points and the connections between them are determined and represented in the Reeb graph (see example in Figure 3.8).

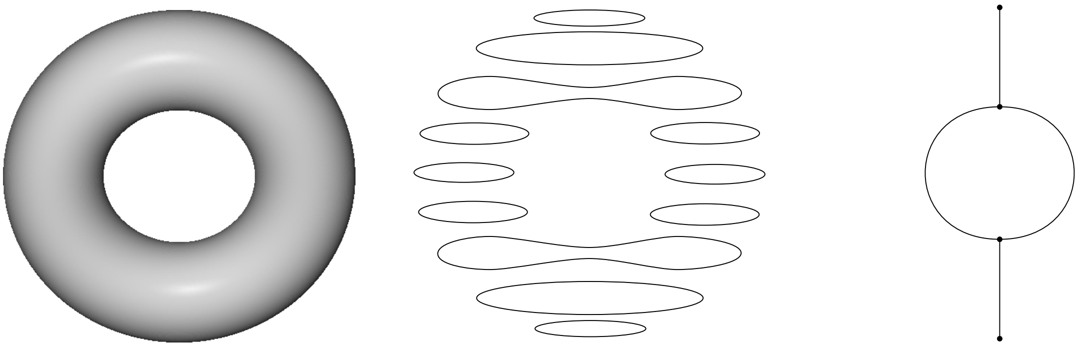
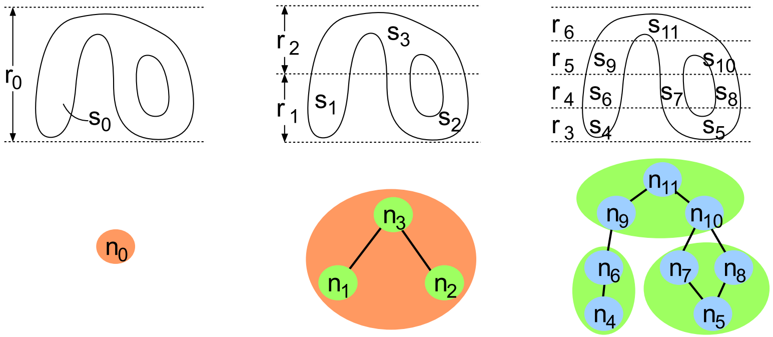

(a) Shape: torus. (b) Various levels of the height function. (c) Derived Reeb graph.

Figure 3.8: Deriving a Reeb graph of a torus.

The advantage of Reeb graphs is that they are strictly one-dimensional and, thus, are easy to handle. In addition, if the scalar function is defined appropriately, the Reeb graph is invariant to translation and rotation.

Hilaga et al. use Reeb graphs for extracting shape information from three-dimensional objects for shape matching [HSKK01]. In particular, they use the extracted shape information as a key for searching in shape databases. In this context, they extend the concept of Reeb graphs to *multi-resolution Reeb graphs* (MRG) (see Figure 3.9). This means that the Reeb graph is constructed hierarchically by a successive re-partitioning of regions. This way the MRG converges to the original Reeb graph.



(a) First level. (b) Second level. (c) Third level.

Figure 3.9: Deriving a Multi-resolution Reeb Graph. From [HSKK01].

Hilaga et al. use these MRGs based on geodesic distance measurement—the distance between two points on the surface—for the purpose of shape matching. The choice of using the sum of geodesic distances between a point and all points on the surface as the mapping function was made because it is necessary to have a function that is invariant to a number of transformations, for example, translations, rotations, and resampling (see example in Figure 3.10). In addition, the chosen function also does not require a certain source point for each shape. Thus, it is not necessary to rely on the unstable computation of such a source point.

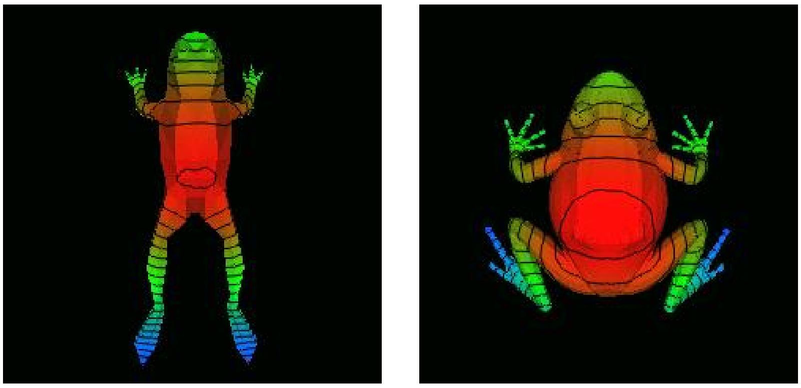


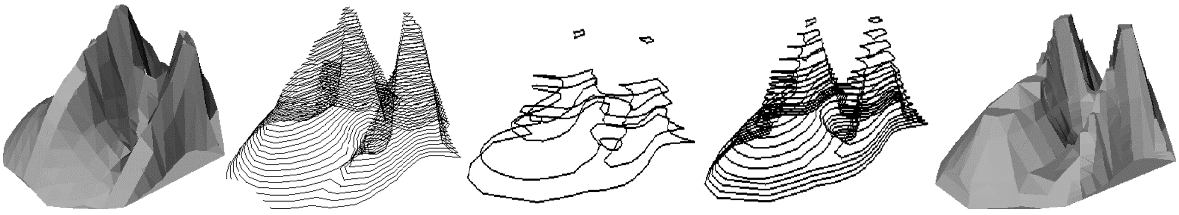
Figure 3.10: Color-coded geodesic distance for two similar objects used for multi-resolution Reeb graph shape matching. From [HSKK01].

The matching of two shapes now works as follows. First, the MRGs of both objects are constructed. Then, both MRGs are traversed hierarchically while trying to match the MRG nodes based on a similarity attribute computed from area and length. Using a database with 230 models, the authors show that the matching works efficiently and accurately. However, although the multi-resolution Reeb graph approach works well for matching 3D shapes, is does not allow for decoding, i. e., it does not allow to reconstruct the models from the MRG representation.

Chen and Ouhyoung show how to apply this MRG based shape matching to 3D object retrieval systems [CO02]. They discuss how to apply pre-processing, for example, to the handling of models with objects that have previously not been connected.

Biasotti, Mortara, and Spagnuolo also extend the basic Reeb graph concept for use in shape compression and reconstruction [BMS00]. This extension of the Reeb graph only encodes the adjacency relationships between a few critical cross sections of the shape using height as the mapping function. Critical regions are, for example, flat, connected areas of the models having the same mapping function value. First, these areas are detected (Figure 3.11(b)). Then, their influence zones are found. Finally, the adjacency relationships of the influence zones are coded into the extended Reeb graph. In addition, Biasotti, Mortara, and Spagnuolo discuss the reconstruction of the Reeb graph by showing how to find the best surface that fulfills the constraints of the constructed Reeb graph (see Figures 3.11(d) and 3.11(e)).

The original version of Biasotti, Mortara, and Spagnuolo's extended Reeb graph algorithm is restricted to terrain-like, not closed meshes. Biasotti discusses how to extend the concept to also allow for closed meshes [Bia01]. However, although this concept allows for shape coding and decoding as well as compression, it has not been shown whether it can be used for shape matching. In particular, small errors, for



(a) Original shape. (b) Extracted height levels. (c) Critical regions. (d) Reconstructed height levels. (e) Reconstructed shape.

Figure 3.11: Using an extended Reeb graph for shape compression. From [BMS00].

example in form of holes in the shape, may lead to a different topology of the shape which would make matching of intuitively similar shapes impossible.

As can be seen in the approaches described in this section, Reeb graphs can be used for representing shapes. They do not capture geometric properties but only the pure topology of a shape. Therefore, they are used rather for shape matching instead of shape reconstruction. However, some adaptations allow for extension of the concept to also facilitate shape reconstruction.

In general, topology based shape analysis or shape representation concepts such as Voronoi diagrams, medial axes, and Reeb graphs can be used to identify significant shape information. However, these concepts have certain limitations that have been discussed above. On the other hand, they are the basis of many algorithms used in practice that extract essential shape information from objects. A survey of these algorithms will be given in the remainder of this chapter. First, algorithms that can be applied to discrete data representations are going to be reviewed.

3.2 Feature Line Extraction from Discrete Data Representations

Discrete data representations are very common in computer graphics. They are characterized by sampling the data in most cases on a regular grid. Many modern data acquisition techniques (e. g., 2D image scanners, digital cameras, MRI scanners, CT scanners, and others) are based on capturing discrete data sets. The most common discrete data representations are two-dimensional pixel images and volumetric data sets. For both, various specific forms and optimizations exist. For example, space partitioning (such as quadtrees and octrees) can be used to improve the storage space efficiency of discrete data. Also, grids other than the common Cartesian lattice are sometimes used to improve efficiency.

However, most discrete representations may be converted or transformed into Cartesian lattice based 2D or 3D representations. Thus, algorithms for feature extraction from pixel images or volumetric data sets are by far the most common. Consequently, the methods discussed in this section are all working on either pixel or voxel data. The structures in form of thin lines or curves extracted from two- or three-dimensional sampled data are commonly referred to as *skeletons*. In addition, the term *thinning* is used almost synonymously to skeletonization (skeleton extraction) [LLS92].

3.2.1 Skeletons from 2D Pixel Images

There are a large number of methods for extracting skeletons from 2D images. The application areas for these algorithms are found, for example, in pattern recognition and shape matching (e. g., optical character recognition (OCR)). In particular *thinning algorithms* are used to extract these skeletons from sampled data. Lam, Lee, and Suen give a comprehensive overview of these 2D thinning methodologies used in image processing [LLS92].

The effect of thinning methods on two-dimensional shapes in pixel matrices is comparable to the effect of medial axis extraction from two-dimensional polygons (see Section 3.1.1). The process is based on the iterative deletion of successive layers of contour pixels from a shape until only a one pixel thick structure remains—the skeleton. Lam, Lee, and Suen distinguish between sequential and parallel algorithms depending on how they examine the configuration of pixels in the local neighborhood to determine which pixel gets deleted. Sequential algorithms use a fixed sequence in each iteration to decide whether a pixel gets deleted or not. Hence, the decision on whether to delete a certain pixel depends on the sequence of decisions that have been made thus far. In contrast, in parallel algorithms the deletion of a pixel depends only on the situation at the beginning of the iteration. Thus, the order in which the pixels are visited within an iteration is not important.

In addition to iterative thinning methods, Lam, Lee, and Suen also discuss non-iterative algorithms except those that are based on medial axis and distance transforms. Non-iterative algorithms, for example, include scan-line based methods which trace sections of black pixels and connect them in order to form skeletons. Other methods use window operations to determine strokes locally and connect them in order to form the skeleton. Also, run length coding of the image can be used to extract a skeleton based on finding consecutive intervals of black pixels which are of approximately the same size and more or less collinear.

Finally, Lam, Lee, and Suen discuss a number of requirements for the described thinning algorithms [LLS92]. They name processing speed, preservation of topological and geometric properties, reconstructability, and invariance under rotation. The discussed algorithms are evaluated and judged according to these requirements.

In [LLS93], Lee, Lam, and Suen conduct a quantitative and qualitative evaluation of 20 widely used skeletonization algorithms (see examples in Figure 3.12). In particular, they evaluate their computation times, sensitivity to noise, similarity to a reference skeleton,[2] reconstructability, and how to implement them for parallel computing. However, they cannot determine one single algorithm which performs best for all criteria. Instead, the performance of the algorithm always depends on which criteria are important for the specific application. In [LS95], Lam and Suen concretize their results and conduct a similar survey of parallel thinning algorithms specifically for the application in optical character recognition (OCR).

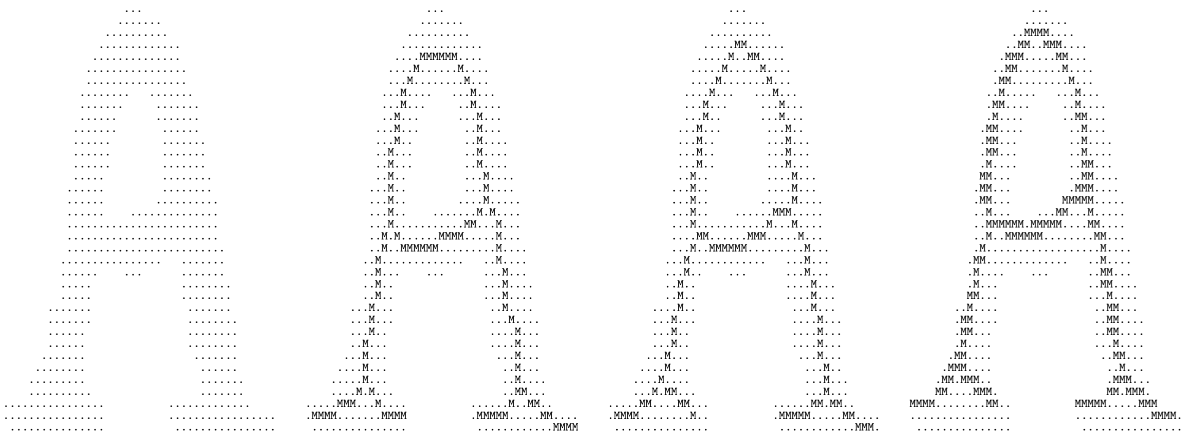


Figure 3.12: Original shape and selected results (using ARCEL, HILDI, and PAVLI) from Lee, Lam, and Suen's survey of skeletonization algorithms. From [LLS93].

3.2.2 Skeletons from Volumetric Data

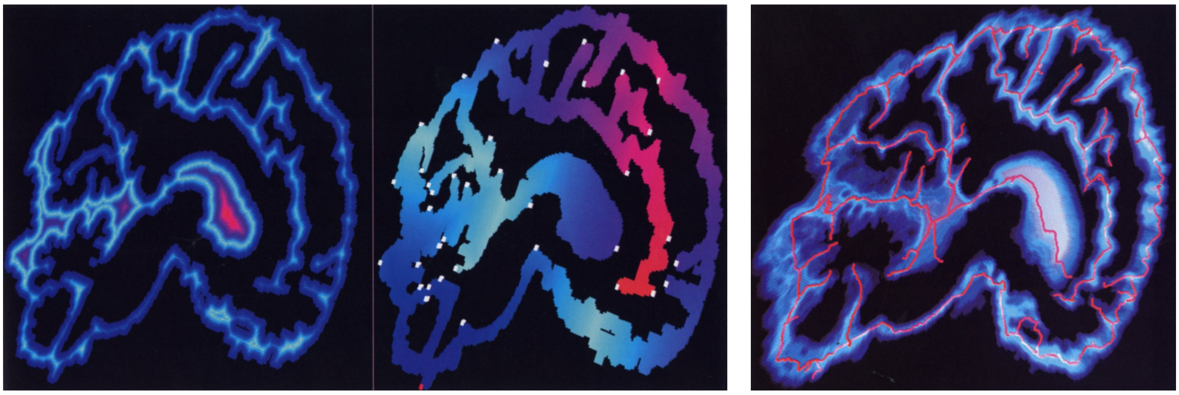
There are many application fields where volume data is generated and processed in the form of 3D voxel images, in particular medical imaging and technical visualization. Examples for such domains in medical visualization are *Computer Tomography* (CT) and *Magnetic Resonance Imaging* (MRI). Other examples in technical visualization are numeric simulation of physical processes. For example, the physical processes that take place in fuel cells can be captured in a volumetric data set including the temperature, the pressure, as well as flow directions and used for visualization [Röb03]. In a variety of areas of scientific visualization it is important to find significant and meaningful structures in the volume data. In these cases, skeleton extraction algorithms are typically applied to the voxel models to find these structures. The first step in this process is the segmentation of the 3D data for being able to treat different parts of the model independently. Afterwards, skeleton extraction can be applied to the individual structures. This makes it possible, for example, to evaluate the topology of the blood vessel system [HSEP00].

For extracting skeletons from volumetric data Zhou and Tonga use a voxel-coding methodology [ZT99]. Their goal is to identify connected, one voxel thick voxel paths representing the skeleton of the volume data. Their method first assigns two values to every voxel: the *boundary seeded field* value and the *single seeded field* value. Both

---

[2] The reference skeleton was obtained as the average of skeletons resulting from a survey conducted with a group of people.

fields are generated using voxel coding and a specific distance metric. For a specific voxel and its neighbor, this process increases the integer value assigned to the current voxel by an increment according to the chosen metric. The new value is assigned to the neighbor voxel if its previous value is lower than the new value. For boundary seeded coding (*BS-coding*) the process starts with all boundary voxels as seeds while for single seeded coding (*SS-coding*) it starts with a single voxel as the starting point. Thus, the single-seeded field divides the 3D object into clusters. All pixels in one cluster have the same distance from the seed voxel according to the chosen metric. Then, in each cluster the voxel with the highest boundary seeded field value is identified marking it as the medial point of the cluster. All these medial points together form the *cluster graph*—a connectivity preserving skeleton of the volumetric object (see example in Figure 3.13).



(a) Boundary seeded field and single seeded field (both color-coded) of a brain data set.

(b) Resulting skeleton of the brain data set.

Figure 3.13: Example for computing the skeleton from the volumetric data set of a brain. From [ZT99].

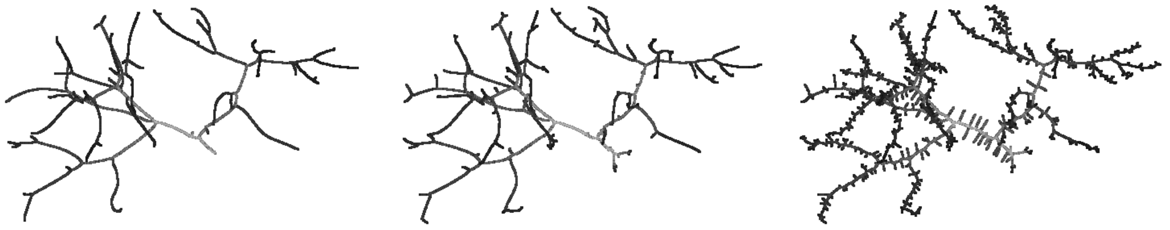
Gagvani, Kenchammana-Hosekote, and Silver extract skeleton trees from volume data for volume animation [GKHS98]. Similar to the previous method, their algorithm is also based on distance transforms. They first compute the skeleton by thinning the object (also see 2D thinning algorithms discussed in Section 3.2.1). This process yields a set of non-connected skeleton voxels. In order to have a structure for easy animation of the volumetric object, they re-connect the skeleton voxels to form a *skeleton tree*. The constructed skeleton is used for animation by first deforming the skeleton and then reconstructing the volume from it.

A slightly different technique that is also based on *thinning* was introduced by Selle et al. [Sel00, SPSP02]. They determine an internal skeleton based on a medial axis transformation for three-dimensional volumetric objects.[3] However, the three-dimensional analogon of the medial axis (see Section 3.1.1) consists of a set of medial surfaces. In some application domains (such as planning of liver surgery in connection with the liver's blood vessel structure as discussed by Selle et al.), on the other hand, instead

---

[3] The method can also be applied to two-dimensional pixel images.

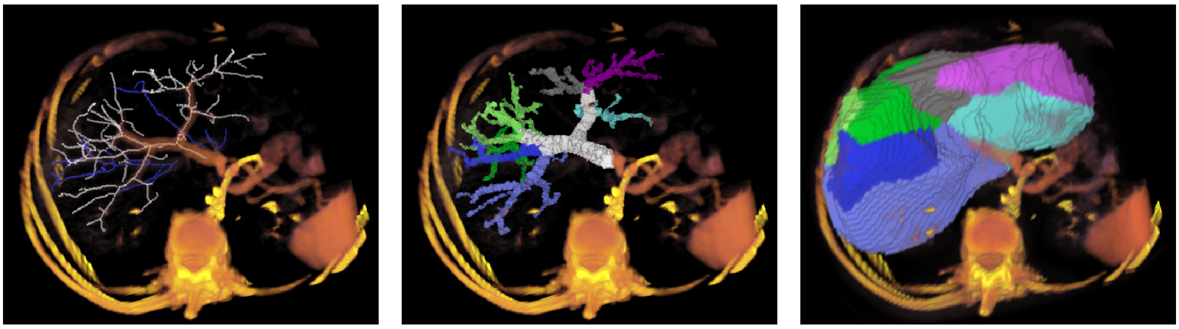
of extracting a set of its medial surfaces it is necessary to find a medial axis or a graph of medial edges for a three-dimensional object. Thus, Selle suggests to use a *discrete medial axis transformation* in 3D [Sel00]. Similar to the definition of the medial axis transformation, this method is based on the grass fire analogy. Starting from the outer boundary of the three-dimensional volume model, border voxels are successively removed (i. e., eroded) in a symmetric manner while preserving the object's topology. This process is called *symmetric erosion.* The medial surfaces that usually result from thinning three-dimensional objects are reduced through this process to medial axes. In addition, the development of so-called *irrelevant medial axes* that are caused by sampling artifacts can be controlled by adjusting a relevance threshold (see Figure 3.14).



(a) $t = 0.0$.
(b) $t = 0.8$.
(c) $t = 0.9$.

Figure 3.14: Influence of the threshold t on the generated internal skeleton using Selle's algorithm. From [SPSP02].

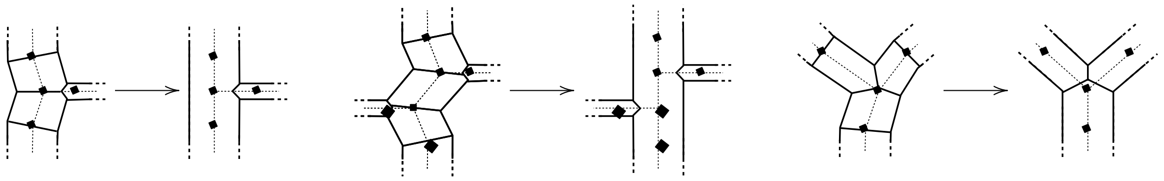
Selle et al. apply this algorithm to determine discrete medial axis transformations for segmented models of human organs [SPSP02]. The generated skeleton can be used to analyze the structure of the organ's blood vessels. The resulting blood vessel skeleton data is then used for operation planning to evaluate the blood support for the remaining organ tissue after an operation (see Figure 3.15).



(a) Skeleton.
(b) Color-coded skeleton.
(c) Color-coded liver parts.

Figure 3.15: Using skeletonization to determine which parts of the liver are supplied by which branch of the portal vein. From [SPSP02].

In addition, the generated skeleton can be used to visualize a complicated structure at interactive frame-rates as suggested by Hahn et al. [HSEP00]. To be able to visualize, for example, the complex blood vessel systems of a human liver interactively, first its skeleton is constructed in a pre-processing step. In an additional step, artifacts in the skeleton such as the undesired form of the skeleton at bifurcations can be removed (see examples in Figure 3.16). Also, the skeleton itself can be pruned and reduced in complexity for later speed-up of the visualization. Afterwards, the modified skeleton is used to generate the visualization using 3D rendering primitives such as frusta (see Figure 3.17).



(a) Small branches. (b) Successive small branches. (c) Symmetric bifurcation.

Figure 3.16: Removal of artifacts in the skeleton at bifurcations. From [HSEP00].

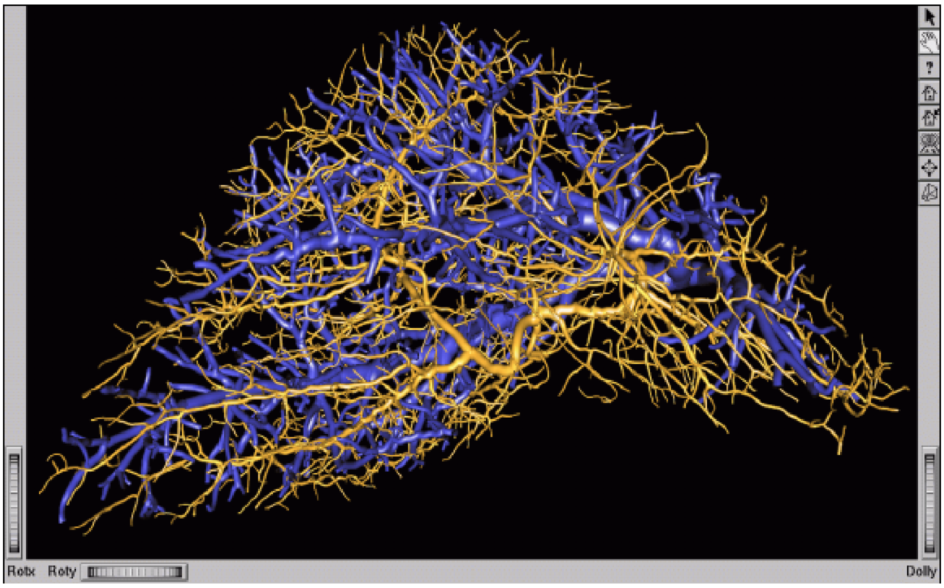


Figure 3.17: Visualization of the skeleton using frusta. From [HSEP00].

Wink, Niessen, and Viergever review additional techniques for extracting a central vessel axis from three-dimensional data sets generated in the medical domain [WNV00]. They distinguish between *indirect approaches* and *direct approaches*. Indirect approaches perform preprocessing of the volumetric data prior to extracting the central axis, e. g., segmentation of the data set into objects, while direct approaches extract the axis directly. In their own approach, Wink, Niessen, and Viergever start with at least two user-specified starting points on the central axis and iteratively compute the next points. One iteration first determines a candidate for the next central axis point. Then the central axis is extended with this point and a perpendicular plane to this estimated axis is computed. On this plane, the center of the vessel is determined and used as the new point for the central axis. Since this approach does not need any pre-processing such as segmentation it is a direct method.

The discussed methods and similar approaches are well suited for extracting internal skeletons from volumetric models. In addition to these methods for extracting internal structures of volumetric datasets there are also methods for extracting surface features. For example, Thirion and Gourdon use a method they call *marching lines* [TG92, TG93, TG96] that is based on the *marching cubes algorithm* [LC87]. In a different technique, Interrante, Fuchs, and Pizer base the identification of ridge and valley lines on a surface on the approximation of the principal directions and curvatures [IFP95].[4]

However, using any of the methods discussed in this section with polygonal models would involve a separate step of creating a volumetric representation from the polygonal data. This step, besides causing additional time and memory consuming computation, would introduce new sampling artifacts. There are, on the other hand, algorithms that compute skeletons directly from polygonal models. These methods are discussed in detail in the following sections.

3.3 Extracting Internal Features from Boundary Representations

There are a variety of shape description types for representing the boundary of three-dimensional objects. For example, one can use free-form surfaces, mathematical shapes, implicit surfaces, and polygonal meshes. The most common type of shape description for three-dimensional objects in current computer graphics, however, is the polygonal mesh representation. As already argued in Section 1.1, this results from the growing hardware support for the display of such data in modern PCs. This is caused, at least in part, by the ever-growing usage of 3D worlds in computer games and the resulting hardware development. In addition, most other types of shape descriptions can be translated into a polygonal form. Even discrete volumetric data can be transformed into polygonal representations. For example, voxel data can be converted into polygonal meshes using the *marching cubes algorithm* [LC87].

The translation of other types of shape representations into polygonal meshes, however, introduces sampling artifacts in most cases, e. g., there is no correct polygonal representation for a sphere. Regardless how fine the tessellation of the sphere's polygonal mesh gets, it is always only an approximation of the sphere.

In the previous sections, the distinction has been made between 2D and 3D shapes. While this classification is possible for boundary representations as well, feature extraction techniques for two-dimensional boundary representations (i. e., polygons) are not used in practice very often.[5] Thus, the approaches introduced in the following are dealing with three-dimensional boundary representations (i. e., polygonal meshes).

---

[4] For a comprehensive overview of algorithms for the analysis of volumetric datasets see, for example, Lohmann [Loh98].

[5] For Voronoi diagram and medial axis computation for polygons see Section 3.1.

3.3.1 Medial Surface or Voronoi Diagram Based Algorithms

A group of algorithms for extracting skeletons from polygonal models is based on computing the medial axis or Voronoi diagram for the model (also see Section 3.1.1). From this data a graph consisting of one-dimensional lines or curves is extracted which forms the skeleton of the polygonal shape.

TEICHMANN and TELLER base their skeleton extraction algorithm on the computation of the three-dimensional Voronoi diagram of the mesh's vertices [TT98]. This Voronoi diagram is simplified to yield the final skeleton. During this process of simplification, TEICHMANN and TELLER rely on user interaction for selection of a number of Voronoi vertices as the ends of the expected features of the object respectively the skeleton. Their algorithm then proceeds and reduces the medial axis graph by removing edges without disconnecting the graph until a tree structure emerges (see examples in Figure 3.18). Finally, the skeleton is connected to the surface for later animation of the model.

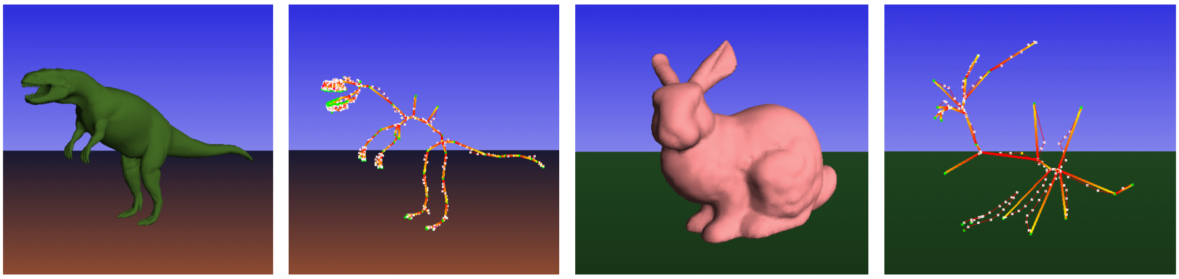


Figure 3.18: Shapes and their skeletons as computed by TEICHMANN and TELLER. From [TT98].

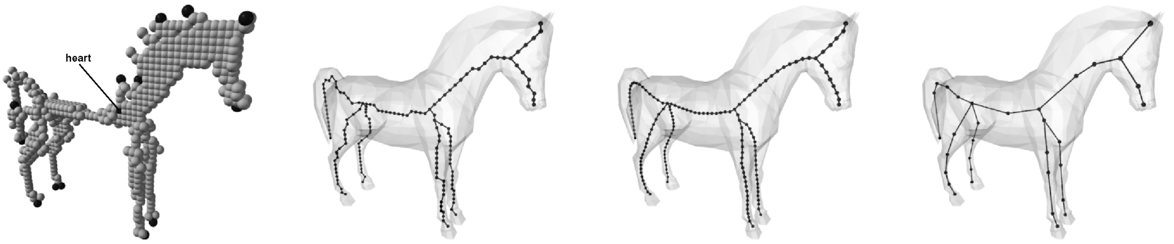
BLOOMENTHAL and LIM use a method for skeleton extraction which they call *direction testing* [BL99]. This implicit technique is based upon the definition that skeleton vertices are those points where the direction to the nearest point on the surface changes. They find and connect the points using a piecewise-linear implicit surface polygonizer. For points on a lattice the direction to the nearest surface point is computed. For each pair of directly connected lattice points where this direction differs sufficiently, the skeleton vertex is determined by binary subdivision.

WADE and PARENT present an algorithm for extracting skeletons from polygonal models based on the transformation of the polygonal data to the discrete domain [WP00, Wad00, WP02]. Then, the problem is solved in voxel-space and the skeleton is transformed back into continuous space. After the voxelization of the object, a discrete medial surface (DMS) is computed (see Figure 3.19(a)). Starting from the central voxel, a path tree is constructed[6] that ends in the voxels farthest away from the center (see Figure 3.19(b)). Next, this path tree is smoothed (see Figure 3.19(c)) to remove the artifacts that result from the regular grid. Finally, the path tree is converted into the

---

[6] Note, that always a tree is constructed. Thus, any information about the topological genus is destroyed in the skeleton.

skeleton by converting segments in the tree (sections between junction points or junction points and end points) into skeleton edges. If necessary, these edges are recursively split to reduce the error that is introduced by the conversion (see Figure 3.19(d)).



(a) Voxels of the DMS. (b) Path tree. (c) Smoothed path tree. (d) Skeleton.

Figure 3.19: Constructing the skeleton using Wade and Parent's algorithm. From [Wad00].

The main application for skeleton extraction as discussed by Teichmann and Teller [TT98], Bloomenthal and Lim [BL99], and Wade and Parent [WP00, Wad00, WP02] is the automatic generation of bones for the animation of polygonal meshes. This means that in addition to the mere skeleton extraction, all algorithms must also make the connection from the skeleton graph to the object's surface in order to be able to turn the motion of the skeleton into motion of the object. The data structure that captures this relation is commonly called *I-K-skeleton.*[7]

In contrast to the previous approaches that extract a skeleton in form of connected line segments, Amenta, Bern, and Kamvysselis present an algorithm to approximate the Voronoi diagram of a 3D shape [ABK98, AB99]. Their goal is to find an approximation for the smooth surface that is given by a number of points on that surface. They accomplish this through the extraction of the Voronoi vertices of the surface. They note that in 2D for a number of vertices approximating a smooth curve, the Voronoi vertices approximate the medial axis. However, in 3D this does not hold anymore. In the 3D case Voronoi vertices can be located very close to the surface. These vertices are not part of the medial axis of the smooth surface the mesh is approximating.[8] Thus, in order to find vertices that do approximate the medial axis, the authors suggest to only use so-called *poles*—the two Voronoi vertices farthest away from the surface point, one on either side of the surface. These resulting points can now be used to reconstruct an approximation of the medial axis.

3.3.2 Edge Collapse Based Algorithms

The algorithms introduced previously are based on the topologic concepts such as the Voronoi diagram and the medial axis. They compute their internal feature represen-

---

[7] I-K stands for *inverse kinematics.*

[8] An example for this is a group of four co-planar vertices that are located on a circle. In this constellation a Voronoi vertex would be generated in the center of the four points which is not part of the medial axis.

tation explicitly, i. e., in one pass. In addition to these type of algorithms, there is another major group for extracting internal features from polygonal meshes. In contrast to the previous one, this group—the edge collapse based algorithms—is based on continuous simplification in order to extract the internal features. This means that a simplification step is iteratively applied to the polygonal mesh until the internal feature representation emerges. This will be discussed in detail in the following.

Simplification of Polygonal Meshes

In [HDD$^+$93], Hoppe et al. introduced an algorithm for successive simplification of polygonal meshes. It is based on an *edge collapse* operation (see Figure 3.20). In this operation, one edge from the mesh is collapsed to a single vertex. This reduces the mesh typically by two faces, three edges, and one vertex. Thus, by applying this operation iteratively to the polygonal mesh it is successively simplified. The order in which the edges are selected is determined by trying to minimize an energy function.

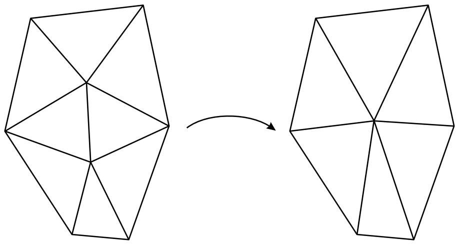


Figure 3.20: Edge collapse operation.

The process of applying successive edge collapse operations terminates once the desired simplification level is reached (i. e., the model is simplified to a certain number of polygons, edges, or vertices) or before the mesh becomes degenerate. As can easily be seen, this process is reversible. By recording every edge collapse step and applying the reverse operation—the *vertex split*—the original mesh can be reconstructed. Therefore, Hoppe et al. suggest to use this process for progressively storing polygonal meshes [Hop96]. To achieve this, the simplified version of the mesh is stored along with the necessary vertex split operations that are needed to reconstruct the detailed mesh.

Edge Collapse Based Skeleton Extraction

Raab suggests to use the edge collapse based simplification method to generate an internal skeleton for polygonal meshes [Raa98, DHR$^+$99]. Instead of stopping the algorithm before the mesh becomes degenerate, Raab continues to apply edge collapse operations. The modified algorithms stops for an edge if this edge is not attached to a valid polygon anymore. The remaining degenerate edges form the internal skeleton of the original polygonal mesh. The process and the result of an edge collapse based skeleton extraction is demonstrated for an example in Figure 3.21.

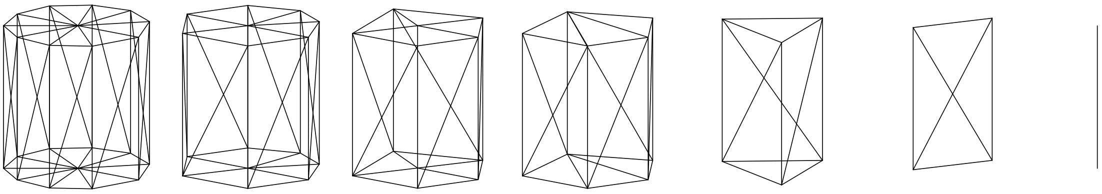
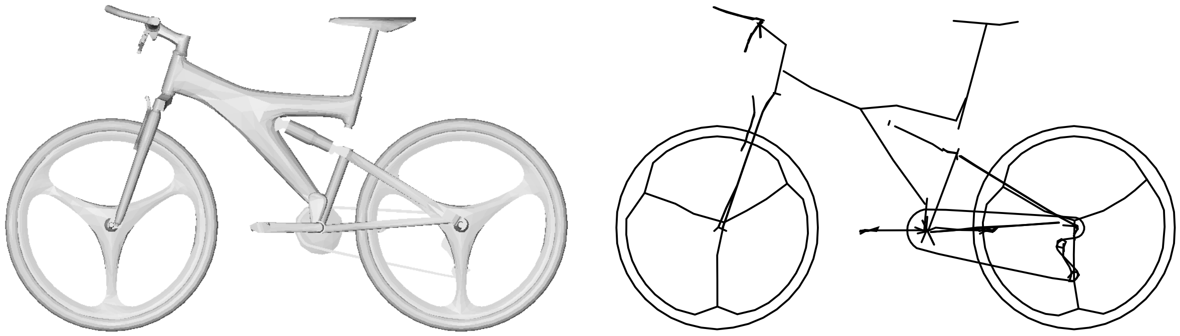


Figure 3.21: Example for successive edge collapse operations resulting in the skeletonization of the polygonal approximation of a cylinder.

In order to be able to approximate the global structure of the original polygonal mesh, Raab suggests to always select the locally shortest edges for collapsing. However, this local character or the algorithm in terms of selecting just the shortest edge can lead to unwanted results. In particular, for not evenly tessellated meshes the local approximation in the skeleton extraction algorithm can lead to a bad approximation of the global form of the objects. In those cases the skeleton would "move" from the finely tessellated to the coarser parts. In order to prevent this, Raab assigns a flag to each vertex that is initially set to `false`. Now, the edges are examined in sorted order from the shortest to the longest. An edge can only be collapsed if neither of its vertices are tagged. For every edge that has been looked at and that could not be collapsed because one vertex had been tagged before, the remaining vertex gets tagged as well. Once an edge has been found with neither vertex tagged, this edge is collapsed and the resulting vertex gets tagged. This process continues until all remaining vertices are tagged. Then, the flags of the remaining vertices are again set to `false` and the process starts over again. This modified algorithm works better in approximating the global form of objects (see example in Figure 3.22).



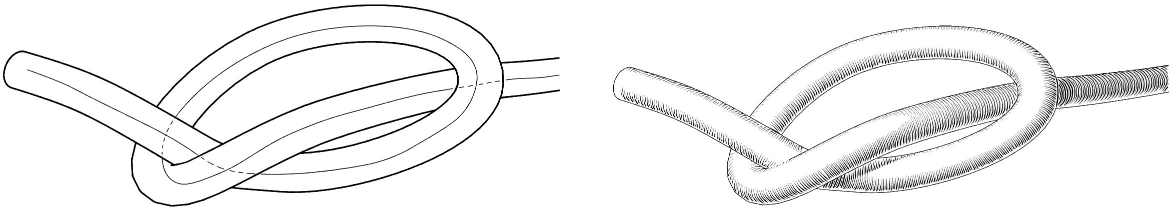
(a) Shaded model of the bicycle.

(b) Skeleton of the bicycle.

Figure 3.22: Model of a bicycle and the skeleton that was extracted from it using successive edge collapse operations.

The edge collapse based skeleton extraction technique is very practical for models that are piecewise tube-like. In this case, the algorithm will yield predictable results that are very much like what one would intuitively expect. In contrast, for objects that are more compact and sphere-like the results are insufficient and not quite predictable. In this case, the resulting skeleton graphs are highly dependent on the specific triangulation of the mesh. Small alterations of the triangulation may lead to serious differences in the skeleton.[9]

A possible application of the algorithm was presented by Deussen et al. in the context of non-photorealistic illustration [DHR+99]. First, their system computes the skeleton of objects. It is used for deriving the orientation of crosshatching lines in order to convey shading in line drawings. Then, they compute intersections of planes perpendicular to the skeleton with the objects. Finally, they draw lines along these intersections to achieve the crosshatching illustration style (see Figure 3.23).



(a) Sketch of the model with the skeleton. (b) Resulting crosshatched illustration.

Figure 3.23: Example for the application of the skeleton extraction algorithm in non-photorealistic rendering. From [DHR+99].

Another potential application is, for example, the computation of so-called *bones* for animation purposes as previously discussed in Section 3.3.1. However, due to the previously discussed limitations of the algorithm the skeletons would have to be revised. Hence, the skeleton could serve as a first estimate of the bone structure that would have to be adapted by the animation artist, afterwards.

Solving the Problem of Separated Model Parts

One problem with the edge collapse based skeleton extraction algorithm is that a separated skeleton graph is generated when the original geometric model consists of segmented parts. This problem can be overcome by an algorithm presented by Garland and Heckbert [GH97, GH98]. Based on the edge collapse algorithm by Hoppe et al. they describe a method to simplify polygonal models which can operate even on seriously segmented data. This would be advantageous for skeleton based animation of models because it allows treating the model as a union although the underlying mesh data is segmented into patches or objects.

In contrast to the original edge collapse technique, the modified algorithm not only looks at edges that are in fact present in the model but instead considers *valid pairs*. Those are, in addition to model edges, potential edges in form of pairs of vertices which

---

[9] However, even for piecewise tube-like shapes the resulting skeleton can be very sensitive to the specific tessellation as will be demonstrated in Section 5.1.

are less than a certain user-specified distance apart. When applying edge collapse operation to these valid pairs during the simplification process, this causes the algorithm to merge previously non-connected parts of the model.

Furthermore, Garland and Heckbert do not use an energy function or look for the shortest edge to be collapsed next but instead use an approximation of the potentially introduced error as the criterion. According to the authors this yields better results.

Similar to Raab's modification of the plain edge collapse method one can use Garland and Heckbert's modified algorithm for skeleton extraction. The edge collapse operations are continued until only a set of degenerate edges remain. Again, these make up the skeleton of the polygonal model.

Further Improvements

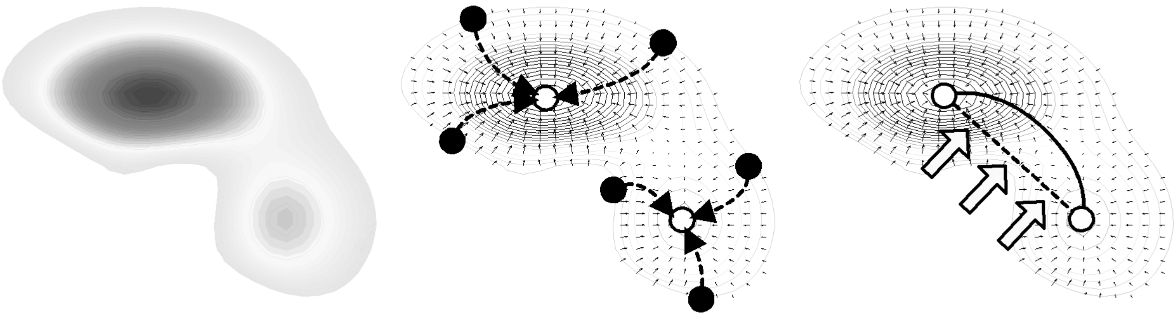
Ronfard and Rossignac discuss an edge collapse algorithm which also tries to improve the quality of simplification by using an estimated error as the selection criterion [RR96]. Their method is based on finding the edge which introduces the smallest estimated geometric error when being collapsed.

Sander et al. introduce an additional edge collapse algorithm with different simplification behavior [SGG^{+}00]. This algorithm makes sure that no part of the simplified version of a model is inside the more complex version in any step. This means that the model is "growing" while being simplified. This might be useful for clipping algorithms where a simplified version of an object can be used for a first clipping test instead of a bounding object (bounding box or bounding sphere).

3.3.3 Radial Basis Function Approach

Ma, Wu, and Ouhyoung choose a different approach for extracting an edge based skeleton from polygonal shapes [MWO03]. Their algorithm is based on *radial basis functions* (RBF) that are computed for the given 3D model (see Figure 3.24(a)). Afterwards, a gradient descent from each vertex of the model is invoked to find local maximum positions. To do this, a gradient vector field is constructed using the partial derivatives of the RBF (see Figure 3.24(b)). The local maxima positions found using this method are grouped into clusters using a given range. Then, any two connected maxima clusters are linked with a snake. Finally, the potential energy of the snake is minimized to relocate it in order to be centered in the object (see Figure 3.24(c)). A few examples of computed skeletons are shown in Figure 3.25.

This skeleton extraction algorithm, in contrast to the previously discussed techniques, is capable of finding skeleton curves that are central to the model.



(a) The RBF computed for an object.
(b) Deriving a vector field and finding local maxima.
(c) Linking the clusters with snakes and relaxing them.

Figure 3.24: Computing internal skeletons using RBFs and snakes. From [MWO03].

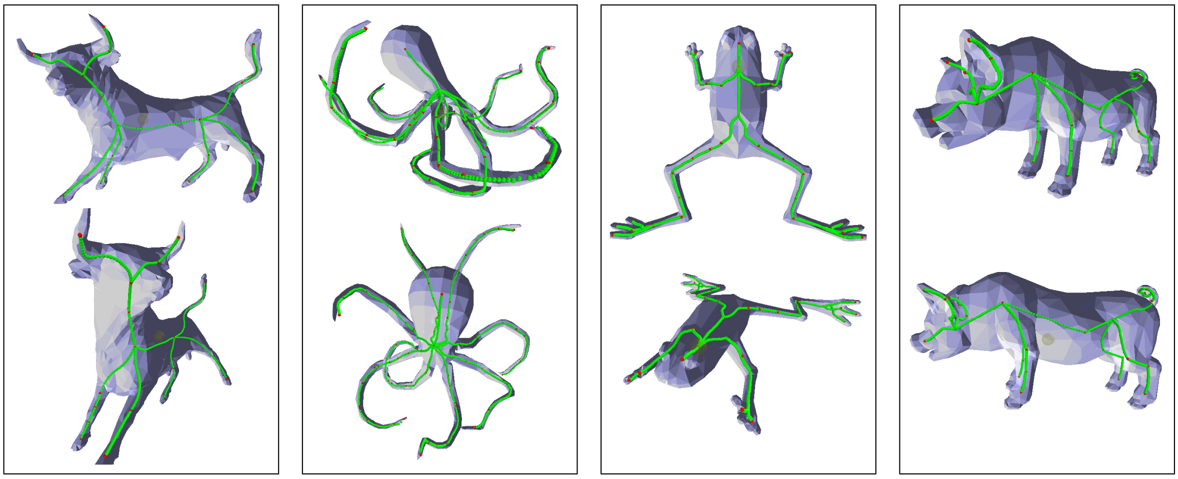
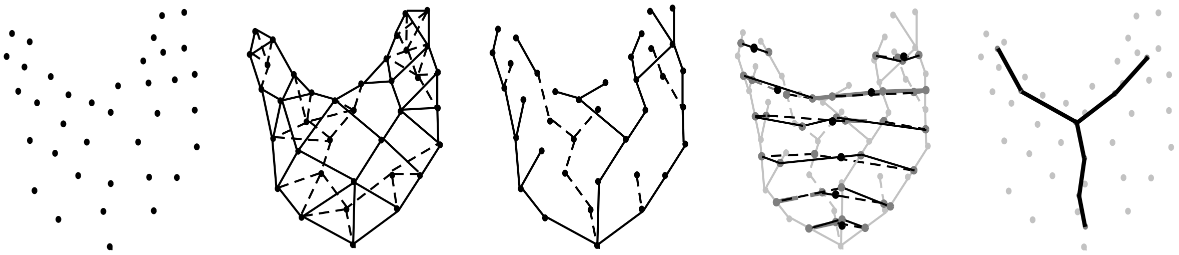


Figure 3.25: Examples for the approach based on radial basis functions and snakes for skeleton extraction. The red spheres denote the maxima clusters and the green spheres the location of the snakes after minimizing their energy. Courtesy of Wan-Chun Ma, from [MWO03].

3.3.4 Skeletons From Point Clouds

Besides extracting shape information from polygonal modes it is also possible to use point clouds instead. Although point clouds contain less information about a shape than polygonal meshes they can still be used to extract an internal skeleton from a shape. In addition, it is always possible to generate point clouds from polygonal meshes by sampling points on the triangles, either by using the mesh's vertices or even by generating more sample points on edges and faces. In order to be able to extract a meaningful skeleton it is only necessary that the points are located on the surface of the unknown shape and that the sampling is sufficiently dense.

Verroust and Lazarus describe an algorithm that extracts a skeleton that consists of a graph of piecewise linear skeleton edges from a point cloud [VL97a]. Their algorithm is based on the *cylindrical decomposition* of the point set (see Figure 3.26(a)) meaning that the skeleton is constructed as follows. First, a *neighborhood graph* with order n is constructed from the point set using a voxel decomposition of the space (see Figure 3.26(b)). This means that the graph consists of edges (p, q) where p is one of the n nearest neighbors of q or vice versa. From this graph, a *geodesic graph* is constructed (see Figure 3.26(c)) starting from a user-defined source vertex by computing the shortest paths in the neighborhood graph between the vertices and the source vertex. Afterwards, the geodesic graph is used to determine k level sets of the distance map with k being user-defined (see Figure 3.26(d)). Thus, the points in a level set are at the same distance from the source vertex. The points in a level set are connected and connected components are found. Finally, the centroids of the connected level set components are connected hierarchically to form the skeleton graph (see Figure 3.26(e)). Figure 3.27 shows some examples of extracted skeletons.



(a) Point set. (b) Neighborhood graph. (c) Geodesic graph. (d) Level sets. (e) Extracted skeleton.

Figure 3.26: Computing a skeleton from a point cloud. From [VL97a].

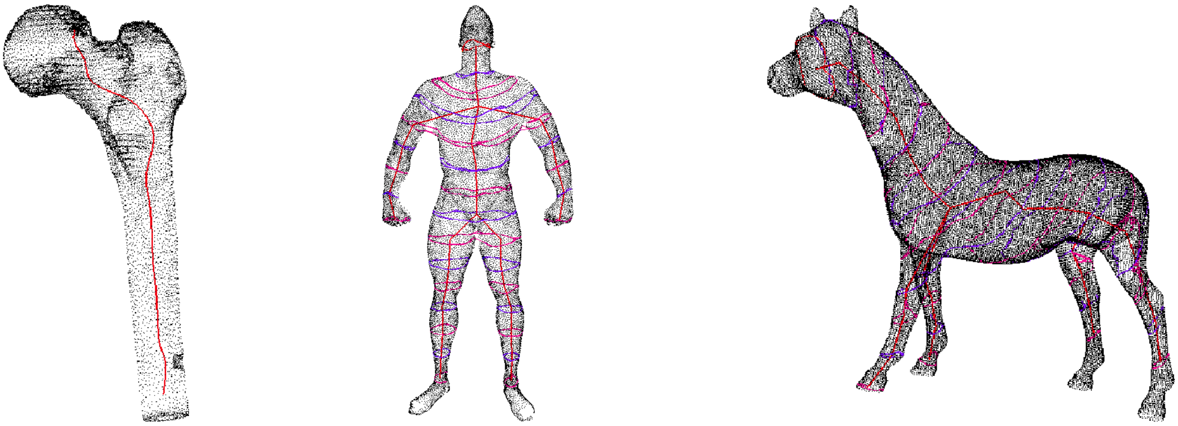


Figure 3.27: Examples for skeleton extraction from point sets. From [VL97b].

In general, it can be observed that the algorithms discussed above for extracting internal features from boundary representations produce a curve based structure. Being computed by an internal ESR scheme, this graph captures the global form of objects rather than surface properties. This means that internal skeleton graphs facilitate the analysis of form and topology of shapes but usually do not perform well when used for reconstruction even if additional data is stored with the skeleton. Therefore, the following section explores the extraction of external features from boundary representations.

3.4 External Feature Extraction from Boundary Representations

Instead of looking for a skeleton inside of shapes, sometimes it is necessary to find significant structures on their surfaces. In most cases, these structures are called *feature lines*. Since surface features of two-dimensional objects are simple, non-connected vertices, in most cases it only makes sense to look for structures on the surfaces of at least three-dimensional objects. The following sections discuss a number of approaches which extract such structures from 3D polygonal shapes.

3.4.1 Feature Lines Based on Approximated Curvature

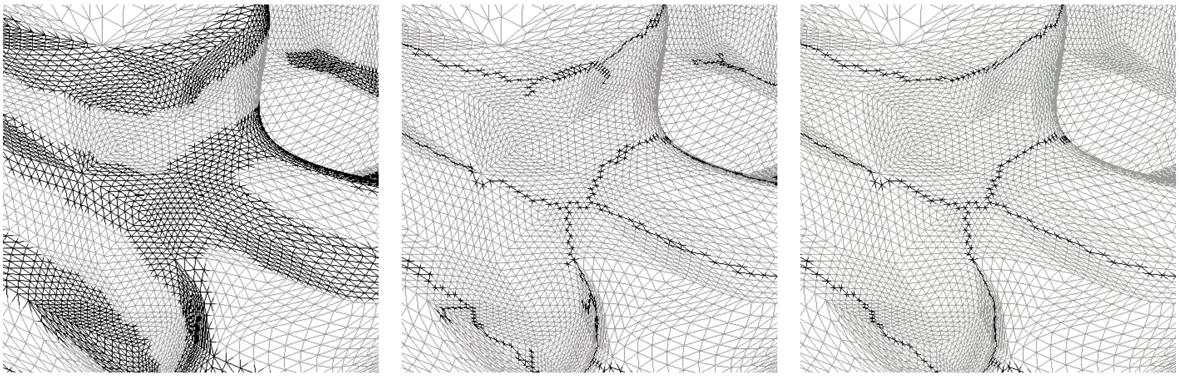
In [RKS00], Rössl, Kobbelt, and Seidel present a method for extracting *feature lines* from surfaces of polygonal meshes. They suggest to use *first* and *second principal curvature*[10] κ_1 and κ_2 in order to denote regions of interest. Since curvature is only defined for smooth surfaces, they develop a method for determining the *discrete curvature* of vertices of a polygonal mesh [Rös99, KBB$^+$00] (for details see also Section 4.1.3). In order to estimate the discrete first and second principal curvature and their associated *first* and *second principal directions* (the curvature directions associated with the first and second principal curvature are called *first* and *second principal direction*), they approximate the local neighborhood of the vertex (the vertex and all its direct neighbors) by a biquadratic Taylor polynomial. The necessary parameterization is obtained either from projecting the vertices' neighbors onto its tangential surface (as given by its normal vector) or by using an exponential map. The local approximation of the surface is done by a least squares solution of a linear system. This yields the Taylor coefficients which can be used to estimate the normal vector and the second fundamental form. From this, the principal curvatures and their directions are derived.

Having determined the local principal curvatures as the notion of importance, a threshold is applied to the significance measure in order to segment the surface into important and dull regions (see Figure 3.28(a)). The skeleton extraction described by Rössl, Kobbelt, and Seidel is based upon the repeated application of two morphological operators: *dilation* and *erosion*. Based on the dilation and erosion operator, the authors define the *opening* (first apply dilatation and then erosion) and *closing* operator (first apply erosion, then dilatation). Both are used to remove noise and artifacts from

---

[10] The maximal and minimal curvature in a point are called *first* and *second principal curvature*, respectively.

the segmented features. In order to extract the skeleton from the determined features, they are iteratively eroded (see Figure 3.28(b)) while respecting certain non-removable vertices—*complex vertices* which have been already determined to be part of the skeleton. In an additional step, pruning can be applied to the skeleton in order to remove certain unwanted artifacts (see Figure 3.28(c)).



(a) Segmented surface. (b) Derived skeleton. (c) Pruned skeleton.

Figure 3.28: Deriving an external skeleton as presented by RÖSSL, KOBBELT, and SEIDEL. From [RKS00].

WATANABE and BELYAEV extract so-called *salient curvature features* from polygonal meshes [WB01]. They also base their approach on the approximation of the principal curvatures κ_1 and κ_2 at the vertices of the surface. Their method, similar to the previous one, uses the direct neighbors of a vertex and a Taylor series to obtain the principal curvatures. However, they do not compute the principal directions and they extract sets of feature triangles instead of feature edges. Based on the principal curvatures at the vertices, the authors construct the two *focal surfaces* of the mesh. For every point on the surface, the point on the focal surface is located on the normal in that point with a distance of $\frac{1}{\kappa_i}$ from the given point. Thus, a triangle on the surface is associated with a focal triangle.[11] Note that there are two focal surfaces—one for each principal curvature.

After approximating the principal curvatures, WATANABE and BELYAEV detect the triangles where the curvature is extreme. They base this detection on the observation, that extreme curvature lines on the surface correspond to ribs—i. e., sharp bends—on the focal surfaces. Thus, they determine the curvature extrema by comparing the size of the regular triangles with the size of their associated focal triangles and selecting only triangles with the smallest 30% of the ratios. However, compared with analytically computed extrema of curvatures, the new approach still needs smoothing and filtering which is applied afterwards. Finally, a thinning algorithm is applied that is very similar to methods applied to discrete 2D data (see Section 3.2.1). This yields the final set of feature triangles (see examples in Figure 3.29).

---

[11] For theoretical background on this approach see BELYAEV, ANOSHKINA, and KUNII's work [BAK97].

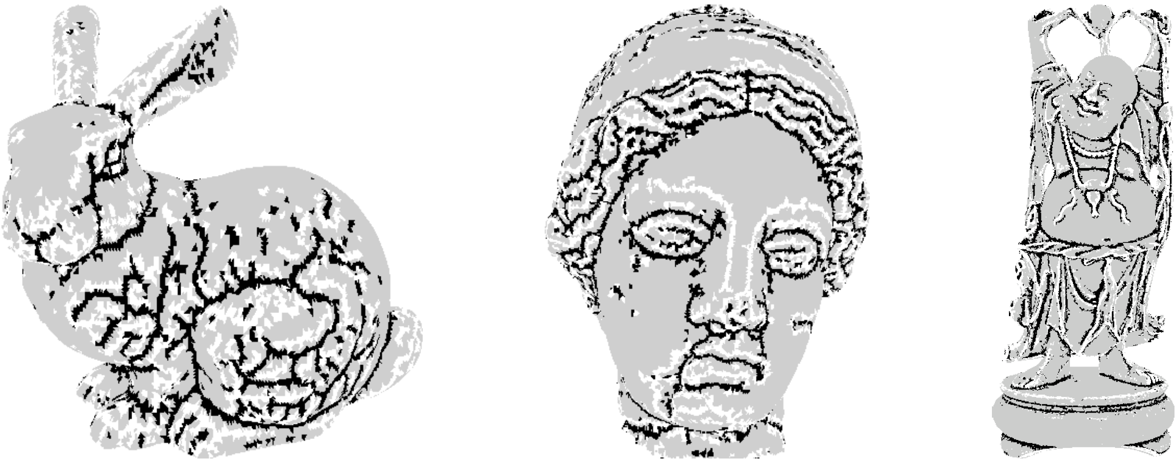


Figure 3.29: *Salient curvature features* extracted by Watanabe and Belyaev. From [WB01].

Watanabe and Belyaev show how to apply their approach for enhancing mesh decimation methods such as the one presented by Garland and Heckbert [GH97]. By modifying the selection of edges to be collapsed (see edge collapse technique in Section 3.3.2) to favor those edges which are not part of the extracted features they improve the quality of the resulting mesh.

In a slightly different approach, Belyaev and Ohtake use methods from image processing and apply them to polygonal meshes in order to find ridges and ravines on the mesh's surface [BO00]. They use algorithms for detecting edges in gray-scale images to find maxima on the approximated principal curvature. Unwanted artifacts are suppressed by thresholding. In contrast to the previous approaches, Belyaev and Watanabe are able to find the features with sub-triangle precision by interpolating the approximated principal curvatures along the triangle edges. In addition to mesh quality measurement and improving mesh decimation, they suggest applying their approach (using the also approximated principal directions) for pen-and-ink rendering by rendering small strokes along the detected ridges (see example in Figure 3.30).

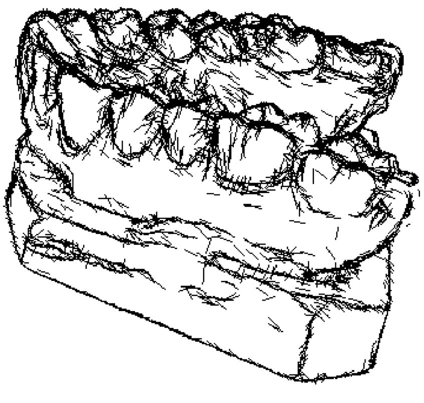


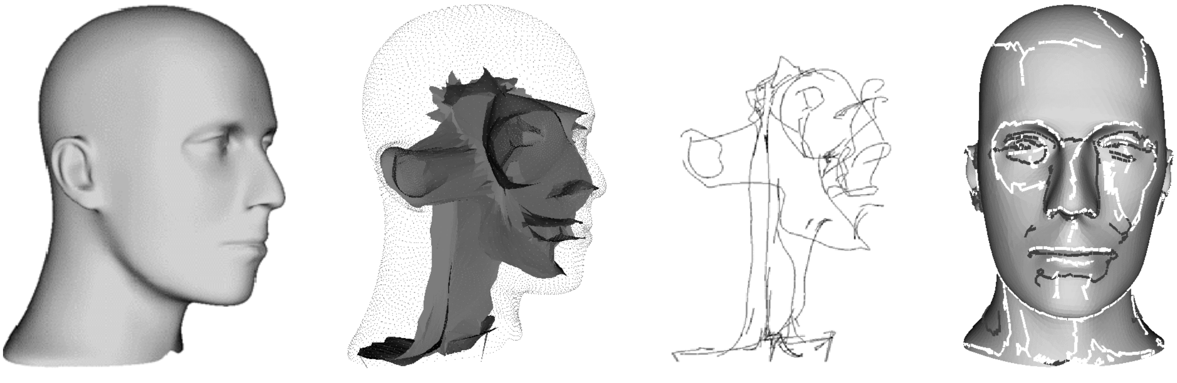
Figure 3.30: Applying strokes to detected ridges to generate a pen-and-ink rendering. From [BO00].

In a way, these methods resemble algorithms that extract skeletons from discrete 2D data. They first segment the mesh—either by vertices or by triangles—and then apply some kind of thinning similar to the methods discussed in Section 3.2.1. However, since the surface on which the algorithms operate is three-dimensional, the resulting features are three-dimensional as well. In addition, since 3D meshes are usually not defined on a regular grid, neither are the extracted features.

3.4.2 External Feature Lines From Internal Skeletons

Hisada, Belyaev, and Kunii pick up the idea of constructing an internal skeleton in form of a 3D Voronoi diagram of a 3D surface [HBK01, HBK02] (as previously discussed in Section 3.1.1). They show how to construct such a 3D Voronoi diagram efficiently for polygonal surfaces (see Figure 3.31). Their algorithm is based upon computing the Voronoi diagram for a set of points which are uniformly distributed on the surface. For each point on the surface, the Voronoi poles of its Voronoi region are determined (the two Voronoi vertices farthest away from the point). These poles give a good approximation of the shape's skeleton (i. e., its medial axis; for details see Section 3.3.1 as well as [ABK98] and [AB99]). Having determined these poles, Hisada, Belyaev, and Kunii reconstruct the skeleton by using the connectivity in the original triangle mesh (see Figure 3.31(b)). An edge in the skeleton between two poles is generated if and only if there is an edge between the corresponding vertices in the triangle mesh.

In order to find ridges and ravines on the surface of the mesh, the authors employ singularity theory and look for corresponding sharp edges in the approximated skeleton (see Figure 3.31(c)). The threshold that is used to decide whether an edge is sharp or not is determined by an statistical approach. Thus, the histogram of the cosine of the angle between two faces sharing an edge in the inner skeleton is analyzed and a certain percentage of the highest angles determines the threshold. This allows extraction of the desired surface features (see Figure 3.31(d)).



(a) Original shape. (b) Approximated Voronoi diagram. (c) Sharp edges in skeleton. (d) Extracted surface features.

Figure 3.31: Extracting surface features using an approximated Voronoi diagram. From [HBK01].

3.4.3 Approach Requiring User Interaction

A different approach to feature extraction from polygonal meshes is taken by Lee and Lee [LL02]. They use an adaptation of the snake algorithm used in image processing which they call *geometric snakes.* A geometric snake is an active contour in form of a parametric curve on the surface of the polygonal mesh. It moves over the surface while minimizing its energy function. In contrast to the previously discussed methods, the geometric snakes approach requires user interaction. Initially, the user has to specify the starting position for the snake. Thereafter, the algorithm works automatically. The energy of the snake which is to be minimized by moving the snake along the surface consists of two major terms: an *internal* and an *external* energy. The internal energy takes care of keeping the snake smooth while the external energy is used to locate the features where it is locally minimal. Instead of directly moving the snake on the mesh's surface, the snake along with its local neighborhood of the mesh is mapped onto a 2D plane. Using the original values of the energy function the snake is now moved on the 2D plane which makes the algorithm more efficient. After a number of iterations, the snake is mapped back onto the mesh. If necessary, the process can be repeated several times or the user can interactively make adaptations. Some examples for feature lines that were extracted from 3D models are shown in Figure 3.32. In these examples, normal variations between adjacent triangles are used to define the feature energy.

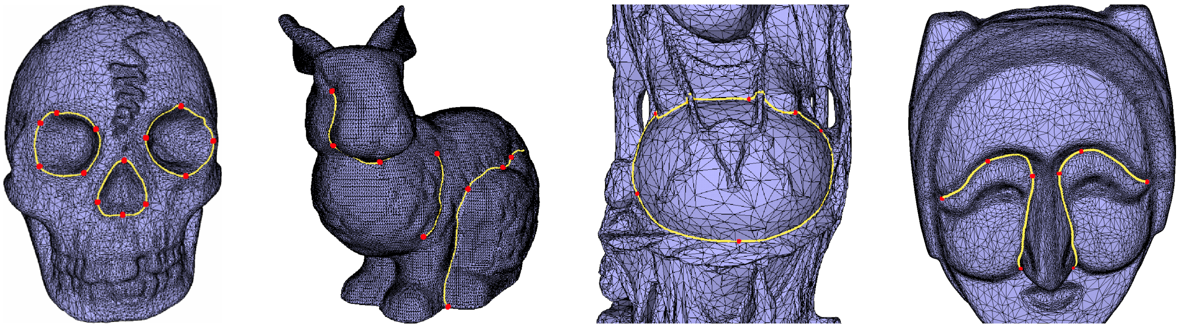


Figure 3.32: Examples for feature extraction with *geometric snakes.* From [LL02].

As it has been demonstrated, most approaches that endeavor to extract the most relevant external characteristics of polygonal shapes do so by evaluating the principal curvature of the surface. For polygonal meshes, however, this property is not defined. Thus, an approximation of κ_1 and κ_2 is used. Other approaches look at related properties such as normal variations on the mesh or use an associated internal skeleton.

In contrast to internal skeleton extraction methods that extract curve based skeletons, external feature line extraction approaches are better suited for compact objects. Because they capture local features rather than global properties, they are better able to locate the essential surface structures for shapes which are not piecewise tubular.

3.5 Summary

This chapter gave an overview of algorithms that extract geometric features from shapes (see Table 3.1). The algorithms were presented depending on what type of shape representation they can be applied to. First, three important mathematical concepts for shape representation and shape abstraction were presented—*Voronoi diagram*, *medial axis*, and *Reeb graph*. These serve as the basis of many algorithms mentioned afterwards. This was followed by a review of algorithms that can be applied to discrete data representations. In this context, both two-dimensional algorithms were discussed as well as methods for volumetric datasets. The remaining two sections covered techniques that were conceived for boundary representations. It was distinguished between algorithms that extract internal structures and those that locate features on the surface of shapes. This distinction can also be made for the techniques that are applied to discrete data. However, in that case, methods that extract internal structures are much more common.

In most cases, the discussed algorithms extract features from the shapes in form of graphs that consist of curves or edges. Only a few methods find structures in form of significant polygons or even connected surface sections. In addition, the algorithms differ in terms of whether they need user interaction or not and whether they yield an exact representation or only an approximate solution.

The presented selection of algorithms can only serve as an overview of the topic and is not meant to be a complete survey. However, the discussed algorithms show that the concept of what is being considered essential in a shape not only differs between different types of shapes or types of extracted features. Instead, even for the same shape and the same category of sought feature it is possible to produce different results. This is illustrated very well, for example, by the comparison in Figure 3.12. It is caused by varying requirements in different applications. In order to cope with this problem that for different notions of what is essential it is needed to use different feature extraction algorithms, a new concept is developed in Chapter 4.

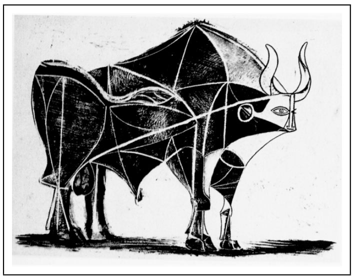
In particular, the internal and external structures extracted from boundary representations are important in the context of this dissertation (see summary in Table 3.2). It can be observed that internal features (skeletons) extracted from polygonal shapes do capture global properties of certain types of objects. However, these skeletons fail to represent certain local characteristics that are important as well. A detailed discussion of this topic will be given below in Section 5.1. On the other hand, the methods that find external structures on polygonal shapes extract local features but are restricted in terms of the criteria they are applying. The criteria cannot be changed among the algorithms and vice versa. This shows that a concept is necessary that allows for using different criteria with the same algorithm. Such a concept will be presented in Chapter 5.

algorithm	applied to 2D/3D	type of shape representation applied to	extracted features	extracted primitives
Voronoi diagram	2D	mathematical concept	internal	curves
	3D			surfaces
medial axis	2D	mathematical concept	internal	curves
	3D			surfaces
Reeb graph	2D/3D	mathematical concept	internal	graph
discrete skeleton extraction	2D	discrete data	internal	pixel
	3D		internal (and external)	voxel or graph
boundary representation feature extraction	3D	boundary representation	internal	graph or surface
			surface	graph or faces

Table 3.1: Summary of the main groups of feature extraction algorithms.

algorithm	internal/ external	result	interaction required	main features captured
[TT98]	internal	graph	yes	global
[BL99]	internal	graph	no	global
[WP00] etc.	internal	graph	no	global
[ABK98] etc.	internal	surface	no	global
[Raa98] etc.	internal	graph	no	global
[MWO03]	internal	graph	no	global
[VL97a]	internal	graph	yes	global
[RKS00]	external	graph	no	local
[WB01]	external	faces	no	local
[BO00]	external	faces	no	local
[HBK01] etc.	external	graph	no	local
[LL02]	external	graph	yes	local

Table 3.2: Summary of feature and skeleton extraction algorithms for boundary representations.



Chapter 4

Degree of Interest Functions

In the discussion of the essential shape representation concept in Section 2.5 it has been argued that in order to capture the essence of shape it is necessary to keep the specific ESR criterion separate from the specific ESR scheme. These ESR criteria define what it to be considered as important in a shape (also see Definition 2.3). Before introducing a new ESR scheme in Chapter 5, this chapter surveys a variety of shape indicators for their use as ESR criteria. In addition, new ESR criteria are explored partially based on this review. Other criteria are developed based on new concepts.

This chapter concentrates on ESR criteria for evaluating the importance of points on a polygonal surface of objects—Degree of Interest functions. In addition, only direct criteria will be considered because the algorithm presented in Chapter 5 is based upon scalar values computed for vertices on the surface. Most of the criteria discussed are based on geometric properties, in particular based on the approximation of the local curvatures in a point on the surface. However, some criteria are developed that are not based on geometric properties at all. These are used to demonstrate that the concept can also be used in applications other than feature detection.

4.1 Concept and Framework

In order to be able to evaluate previously discussed criteria for local shape classification and to develop new ones, the concept of the Degree of Interest function needs to be explained as well as the related concept of Extreme Ridges. For this purpose, Section 4.1.1 gives definitions and derives properties for these concepts. Afterwards, the restriction to polygonal surfaces is discussed in Section 4.1.2. In order to be able to derive specific criteria later on, the mathematical background of curvature approximation from polygonal meshes is given in Section 4.1.3. Finally, Section 4.1.4 discusses some basic DoI functions that can be derived directly from local curvature values.

4.1.1 Degree of Interest and Extreme Ridges

As has been mentioned above, the criteria discussed in this chapter are restricted to direct ESR criteria that are evaluated for surfaces of shapes. They will be defined as follows:

Definition 4.1 *A direct ESR criterion that captures the importance of points on a surface in form of scalar values will be called* Degree of Interest *(DoI). A specific importance value will be called* DoI value *and the function that computes the DoI value will be called* DoI function.

Formally, the DoI function shall be defined as follows:

Definition 4.2 *Consider a three-dimensional shape* $\Omega \subseteq \mathbb{R}^3$*, bounded by a surface* $S = \partial\Omega$*. The* Degree of Interest *function (*$\mathrm{DoI} : S \to \mathbb{R}$*) is defined such that it assigns a scalar value to every point of* S*.*

Various choices for DoI are possible; some examples will be given below. All DoI functions shall have the following properties:

1. Without loss of generality, the local maxima of the DoI function are zero-dimensional (i. e., they are isolated points on S).

2. The DoI possesses a network of *Extreme Ridges* (ER) (see Figure 4.1). An extreme ridge E is a smooth 1D path on S, such that in every point P of E, $\mathrm{DoI}(P) > \mathrm{DoI}(P + \varepsilon\, n_P)$ and $\mathrm{DoI}(P) > \mathrm{DoI}(P - \varepsilon\, n_P)$. Here, n_P is a unit vector perpendicular on E in the point P and ε is a sufficiently small positive real number (see Figure 4.2). Informally this means that an extreme ridge is a mountain-ridge of DoI. Formally it can be identified by interpreting the DoI function as a height profile and determining its second principal direction.[1] On both sides of the extreme ridge the values of the DoI function decrease. An extreme ridge can be *degenerate*, i. e., it can be of length zero. In a degenerate extreme ridge, n_P has arbitrary direction.

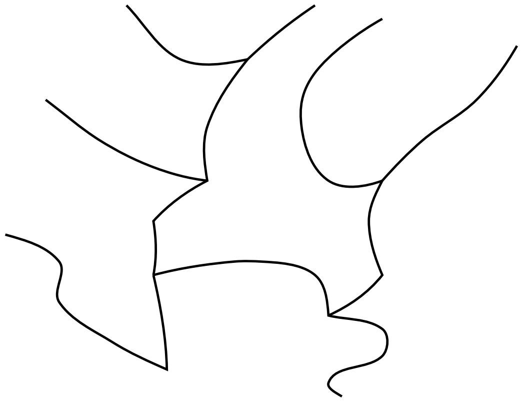


Figure 4.1: Schematic illustration of the network of extreme ridges.

Lemma 4.1 *A degenerate extreme ridge is a local maximum of the DoI function.*

3. An extreme ridge ends in two *Extreme Ridge Terminators* (ERTs) that are either one of the following:

---

[1] If local minima of the DoI function are used to define the extreme ridges the first principal direction has to be used.

]

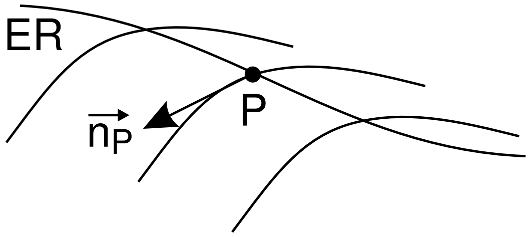


Figure 4.2: Schematic illustration of the definition of extreme ridges.

a) points where three or more extreme ridges are incident (these are the bifurcation points of the network of extreme ridges; some bifurcations may be local maxima of DoI; some local maxima of DoI may be bifurcations of extreme ridges),

b) points that are incident to precisely one extreme ridge (the *end point* or *starting point* of an extreme ridge), and

c) points where precisely two extreme ridges are incident: in these ERTs, the two incident extreme ridges have non-continuous tangents.

Lemma 4.2 *A local maximum of DoI is always on at least one (degenerate) extreme ridge.*

Lemma 4.3 *Every connected set of extreme ridges contains at least one local maximum of DoI.*

Belyaev, Bogaevski, and Kunii give a number of different definitions for ridges on smooth surfaces [BBK96]. They distinguish ridges by determining whether a certain curvature (in most cases the first principal curvature κ_1) attains a local maximum along a specific curve at a *non-umbilic* point.[2] These curves include, for example, the curvature line,[3] the normal section curve along the first principal direction, and the vertical section curve along the first principal direction. These definitions are related to the definition of an extreme ridge given here. However, since DoI functions do not inherently possess an associated direction in this case $\pm\varepsilon\, n_P$ is used to define the direction along which a local maxima is detected.

In [BAK97], Belyaev, Anoshkina, and Kunii define ridges as lines on the surface that correspond to singularities on the surface's caustic. The *caustic* (or *focal surface*) is the dual of a surface characterized by the loci of the centers of curvature of points on the surface. This definition is used in an approach to detect *salient curvature features* from polygonal meshes presented by Watanabe and Belyaev in [WB01] (see also Section 3.4.1).

---

[2] At an *umbilic* point, both principal curvatures κ_1 and κ_2 are equal to each other.

[3] A *curvature line* is the curve that follows a principal direction on the surface.

These notions of ridges[4] are well suited for defining certain geometry-based features on smooth surfaces. However, since these definitions all rely on the local curvature of the surface, they can only be used to find structures which are based on these properties. In addition, for all but the last mentioned approach the adaptation to polygonal surfaces does not follow easily from the definitions.

4.1.2 Discrete DoI Functions

So far, DoI functions were defined for arbitrary surfaces S. However, when working with polygonal meshes which have a piecewise linear surface MS, certain restrictions apply. Since polygonal models are typically only approximations of smooth surfaces with a limited number of samples it is not clear which specific smooth surface is approximated by a specific polygonal mesh. Hence, it is difficult to derive a continuous DoI function that reflects the ESR criterion on the approximated smooth surface. Therefore, the following definition is given:

Definition 4.3 *A* discrete DoI function *is defined as a DoI function on a polygonal mesh that only assigns DoI values to the primitives of a mesh: faces, edges, and vertices.*

In most cases and in the context of this dissertation, DoI functions will be evaluated for the mesh's vertices. Thus, unless otherwise noted, in the following a DoI function is considered to be a discrete DoI function evaluated for mesh vertices. This consequently also means that the network of extreme ridges are part of the polygonal mesh as well. Therefore, the extreme ridges consist of mesh edges and are, hence, piecewise linear (see Figure 4.3).

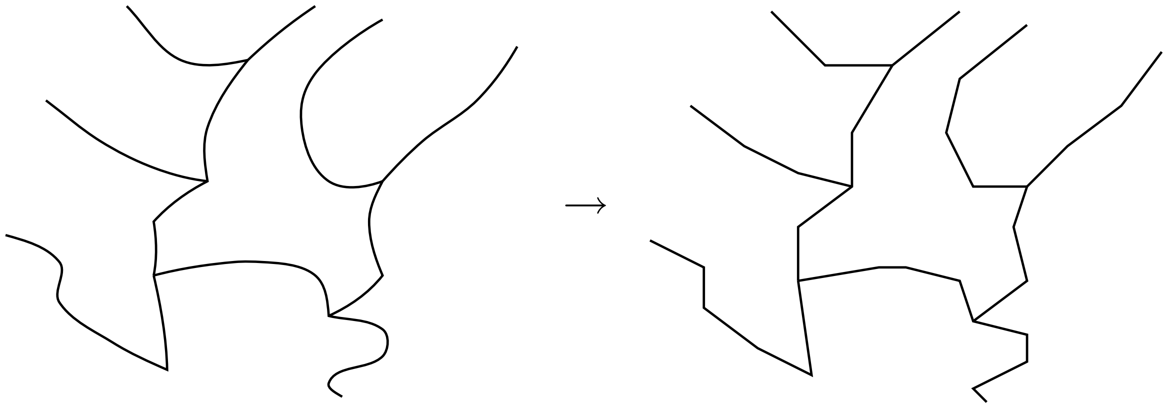


Figure 4.3: Instead of being a set of smooth curves, the network of extreme ridges for polygonal meshes consists of edges of the mesh.

---

[4] It should be noted that there is a branch of applied differential geometry called *ridge theory* that deals with various concepts of ridges and their applications. For a more thorough discussion of ridges and ridge theory see, e. g., [Ebe96].

Equipped with these definitions, specific discrete DoI functions can be formulated, both based on known shape criteria and newly developed functions. However, many previously presented algorithms for extracting geometric features from surfaces are based on principal curvatures and principal directions. Thus, in order to prepare for the discussion of some previously suggested criteria and also for newly developed DoI functions, some background will be given in the following section on the computation of these properties for polygonal meshes.

4.1.3 Determining the Discrete Curvature of Polygonal Meshes

Based on differential geometry, it is possible to compute the curvature for smooth surfaces assuming a sufficiently differentiable domain.[5] However, polygonal meshes do not have a smooth and sufficiently differentiable surface. Thus, the curvature is not defined for such surfaces (other than being infinity at edges and vertices where faces meet at an angle and zero everywhere else). Therefore, algorithms have to be employed for computing an approximation for the curvature on meshes, sometimes called *discrete curvature*. These methods will be discussed in the following. The section focuses on methods for estimating the discrete curvature in vertices.

As previously mentioned in Section 3.4.1, Rössl and Kobbelt developed a method for polygonal meshes MS to estimate the discrete principal curvatures κ_1 and κ_2 as well as their principal directions v_1 and v_2 [RK99, Rös99]. In this approximation, a biquadratic Taylor polynomial F is fitted through a vertex V and its neighbors (see Figure 4.4). This method will be explained in more detail below.

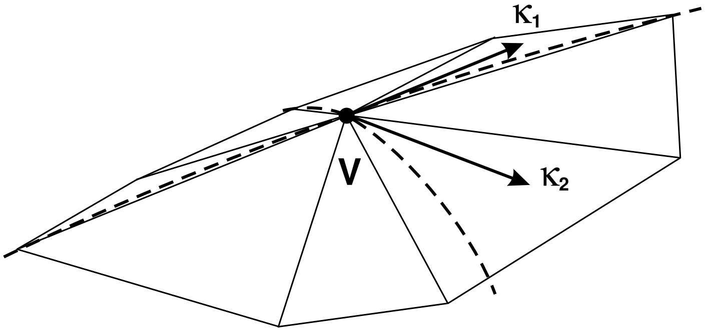


Figure 4.4: Estimating the principal curvatures using the method by Rössl and Kobbelt.

The first step is to transform the vertex and its neighborhood into a local coordinate system so that $F(0,0) = (0,0,0)$. Afterwards, a parameterization of the points to which the surface is going to be fitted has to be determined. Rössl and Kobbelt suggest to use one of two methods: Either, the neighborhood vertices can be projected onto the tangential plane or the parameterization is done using the exponential map (i. e., the angles between two adjacent edges originating from V are uniformly scaled

---

[5] For a detailed overview see, e. g., do Carmo [dC76].

until the sum of the angles equals 360°). The use of either one of the parameterization methods yields u_i and v_i for V's n direct neighbors: $F(u_i, v_i) = Q_i \quad (1 \leq i \leq n)$.

The second order surface is then locally approximated with a biquadratic Taylor polynomial using the partial derivatives in u and v as well as the second partial derivatives:

$$F(u,v) = uF_u + vF_v + \frac{u^2}{2}F_{uu} + uvF_{uv} + \frac{v^2}{2}F_{vv}\,. \tag{4.1}$$

There are n points for which the position on the surface $F(u_i, v_i) = Q_i$ is known: the n vertices which are direct neighbors of V. In addition, their respective parameters u_i and v_i have been determined during the parameterization step. Thus, a linear system consisting of n Taylor polynomial equations—one for each vertex and its parameterization—can be constructed as follows:

$$\mathbf{VF} = \mathbf{Q} \tag{4.2}$$

with $\mathbf{V} = (u_i, v_i, \frac{u_i^2}{2}, u_i v_i, \frac{v_i^2}{2})_i$, $\mathbf{F} = (F_u, F_v, F_{uu}, F_{uv}, F_{vv})^\top$, and $\mathbf{Q} = (Q_i)_i^\top$. RÖSSL and KOBBELT solve this system using a least squares respectively least norm approach yielding

$$\mathbf{F} = \begin{cases} \mathbf{V}^\top(\mathbf{V}\mathbf{V}^\top)^{-1}\mathbf{Q} & : \quad n < 5 \\ \mathbf{V}^{-1}\mathbf{Q} & : \quad n = 5 \\ (\mathbf{V}^\top\mathbf{V})^{-1}\mathbf{V}^\top\mathbf{Q} & : \quad n > 5 \end{cases}\,. \tag{4.3}$$

This procedure allows to approximate the partial derivatives F_u and F_v as well as the second partial derivatives F_{uu}, F_{uv}, and F_{vv} for the local neighborhood of V.

With these partial derivatives of the fitted surface approximated for V and based on the *first fundamental form* as well as the *second fundamental form* it is now possible to formulate the local curvature κ as a function of a direction λ with $\lambda = \frac{dv}{du} = \tan\alpha$:

$$\kappa(\lambda) = \frac{F_{uu}N + 2F_{uv}N\lambda + F_{vv}N\lambda^2}{F_uF_u + 2F_uF_v\lambda + F_vF_v\lambda^2}\,. \tag{4.4}$$

RÖSSL and KOBBELT show that the maximum and the minimum of the curvature in V—κ_1 and κ_2, respectively—correspond to the real solutions of the equation

$$det \begin{vmatrix} \lambda^2 & -\lambda & 1 \\ F_uF_u & F_uF_v & F_vF_v \\ F_{uu}N & F_{uv}N & F_{vv}N \end{vmatrix} = 0\,. \tag{4.5}$$

Thus, the two solutions $\lambda_{1,2} \in \mathbb{R}$ of the equation

$$\begin{aligned} 0 \;=\; & \lambda^2 \;\; (F_uF_v\,F_{vv}N - F_vF_v\,F_{uv}N) + \\ & \lambda \;\; (F_uF_u\,F_{vv}N - F_vF_v\,F_{uu}N) + \\ & 1 \;\; (F_uF_u\,F_{uv}N - F_uF_v\,F_{uu}N) \end{aligned} \tag{4.6}$$

have to be found. Using Equation 4.4 and the $\lambda_{1,2}$ derived from Equation 4.6 it is now possible to determine the principal curvatures κ_1 and κ_2. Alternatively, Rössl and Kobbelt show that it is also possible to compute the solutions for

$$det \begin{vmatrix} \kappa F_u F_u - F_{uu} N & \kappa F_u F_v - F_{uv} N \\ \kappa F_u F_v - F_{uv} N & \kappa F_v F_v - F_{vv} N \end{vmatrix} = 0\,. \tag{4.7}$$

However, it was found through experiments that the latter way was not as numerically stable as the former. In addition, using the former approach it is easier to derive the principal directions. Depending on the angle α which determines λ, the normalized direction vector $\overrightarrow{v(\alpha)}$ associated with the curvature $\kappa(\lambda)$ can be computed as follows:

$$\begin{aligned} \overrightarrow{v(\alpha)} &= \frac{\cos\alpha\, F_u + \sin\alpha\, F_v}{\|\cos\alpha\, F_u + \sin\alpha\, F_v\|} \\ &= \frac{F_u + \tan\alpha\, F_v}{\|F_u + \tan\alpha\, F_v\|}\,. \end{aligned} \tag{4.8}$$

Because $\lambda = \tan\alpha$, the normalized principal directions can now easily be found using $\lambda_{1,2}$ as determined above:

$$\overrightarrow{v_{1,2}} = \frac{F_u + \lambda_{1,2} F_v}{\|F_u + \lambda_{1,2} F_v\|}\,. \tag{4.9}$$

In a different approach, Goldfeather first surveys two methods which are—similar to the one proposed by Rössl and Kobbelt—based on a quadratic approximation of the unknown surface [Gol01].[6] He argues that these methods do not perform well enough for arbitrary meshes. Instead, Goldfeather suggests to also take into account the approximated normal vectors in V's neighbors and use a third order surface for approximation. Using an error analysis, he argues that the third order approximation performs significantly better in some cases.

However, for most triangle meshes with almost regular tessellation a quadratic approximation suffices. Thus, in the following the approach by Rössl and Kobbelt is used. On the other hand, any method that is capable of approximating the principal curvatures for polygonal meshes can be used for the algorithms described below.

4.1.4 Simple DoIs Based on Principal Curvature

Once the first and second principal curvature κ_1 and κ_2 have been determined, these values can be used directly to define DoI functions:

$$\mathrm{DoI}(V) = \kappa_1 \tag{4.10}$$

or

$$\mathrm{DoI}(V) = \kappa_2\,. \tag{4.11}$$

---

[6] See also references in Goldfeather's paper [Gol01] for a number of other approaches.

Instead of just using either maximal or minimal curvature as a degree of interest function, a combination of both values can be used. For example, *Gaussian curvature*

$$\mathrm{DoI}(V) = K = \kappa_1 \kappa_2 \tag{4.12}$$

and *average curvature*

$$\mathrm{DoI}(V) = H = \frac{1}{2}(\kappa_1 + \kappa_2) \tag{4.13}$$

are obvious choices for a combination of κ_1 and κ_2. They have previously been studied as a means for classifying the local shape of surfaces (e. g., by Rössl, Kobbelt, and Seidel [RKS00]). Therefore, they qualify for the use as DoI functions.

More complex DoIs can be defined that might be better suited for capturing the features that are looked for in certain applications. A selection of more sophisticated DoI functions—most of them based on principal curvatures—will be introduced and developed below.

4.2 Heuristic DoI Function

In order to compare it to the curvature based DoI functions later on, a simple heuristic DoI function based on an estimation for the average curvature at a vertex V will be given. This intuitive DoI function is computed using V's approximated normal vector $\overrightarrow{N_V}$ and all k neighboring faces (see Figure 4.5). First, the centroids C_j of all adjacent faces of the vertex are computed:

$$C_j = \sum_{i=1}^{n} \frac{V_{ji}}{n} \qquad j \in [1, k] \, . \tag{4.14}$$

Then, the average C_V of the centroids is computed as follows:

$$C_V = \sum_{j=1}^{k} \frac{C_j}{k} \, . \tag{4.15}$$

The normal of V is approximated based on the normals of its adjacent faces:

$$\overrightarrow{N_V} = \frac{\sum\limits_{j=1}^{k} \overrightarrow{N_j}}{\left\| \sum\limits_{j=1}^{k} \overrightarrow{N_j} \right\|} \, . \tag{4.16}$$

Afterwards, this point C_V is projected onto the normal $\overrightarrow{N}$ of the vertex V:

$$\overrightarrow{P} = \overrightarrow{N_V} \left(\overrightarrow{(V, C_V)} \cdot \overrightarrow{N_V} \right) \, . \tag{4.17}$$

The length of the projected vector $\overrightarrow{P}$ and its direction can be used as an indication for the curvature at this vertex:

$$\mathrm{DoI}(V) = -1(\overrightarrow{N_V} \cdot \overrightarrow{P}) \, . \tag{4.18}$$

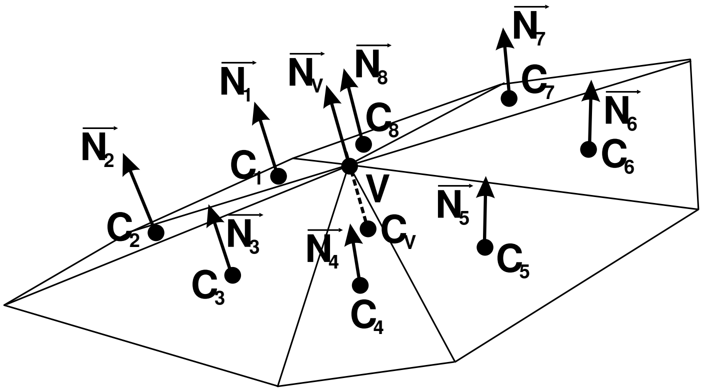


Figure 4.5: Deriving an heuristic measure for the curvature at a vertex of a polygonal mesh.

This curvature measure works well for models with regular triangles. Unfortunately, it does not perform very well when applied to meshes with irregular triangles.[7] In order to improve the method, the size of the faces A_{C_j} can be taken into account when computing the normal of V and the average of the centroids:

$$\overrightarrow{N_V} = \frac{\sum\limits_{j=1}^{k} \overrightarrow{N_j} A_{C_j}}{\left\| \sum\limits_{j=1}^{k} \overrightarrow{N_j} A_{C_j} \right\|}, \tag{4.19}$$

$$C_V = \frac{\sum\limits_{j=1}^{k} C_j A_{C_j}}{\sum\limits_{j=1}^{k} A_{C_j}}. \tag{4.20}$$

However, this intuitive approach relies on regularly tessellated input meshes where all faces have approximately the same size. Thus, it will not generate satisfying results for arbitrary meshes. For completeness, it should be noted that a number of second order differences based methods for estimating the curvature based on edges rather than vertices exist (e. g., [HMG00, HG01, LPRM02, LL02]). In these cases the angle between the normals of two adjacent faces is used to compute the DoI of an edge.

In the remainder of the chapter, new methods including, but not limited to some based on approximated principal curvatures will be discussed. Based on evaluation of a previously discussed measure for the local shape of a surface, a new measure is derived and generalized afterwards. In addition, a new measure based on singularities is introduced. Finally, DoI functions for use in applications other than feature detection are developed.

---

[7] Irregular triangles are very long and narrow.

4.3 Modified Shape Index

In the discussion of DoI functions so far a heuristic DoI function and simple combinations of κ_1 and κ_2 have been introduced. However, there are measures for classifying the local shape of surfaces that have previously been introduced. These can be examined for their eligibility as DoI functions. Alternatively, they can be used to derive a new DoI function which is done in the following.

4.3.1 Shape Index

Koenderink suggests using a so-called *shape index* in order to classify the shape in the neighborhood of a point on a surface. The shape index for a point is computed using its first and second principal curvature as follows [Koe90, KvD92]:

$$\mathrm{DoI}(V) = s = \frac{2}{\pi} \arctan\left(\frac{\kappa_2 + \kappa_1}{\kappa_2 - \kappa_1}\right) \qquad (\kappa_1 \geq \kappa_2)\,. \tag{4.21}$$

The shape index has values close to 1 for locally convex shaped surfaces, values close to -1 for locally concave shaped surfaces, and values close to 0 for locally saddle-like surfaces (see example shapes in Figures 4.6 and 4.7). The major advantage of using the shape index is that it can only have values in the interval $[-1, 1]$. However, its disadvantage is that it does not distinguish between strong and weak features. For example, a small bump has the same shape index value as a big one, given that they are both, for example, locally convex. In addition, in most cases one is not interested in the local convex or concave surface property in most cases. Those locally convex or concave areas are usually considered to be geometrically boring because this is where nothing “happens.” Instead, one would be interested in areas like faults.

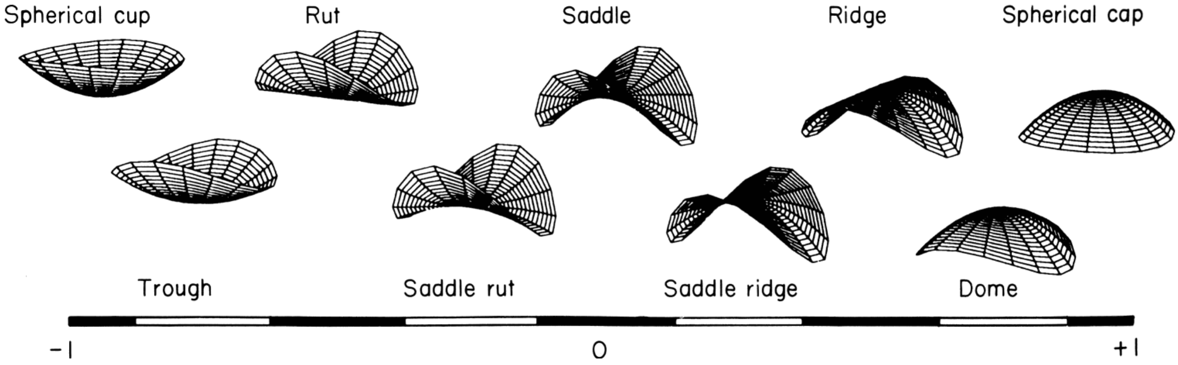


Figure 4.6: Shape index for certain shapes. From [KvD92].

4.3.2 Modifications to Emphasize Ridges and Ruts

In order to develop a DoI function which is better suited for identifying interesting features, the shape index will be modified. The goal in this modification is to empha-

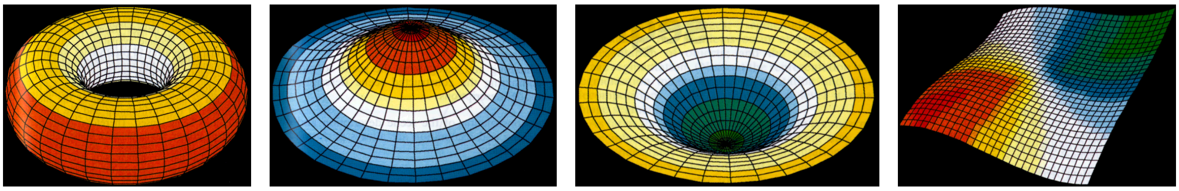


Figure 4.7: Color-coded shape index for some examples (green and blue represent negative while yellow and red represent positive values). From [KvD92].

size ridges and ruts as prominent geometric features of the surface. Nevertheless, the intensity of the feature has to be considered as well. Ideally, features would be sought for a specific footprint that is specified by a scale parameter. However, this is not possible using a purely local DoI function. Therefore, strong ridges and ruts (e. g., a deep narrow valley) will be favored over weak ones.

As KOENDERINK points out in [Koe90, page 322] and more specifically in [KvD92, page 560], the shape indices of ridges and ruts lie at 0.5 and −0.5, respectively (also see illustration in Figure 4.6). Since these are the features most people are interested in, they have to be emphasized while others should be de-emphasized. A gradual transition from maxima at $s_1 = 0.5$ or minima at $s_2 = -0.5$ to zero at $s \in \{-1, 0, 1\}$ can be achieved by various functions, one of which is the sine function. Hence, a *modified shape index* can be defined as follows:

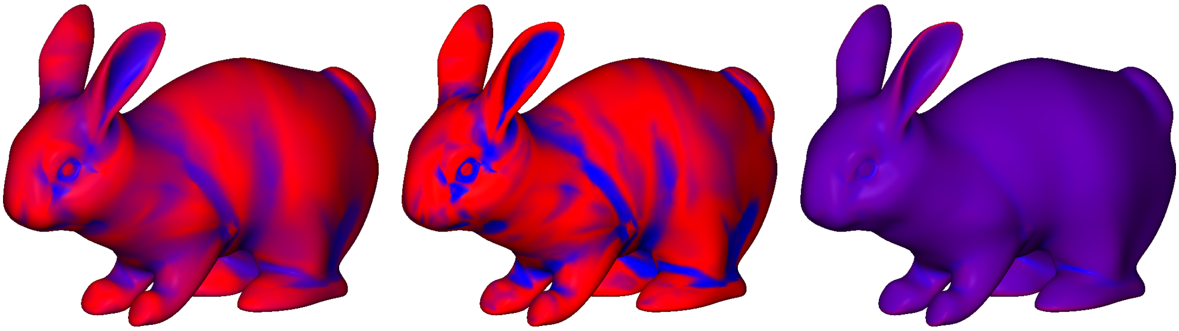
$$\begin{aligned} \mathrm{DoI}(V) &= \sin(\pi s) \\ &= \sin\left(2\arctan\left(\frac{\kappa_2+\kappa_1}{\kappa_2-\kappa_1}\right)\right) \qquad (\kappa_1 \geq \kappa_2) \\ &= \frac{2\left(\frac{\kappa_2+\kappa_1}{\kappa_2-\kappa_1}\right)}{1+\left(\frac{\kappa_2+\kappa_1}{\kappa_2-\kappa_1}\right)^2} \qquad (\kappa_1 \geq \kappa_2)\,. \end{aligned} \tag{4.22}$$

This formula of the modified shape index classifies the environment of a vertex V whether it is an interesting feature (ridge shaped or rut shaped) or if it has a rather uninteresting shape (such as convex, concave, or saddle-shaped). However, it does not make a distinction between strong or weak features. I. e., a point on wide ridge would be classified with the same DoI value as a point on a very sharp ridge.

Thus, the DoI values should be scaled by their significance, i. e., the distance of κ_1 and κ_2:

$$\mathrm{DoI}(V) = (\kappa_1 - \kappa_2)\frac{2\left(\frac{\kappa_2+\kappa_1}{\kappa_2-\kappa_1}\right)}{1+\left(\frac{\kappa_2+\kappa_1}{\kappa_2-\kappa_1}\right)^2} \qquad (\kappa_1 \geq \kappa_2)\,. \tag{4.23}$$

The effect of this function can be seen in Figure 4.8 where the three shape index based DoI functions are applied to an example shape. In Figure 4.8(a), the original shape index was used. Figure 4.8(b) shows the modified shape index which emphasizes ridges and ruts. However, it is clearly visible that no distinction is made as to the strength of a feature. For example, the ridge of the bunny's ear is attributed the same DoI values as the surrounding area since this is also, in principle, ridge shaped. However, when applying further scaling using the distance between both principal curvatures by applying Equation 4.23, this problem is overcome and the strong features are well distinguished from weaker ones (as shown in Figure 4.8(c)).



(a) Shape Index. (b) Modified shape index. (c) Scaled modified shape index.

Figure 4.8: Color-coded shape index based DoI values (red shows high values and blue low values). The DoI values have been normalized in order to fill the whole color interval; the DoI values in each image may be normalized differently.

An attempt to generalize the formula of the modified shape index will be taken in the following section. Similar to the scaled modified shape index, this more general approach also takes the degree of the feature into account.

4.4 Fold Index

As previously mentioned in Section 4.3.2, the desired measure should emphasize strong ridges and long, narrow valleys rather than treating every ideal ridge or rut equally. A surface point on one of these ideal features would have a negative first principal curvature (κ_1) along with a second principal curvature (κ_2) having the value zero (i. e., a point on a ridge) or κ_1 with the value of zero and positive κ_2 (i. e., a point on a rut). Thus, let

$$\mathcal{K} = \left(\frac{\kappa_2 + \kappa_1}{\kappa_2 - \kappa_1}\right) \qquad (\kappa_1 \geq \kappa_2)\,. \tag{4.24}$$

This term $\mathcal{K}$ classifies the shape in an environment around a point in that it equals 1 for environments of a point that are ideal ridges and -1 for ideally rut shaped surfaces. For a certain constant distance Δ_κ between κ_1 and κ_2, $\mathcal{K}$ grows for shrinking κ_i and shrinks for growing κ_i. For cases with negative κ_2 and positive κ_1 the value lies somewhere between 1 and -1. Hence, using Equation 4.24 Equation 4.22 can be simplified to

$$f_1(\mathcal{K}) = \frac{2\mathcal{K}}{1+\mathcal{K}^2}\,. \tag{4.25}$$

Assuming a general continuous function $f(\mathcal{K})$

1. that has its maximum value at $\mathcal{K} = 1$ and its minimum value at $\mathcal{K} = -1$ (for emphasizing ideal ridges and valleys as defined above),
2. that is monotone between $\mathcal{K} = -1$ and $\mathcal{K} = 1$, and
3. that is asymptotic to the $\mathcal{K}$-axis for values above $\mathcal{K} = 1$ and below $\mathcal{K} = -1$,

this function returns the desired interest value (using Equation 4.24) since it yields the DoI value of 1 for ideal ridges, the DoI value of -1 for ideal ruts, and DoI values between -1 and 1 for all other shapes:

$$\mathrm{DoI}(V) = f(\mathcal{K})\,. \tag{4.26}$$

One can easily see that the function $f_1(\mathcal{K})$ in Equation 4.25 which is used for the modified shape index has exactly the properties of $f(\mathcal{K})$ as described above (also see the plot of $f_1(\mathcal{K})$ in Figure 4.9).

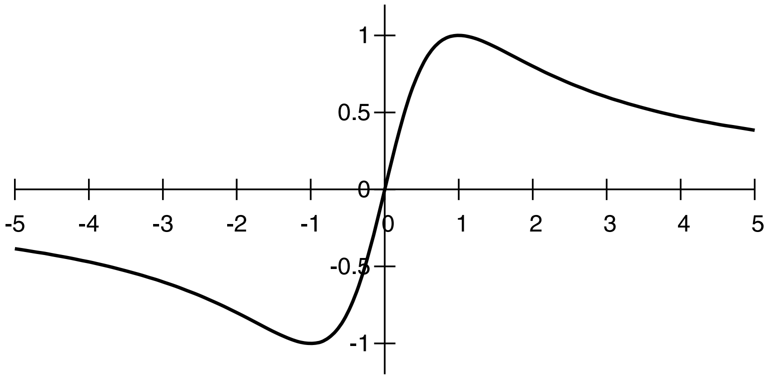


Figure 4.9: Plot of function $f_1(\mathcal{K})$ in Equation 4.25.

As pointed out in Section 4.3.2, a desired trait for the DoI function is that the sharper a ridge is the more important it should be. As previously done with the modified shape index, this can be achieved by multiplying the size Δ_κ of the interval $[\kappa_1, \kappa_2]$:

$$\begin{aligned}\mathrm{DoI}(V) &= (\kappa_1 - \kappa_2)\, f\,(\mathcal{K}) \\ &= (\kappa_1 - \kappa_2)\, f\left(\frac{\kappa_2 + \kappa_1}{\kappa_2 - \kappa_1}\right) \qquad (\kappa_1 \geq \kappa_2)\,. \end{aligned} \tag{4.27}$$

This formula for computing a DoI value for a vertex is called *fold index*. It generalizes the formula used in the scaled modified shape index (Equation 4.23).

In fact, any $f(\mathcal{K})$ that fulfills the requirements mentioned above can be employed in computing the fold index. Besides $f_1(\mathcal{K})$ of Equation 4.25, the following function based on the e-function can be employed as an appropriate $f(\mathcal{K})$ since it has the desired characteristics (also see plot of $f_2(\mathcal{K})$ from Equation 4.28 in Figure 4.10):

$$f_2(\mathcal{K}) = \mathcal{K}\, e^{(1-|\mathcal{K}|)}\,. \tag{4.28}$$

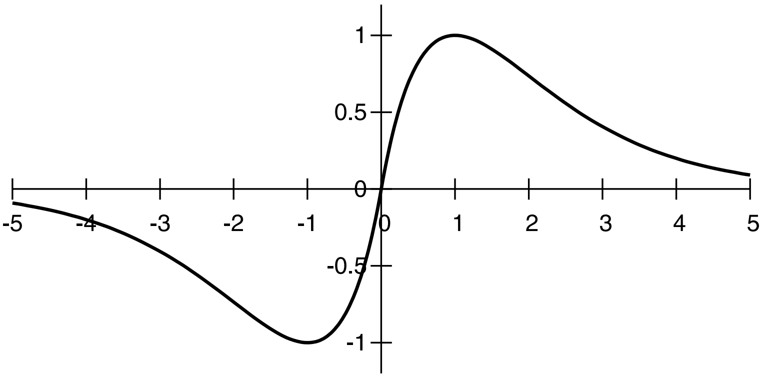
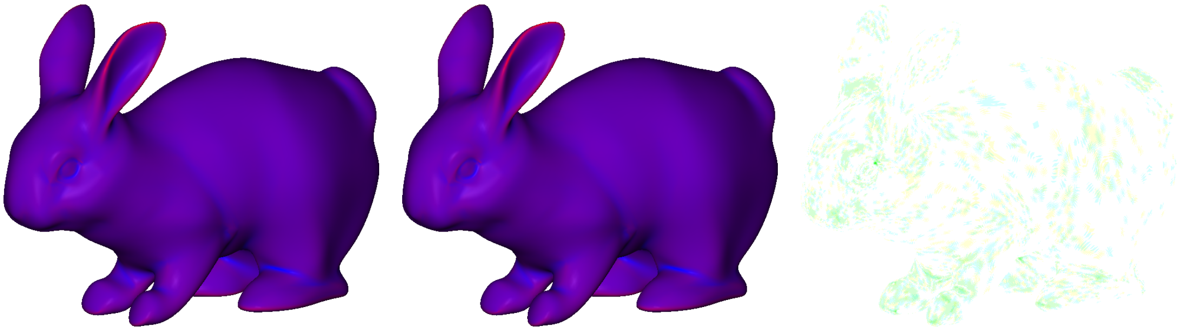


Figure 4.10: Plot of the function $f_2(\mathcal{K})$ in Equation 4.28.

Combining Equations 4.27 and 4.28 yields the following formula for computing the importance of a vertex V based on the interpolated principal curvatures κ_1 and κ_2:

$$\mathrm{DoI}(V) = -(\kappa_1 + \kappa_2)\, e^{\left(1-\left|\frac{\kappa_2+\kappa_1}{\kappa_2-\kappa_1}\right|\right)} \qquad (\kappa_1 \geq \kappa_2)\,. \tag{4.29}$$

As shown in Figure 4.11, the two fold indices (using function $f_1(\mathcal{K})$ or $f_2(\mathcal{K})$) only differ slightly. Although the function $f(\mathcal{K})$ used in both cases differs in the specific characteristics, both DoI functions are almost equally well suited for identifying ridges or ruts on a shape according to the specifications in Section 4.3.2 as shown in Figure 4.11.



(a) Fold Index using $f_1(\mathcal{K})$ (scaled modified shape index).

(b) Fold Index using $f_2(\mathcal{K})$.

(c) Amplified difference image.

Figure 4.11: Color-coded fold indices and the difference image. Although visually both fold indices cannot be distinguished, there are slight differences. However, the differences are less than 2.4% on each of the three 256 index RGB color scales.

The advantage of the fold index over the regular shape index is that it is independent from the local convexity or concavity of the surface. Instead, it emphasizes faults or ridges. Moreover, its value is not dependent on the actual size and shape of the triangles (as the heuristic measure is) because it depends only on the approximated principal curvatures. In addition, in contrast to the modified shape index, it favors strong features over weak ones since the fold index also takes the intensity into account. I. e., a vertex on a strong fault will yield a higher DoI value than a point on a weaker fault while both will have similar DoI values when employing the modified shape index. One disadvantage, however, for both the modified shape index and the fold index is that they do not emphasize single peaks.

4.5 Singularity Index

The (modified) shape index and the fold index only use κ_1 and κ_2 in a single vertex V in order to evaluate DoI(V). One can also take an environment around V into account. Thus, besides considering analytic measures based on the primary curvatures, a different measure was considered that is based on the singularity properties of κ_1 and κ_2 for a mesh vertex. For illustration of what is meant by singularity property the following metaphor is proposed. Consider a window through which one can see a part of the outside world (see Table 4.1). Now if the window is moved along its main coordinate axes the contents of the window may or may not change. If the contents changes regardless of which axis was chosen to move it along the contents can be considered to have the singularity degree of 0. If, instead, the window contents only changes when moving it along one of its two axes a feature of singularity degree 1 is detected. In case the contents does not change at all a feature of degree 2 has been found. In general, a feature or singularity with the degree of n is characterized by n coordinate axes of a coordinate system along which the window can be moved without the contents to change.

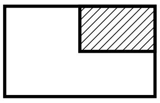
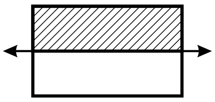
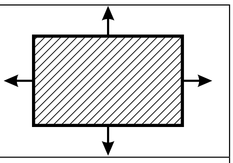
window metaphor			
singularity	0	1	2
$\Delta\kappa_{1,2}(V)$	$\Delta\kappa_{1,2} \geq t$	$\Delta\kappa_i \geq t$; $\Delta\kappa_j < t$	$\Delta\kappa_{1,2} < t$
primitive	point	line	surface

Table 4.1: Deriving a singularity property based on the principal curvature values and directions estimated in the vertices of a polygonal mesh.

This metaphor is applied to the principal curvatures on polygonal meshes. The “window” which is looked through is characterized by the two principal curvatures and the “window contents” by their values. The coordinate axes are defined by the perpendicular principal directions. In order to “move the window”—meaning to compute the $\Delta\kappa_{1,2}(V)$—it is necessary to interpolate the values of the principal curvatures at points other than the vertices of the mesh. The value of κ_i' is estimated for the slightly moved point P by applying SHEPARD's interpolation scheme [She68].[8] It uses the $\kappa_i(V_j)$ for the vertex V and all its n adjacent vertices $V_1 \dots V_n$ and their locations p and $p_0 \dots p_n$ (see Figure 4.12):

$$\kappa_i' = \frac{\sum_{j=1}^{n} \frac{\kappa_{ij}}{|p-p_j|^2}}{\sum_{j=1}^{n} \frac{1}{|p-p_j|^2}} \quad i \in \{1,2\} \,. \tag{4.30}$$

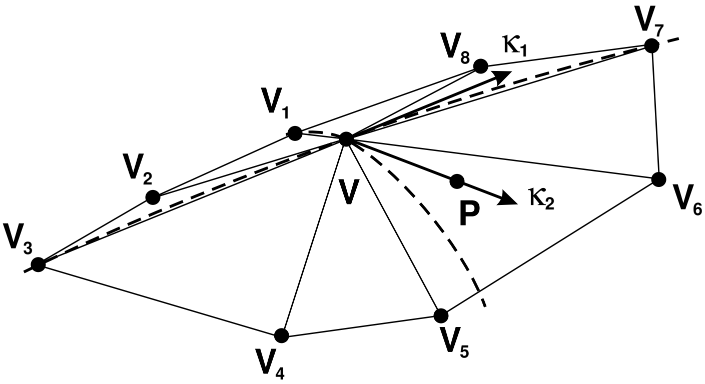


Figure 4.12: Estimation of the principal curvatures at a point other than a mesh vertex by using SHEPARD's interpolation scheme [She68].

The final computation of the singularity degree of a vertex is only based on two user-defined values: the distance between the actual vertex and the point where the curvature values are estimated (how far to "move the window") and a threshold t that determines the smallest change in curvature values that is considered to be significant. The former—the moving distance—is usually assigned the value of half the shortest length of any edge in the mesh.

The singularity degrees for every mesh vertex can now in turn be used to define a new interest measure which is called the *singularity index.* For that it is first postulated that every vertex with a singularity degree of 0 or 1 is considered to be an interesting feature and vertices with degree 2 to be not interesting. Thus, the smallest possible value for the threshold t for which the singularity degree of a vertex V changes from uninteresting (2) to interesting (1 or 0)—the maximum of the variations of κ_1 and κ_2—is an indicator for the degree of interest of that vertex. This leads directly to the definition for the singularity index as

$$\mathrm{DoI}(V) = \max(\Delta\kappa_1(V), \Delta\kappa_2(V)) \,. \tag{4.31}$$

Two examples for the application of the singularity index are shown in Figure 4.13.

---

[8] Other interpolation schemes may be used as well.

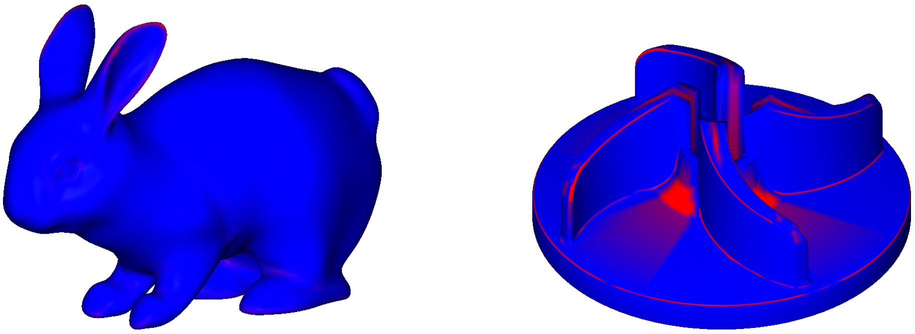


Figure 4.13: Color-coded examples for the application of the singularity index (red shows high values and blue low values).

4.6 DoI Functions for Adaptive Subdivision

Although the DoI function will mainly be used for detecting the significance of shape in the sense of feature extraction, finding important parts of shapes might also be valuable in other applications. However, in application domains other than geometric feature extraction it might be necessary to use DoI functions which do not emphasize ridges or ruts. Thus, in this section a number of other DoI functions are developed in order to illustrate that the DoI concept is not limited to feature detection. The DoI functions introduced in this section were conceived for application in adaptive Loop subdivision (for details see [IHK03]). In this context and in contrast to the previously discussed functions, the DoI values in this section are not only computed for vertices but for faces and edges as well.

The common goal of most subdivision algorithms is to generate meshes that permit visually pleasing renderings of the models. Therefore, most adaptive subdivision techniques use geometric properties such as flatness—measured by the angle between adjacent faces—to identify polygons where further subdivision is needed. There are, however, various applications that have other requirements as to where fine tessellation is needed. Hence, different criteria have to be used to decide where to apply additional subdivision steps. There are even applications for adaptive subdivision where the criterion for refinement is not entirely geometric. Thus, a more general method that uses DoI functions for the adaptive refinement decision are needed in these cases. The subdivision scheme remains the same while the applied DoI function may vary within or across applications. The examples in this section were generated using Loop subdivision, an approximating scheme commonly used for triangle meshes [Loo87]. In principle, however, this method is applicable to any subdivision scheme.

Before every subdivision step, the application computes the DoI function for each vertex, edge, or face on the triangle mesh. These DoI values are now used instead of geometric properties to decide if additional subdivision steps are required or not by comparing them to a certain threshold value. If the DoI is bigger than the threshold t the face is subdivided, otherwise it is not.

4.6.1 DoI Functions for Silhouette Generation

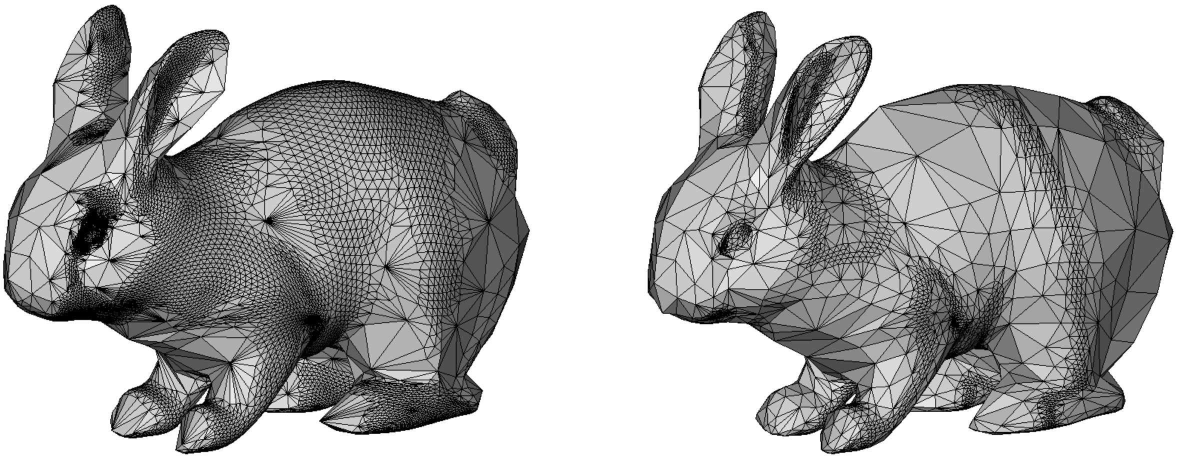
The main goal when using adaptive meshes for silhouette generation is to reduce the number of triangles while retaining a similar quality if the generated rendition. Azuma et al. showed how to generate view dependent models using the progressive meshes algorithm [ACD+01, AWC+03]. These meshes are finely tessellated in areas tangential to the viewing direction and sparsely tessellated perpendicular to it. This means that silhouettes extracted from the model will appear as smooth lines with fewer triangulation artifacts. To achieve similar meshes with the DoI based adaptive subdivision method the DoI functions are based on the angle between face normal and viewing direction. This is achieved by computing the dot product of both vectors

$$DoI(F) = \overrightarrow{n} \cdot \overrightarrow{v} \tag{4.32}$$

(see Figure 4.14(a)). A method that produces even less faces for the same number of adaptive subdivision steps considers edges where front facing and back facing triangles meet. Those silhouette edges are assigned a DoI value of 1 while all other edges are assigned a DoI value of 0

$$DoI(E) = \begin{cases} 1 & : \quad [(\overrightarrow{n_1} \cdot \overrightarrow{v} \geq 0) \wedge \ (\overrightarrow{n_2} \cdot \overrightarrow{v} < 0)] \vee \\ & \quad\; [(\overrightarrow{n_1} \cdot \overrightarrow{v} \leq 0) \wedge \ (\overrightarrow{n_2} \cdot \overrightarrow{v} > 0)] \\ 0 & : \quad otherwise \end{cases} \tag{4.33}$$

with n_i being the normals of the two faces. This results in a binary DoI function. Hence, the threshold t to which the DoI function is being compared can be set to anywhere in the interval $(0;1)$. The result of using this DoI function is shown in Figure 4.14(b).



(a) Faces with an angle of less than 20 degrees to the viewing direction are subdivided.

(b) Edges where front facing and back facing triangles meet are subdivided.

Figure 4.14: Dynamic adaptive subdivision (three steps) for better silhouette generation. In both cases the model is viewed directly from the front.

Meshes generated with adaptive subdivision using these DoI functions have to be updated in every frame since they are view dependent. For generating static meshes which better support silhouette generation one can use the observation that the probability for an edge to be a silhouette edge depends on the angle between both its adjacent faces. In addition, concave edges cannot become silhouette edges at all. Thus, the DoI in this case can be derived directly from the angle between the two faces

$$DoI(E) = \arccos(\overrightarrow{n_{f_1}} \cdot \overrightarrow{n_{f_2}}) \tag{4.34}$$

(see Figure 4.15). If the edge is concave it could additionally be set to zero because these edges are never silhouette edges. The result of this approach is similar to the one presented in [AFR03].

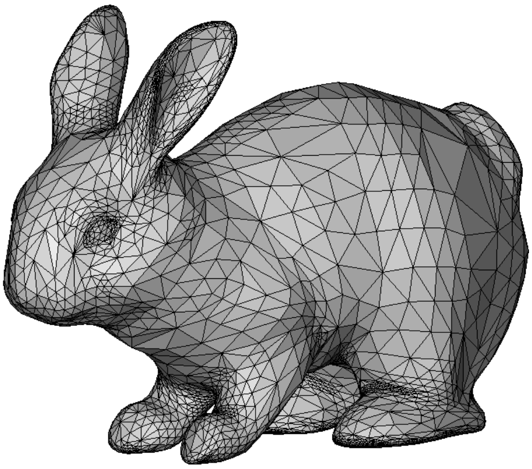


Figure 4.15: Static adaptive subdivision for better silhouette generation: adjacent faces with an angle higher than 30 degrees are subdivided.

4.6.2 DoI Functions for Improving Haptic Rendering

A second example incorporates adaptive subdivision into a haptic rendering algorithm. The major disadvantage of polygonal meshes in haptic rendering is that users perceive discontinuities during the exploration of objects. This problem is usually overcome by *force shading*. This technique interpolates the direction of the reaction force to reduce the discontinuities along edges. Unfortunately, this causes lateral forces that can significantly disturb users during the exploration and can influence their perception.

To overcome this problem, there are basically two approaches. The first works directly on the original triangular mesh and uses the LOOP subdivision algorithm for a dynamic, successive refinement in the neighborhood of the surface contact point (SCP) of the haptic interaction device. The limited time to determine the SCP prohibits the application of adaptive subdivision within the intersection calculation. Hence, meshes within different subdivision levels have to be pre-calculated and stored (see [RBR01]).

As opposed to the approach just discussed, a second solution is based on a piecewise interpolation of the displayed surface using Bézier patches. For a smooth mesh it is necessary to have a smooth transition between adjacent patches. However, in the underlying triangular geometry at some places two adjacent faces might connect with a too sharp angle causing perceivable artifacts in the haptic rendering. A DoI function is used to identify these places on the surface and cause the mesh to be adaptively subdivided. This results in meshes which are usable for Bézier patch based haptic rendering (see example in Figure 4.16).[9]

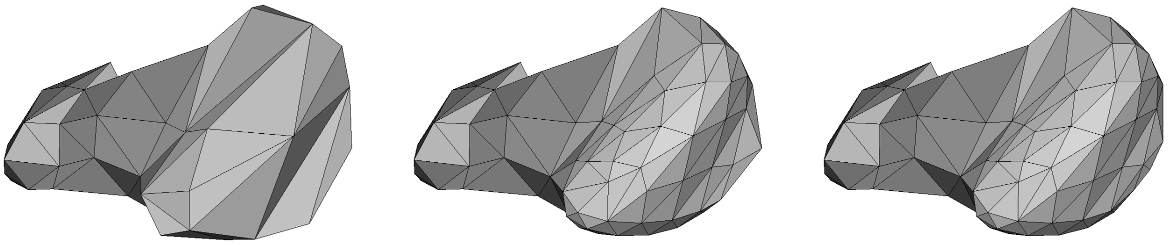

(a) No subdivision steps. (b) One subdivision step. (c) Two subdivision steps.

Figure 4.16: Triangles that produce perceivable discontinuities in haptic rendering are being subdivided.

4.6.3 DoI Functions Based on User Interaction

Priority values not based on geometric properties are used in a number of interactive illustration systems (e. g., ZoomIllustrator [PRS97], TextIllustrator [SS00], and AGILE [HS02]). The DoI functions or dominance values used in these systems are extracted by analyzing user interactions or visible text segments in order to guarantee a coherent multi-modal presentation.

An analysis of scientific illustrations reveals that human artists display details within the local environment of the most dominant objects. Moreover, these details fade out with increasing distance to the dominant object. Hence, to enhance such interactive illustration applications, the DoI values provided by the systems can directly be used for adaptive subdivision. This makes sure that the most dominant structures are enhanced through the finer tessellation whereas irrelevant geometric parts or objects remain coarse (see example in Figure 4.17).

The presented DoI functions for adaptive subdivision are examples for using the degree of interest concept in application domains other than feature extraction. In these cases it is not essential to find geometrically interesting regions of a shape. Instead, the DoI functions classify regions where additional subdivision is necessary. It could be demonstrated that even DoI functions that are not based on geometric properties at all can be used in this context.

---

[9] For a detailed description on how to compute the DoI function for haptic rendering see [IHK03].

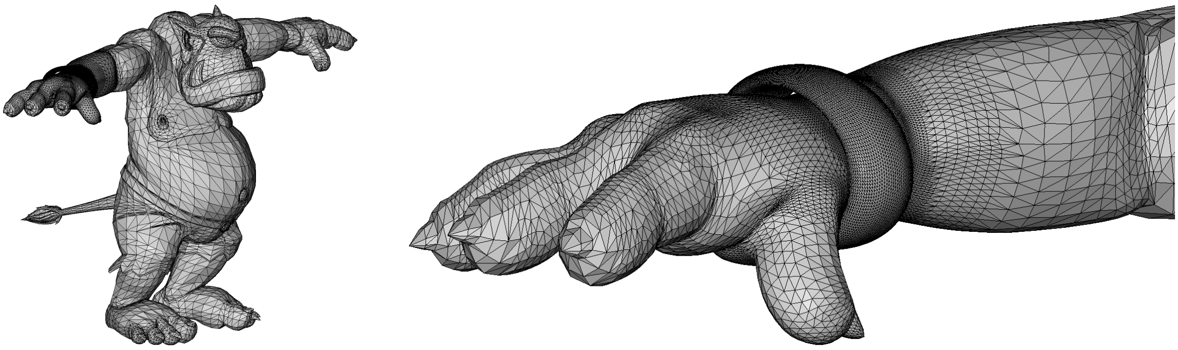


Figure 4.17: Olaf's bracelet is the object of focus. Thus, its triangles and all triangles of objects close to it are subdivided. The closer the triangles are to the focus point the more subdivision steps are used.

4.7 Summary

For a summary of the DoI functions discussed in this chapter, in Section 4.7.1 they are first classified according to a number of criteria such as what properties they are based on or where they can be primarily applied. In addition, in Section 4.7.2 the geometric DoI functions for capturing the essential parts of a shape will be compared. In order to allow for evaluating their behavior for example shapes color-coding of the computed DoI values is used.

4.7.1 Classification and Hierarchy of DoI Functions

The DoI functions that were introduced in this chapter are direct, surface detail ESR criteria according to the classification in Section 2.6. For the purpose of formulating a hierarchy of the different DoI functions, they are categorized according to whether they are based on geometric properties or whether they are computed non-geometrically. The geometric DoI functions in turn can be split into functions that are based on principle curvatures such as shape index, modified shape index, fold index, and singularity index. On the other hand, there are geometric functions that are not based on κ_1 and κ_2 such as the heuristic DoI function introduced in Section 4.2, the silhouette property based DoI function, and the DoI function that takes into account errors in haptic rendering.

Besides being based on geometric properties, DoI functions can also be computed entirely differently. For example, one could think of a farmer who would classify different locations on the farmland by the fertility or the amount of income it produces. This could be called the *farmland index* which is not based on geometric shape properties. In addition, the DoI values based on user interaction are based on non-geometric properties as well. In fact, any criterion that could lead to a pseudo segmentation[10] or

---

[10] A pseudo segmentation is a segmentation where not all segments need to have closed borders.

partition of S can be employed. Such non-geometric DoI functions can be used just as well to compute an external skeleton. Hence, a taxonomy of the DoI functions described in this chapter could look as depicted in Figure 4.18.

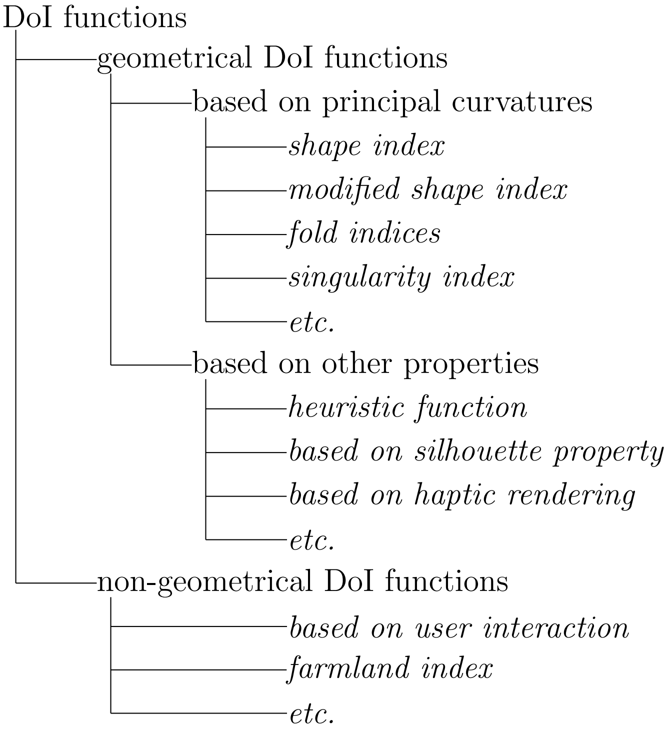

DoI functions
geometrical DoI functions
based on principal curvatures
shape index
modified shape index
fold indices
singularity index
etc.
based on other properties
heuristic function
based on silhouette property
based on haptic rendering
etc.
non-geometrical DoI functions
based on user interaction
farmland index
etc.

Figure 4.18: Proposed taxonomy for the DoI functions developed and discussed in this chapter.

Besides building a hierarchy based on the type of property that is evaluated in order the compute DoI values, the DoI functions can also be classified according to whether they are view-dependent (such as two of the DoI functions based on silhouette property of the vertices) or view-independent (as the rest). In addition, the DoI functions can be grouped into those which are suited for extracting some sort of feature from the model's surface (such as shape index, modified shape index, fold index, singularity index, intuitive measure, and farmland index) and the rest which is used, e. g., for adaptive subdivision.

4.7.2 Comparison of Geometric DoI Functions

A number of different DoI functions have been conceived and/or examined in this chapter. Some of them can be used to identify significant parts of objects. In particular, the heuristic measure, the simple principle curvature based functions, the shape index, the modified shape index, and the fold indices can be used for this purpose.

In order to be able to compare them (see summary in Table 4.2), these functions have been applied to example shapes as shown in Figures 4.19 and 4.20. In the figure, the DoI values have been color-coded to visualize their value. In each individual image, the interval of DoI values computed have been linearly scaled to fit the red-blue scale. High values are presented in red while low values are shown in blue.

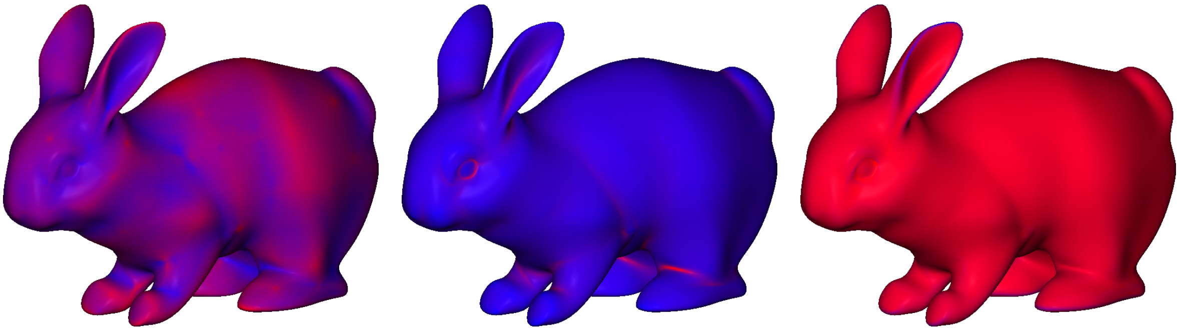
DoI function	robust to tessellation	feature strength considered	ridges and ruts well distinguished
Heuristic measure	no	yes	no
Maximal curvature	yes	yes	n/a
Minimal curvature	yes	yes	n/a
Maximal absolute curvature	yes	yes	no
Minimal absolute curvature	yes	yes	no
Average curvature	yes	yes	yes
Gaussian curvature	yes	yes	no
Shape index	yes	no	yes
Modified shape index	yes	no	yes
Fold index using $f_1(\mathcal{K})$	yes	yes	yes
Fold index using $f_2(\mathcal{K})$	yes	yes	yes
Singularity index	no	yes	no

Table 4.2: Comparison of the properties of geometric DoI functions.

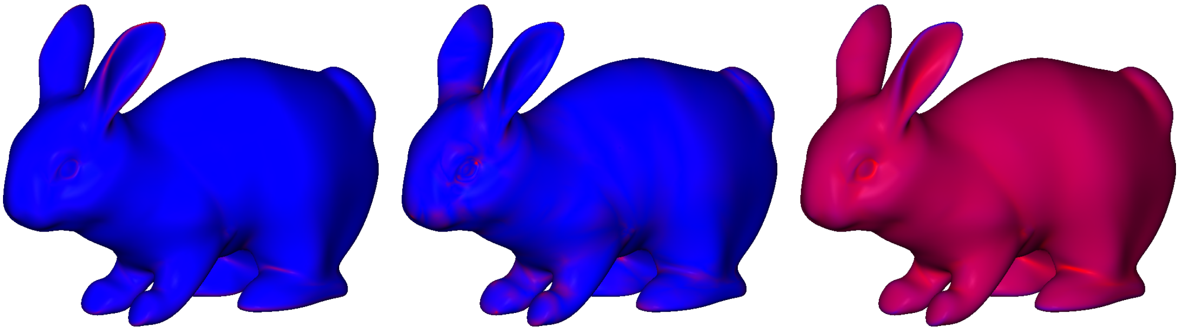
As can be seen in the figures, most results of the various DoI functions differ significantly. For example, it can be observed that the heuristic measure is somewhat dependent on the triangulation (see Figures 4.19(a) and 4.20(a)). It can also be seen that the shape index and the modified shape index do extract the local shape using their individual scales but do so regardless how strong the features are (see Figures 4.19(h), 4.20(h), 4.19(i) and 4.20(i)). Finally, it should be noted that it seems that the fold indices nicely extract ridges and ruts as extreme values (red and blue parts) while non-significant parts are assigned values right in the middle (purple parts, i. e., the DoI values are zero in this case) between both extremes (see Figures 4.19(j), 4.20(j), 4.19(k), and 4.20(k)).

Thus, depending on the application at hand either one of the DoI functions introduced can be used to evaluate the significance of points on the surface. Hence, it is up to the user to decide which measure does best fits his or her needs. In addition, multiple DoI can be used in separate runs of the algorithms while the individual results are combined (see example in Section 5.3.3).

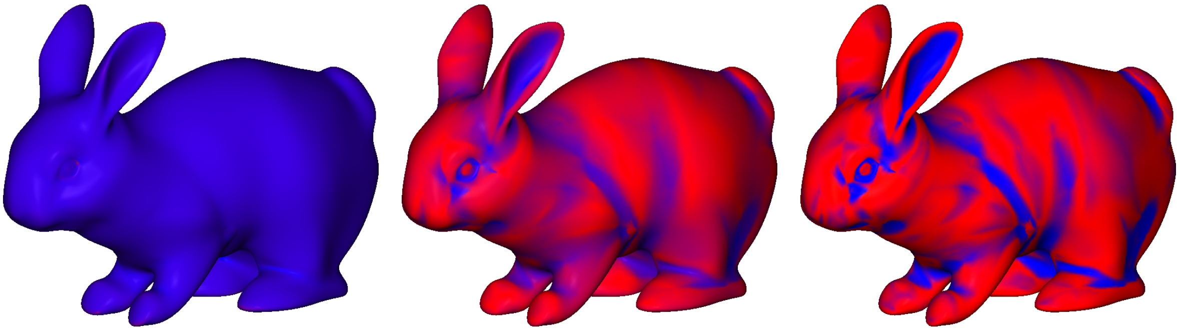
Besides the DoI functions introduced in this section many more are possible. The selection presented here is intended on the one hand to illustrate the concept of and show examples for possible DoI functions. On the other hand the specific functions presented will be applied in the following sections for finding significant parts of a polygonal mesh.



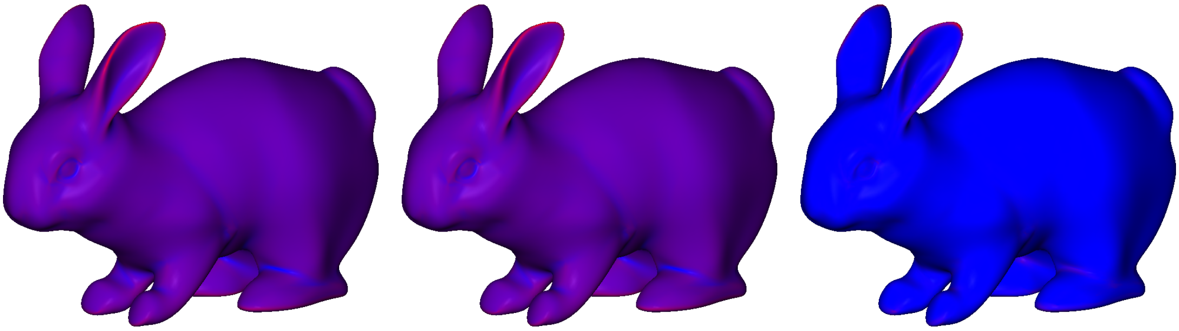
(a) Heuristic measure. (b) Maximal curvature. (c) Minimal curvature.



(d) Maximal absolute curvature. (e) Minimal absolute curvature. (f) Average curvature.

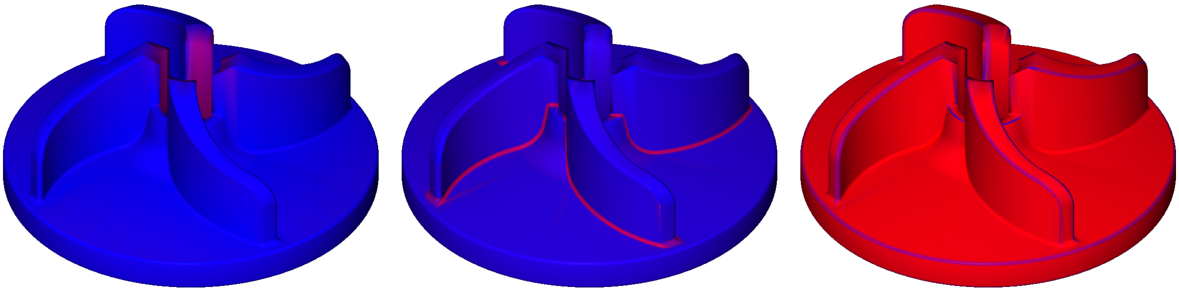


(g) Gaussian curvature. (h) Shape index. (i) Modified shape index.

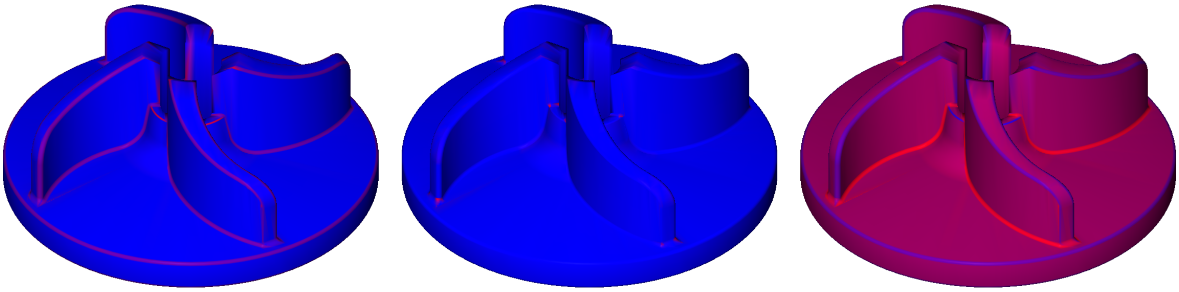


(j) Fold index using $f_1(\mathcal{K})$. (k) Fold index using $f_2(\mathcal{K})$. (l) Singularity index.

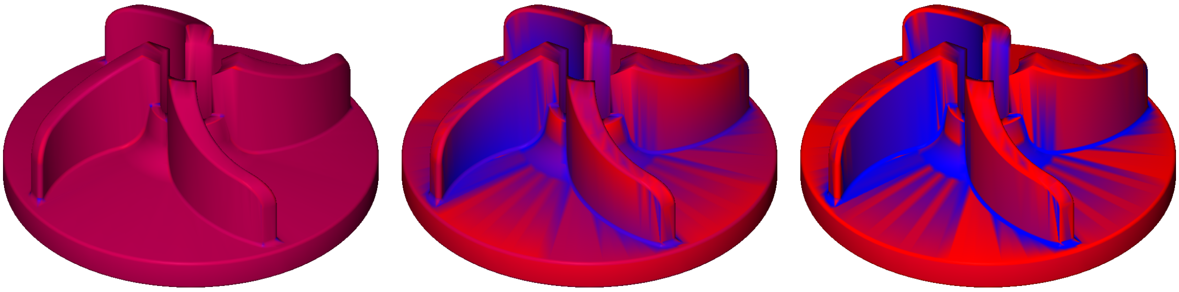
Figure 4.19: Color-coded DoI values using some of the DoI functions introduced in this chapter (red shows high values and blue low values). The DoI values have been normalized in order to fill the whole color interval; the DoI values in each image may be normalized differently.



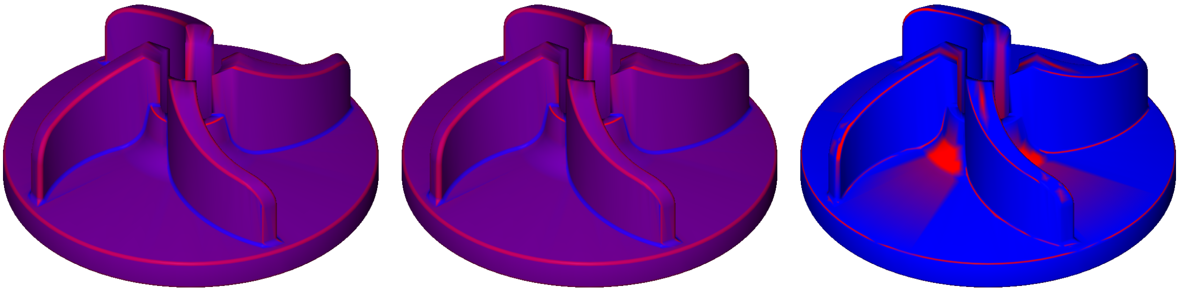
(a) Heuristic measure. (b) Maximal curvature. (c) Minimal curvature.



(d) Maximal absolute curvature. (e) Minimal absolute curvature. (f) Average curvature.

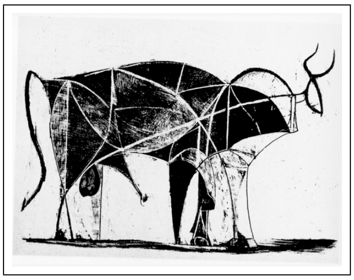


(g) Gaussian curvature. (h) Shape index. (i) Modified shape index.



(j) Fold index using $f_1(\mathcal{K})$. (k) Fold index using $f_2(\mathcal{K})$. (l) Singularity index.

Figure 4.20: Color-coded DoI functions as used in Figure 4.19 applied to a technical model.



Chapter 5

An ESR in 3D: the External Skeleton

In the previous chapter, a concept for specifying what is considered to be important for a specific application was introduced—DoI functions. Based upon these, a new ESR scheme is developed that allows detection of the important features of the surface of an object. This scheme utilizes the property of the DoI function that it defines extreme ridges on the object's surface as noted in Section 4.1.1. These curves are identified using a wave propagation technique that traverses the surface of the object.

Before going into detail about the new essential shape representation algorithm, the need for this new concept will be argued. This discussion is based on a review of previous internal ESRs. Afterwards, the wave propagation based algorithm is developed and examined in detail. In the following section a number of algorithmic considerations and potential modifications of the technique are presented. Finally, the method is analyzed and evaluated in particular in terms of how it supports the representation of scale, the steps necessary in order to reconstruct the shape, and robustness as well as potential problems.

5.1 Analysis of Internal Skeletons

A selection of previously presented algorithms for extracting internal skeletons from polygonal mesh based geometry models has been introduced and discussed in Section 3.3. In the following section, these algorithms are going to be reviewed in terms of the requirements for essential shape representations mentioned in Section 2.3. First, the appropriateness of the algorithms for certain types of models is examined and how well the extracted skeletons resemble the form of shapes. In addition, the sensitivity of the algorithms to the tessellation of the models and errors in the meshes is described. Finally, it will be shown how the algorithms deal with segmented meshes. Based on this discussion the necessity of a new concept is concluded.

5.1.1 Suitability for Different Classes of Models

In general, it can be observed that edge or line based internal skeletons of three-dimensional shapes are well suited for representing tubular shapes. This especially applies to tubular shapes that have an approximately circular cross-section. In these cases, the skeleton can easily be used for creating a reconstruction close to the original shape (Requirement 1 in Section 2.3) by moving the circular cross-section along the skeleton path oriented perpendicular to the skeleton. The reconstructed surface is derived by connecting the circumferences of the cross-sections. If the skeleton additionally stores the local average distance of the shape to the skeleton this can be used to scale the cross-section accordingly. In addition, the original shape does not necessarily have to resemble one single tube. It is sufficient if the shape's parts are tubular. Then, the generated skeleton will comprise of one skeleton curve per tubular part (see example in Figure 5.1).

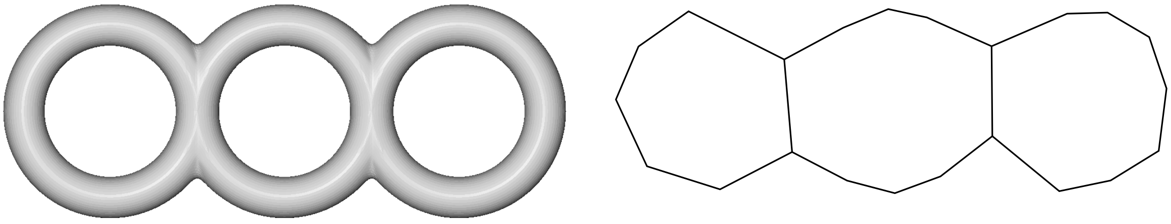

(a) Example shape consisting of tubular parts. (b) Derived line based skeleton.

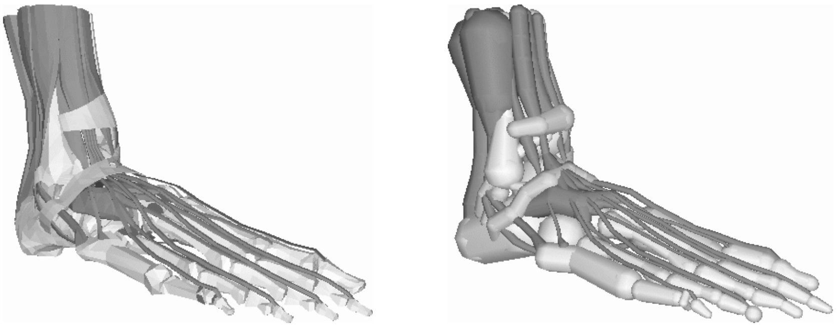
Figure 5.1: Deriving a line based skeleton from shapes which are tubular or consist of tubular parts.

However, not all shapes are tubular. Many have a more compact or flat nature. In these cases, the essence of their shape lies not in the way tubular parts are assembled to form the final object but rather in what the characteristics of the surface are. Such shapes can only be poorly approximated with a line based internal skeleton. A correct or almost correct reconstruction is hardly possible from theses skeleton graphs (see Figure 5.2). Thus, line based internal skeleton extraction is not as well suited for this class of shapes.

5.1.2 Correlation of the Skeleton Graph with the Original Model

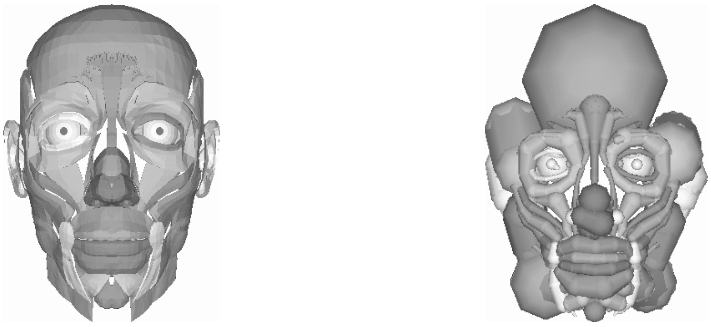
A potential problem with skeletons generated with particularly the edge collapse based algorithms is that the skeleton edges are not necessarily located exactly in the middle of the original model part as previously discussed in Section 3.3. In fact, in certain cases it can happen that a skeleton edge can even be partly outside of the object part it is approximating (see example in Figure 5.3 and 5.4). However, in some application domains it is necessary that the skeleton graph is located centrally to its associated structure. For example, in medical visualization it is essential that the skeleton lies exactly in the center of the shape to be applicable in vessel analysis and operation planning (see, e. g., [Sel99, HSEP00, SPSP02]).

The other algorithms introduced in Section 3.3 yield better skeletons in terms of the location of the skeleton edge central to the shape. However, most of them cannot guarantee the precise central location of the skeletons, either. For example, Figure 3.19 on page 35 shows that Wade and Parent's algorithm [Wad00] yields skeletons which



(a) Shape with tubular objects.

(b) Reconstruction of the tubular objects.



(c) Shape with compact objects.

(d) Reconstruction of the compact objects.

Figure 5.2: Reconstructing shapes from an internal edge based skeleton using the quadratic error. As shown in (a) and (b), the reconstruction of tubular objects works well. However, the reconstruction of compact objects does not correspond to the original as well ((c) and (d)). From [Raa98].

are not central to the object (see the neck of the horse). Also, the skeleton extraction algorithm for point clouds by Verroust and Lazarus [VL97a] cannot guarantee central skeletons (see Figure 3.27 on page 41).

Another observation can be made for all skeleton extraction algorithms that yield edge or curve based skeleton graphs. None of these algorithms computes a mathematically correct skeleton (see Sections 3.1 and 3.1.1). This is inherent in using lines or curves as the primitive of the skeleton rather than planar primitives. Naturally, the fact that the skeleton curves are not necessarily central to the shape also contributes to the non-mathematical character of the skeletons.

5.1.3 Sensitivity to the Tessellation of Shapes

In edge collapse based skeleton extraction algorithms the position of generated skeleton edges and even the decomposition of the skeleton graph into skeleton edges is highly dependent on the order in which the edges are selected for being collapsed. Thus, it

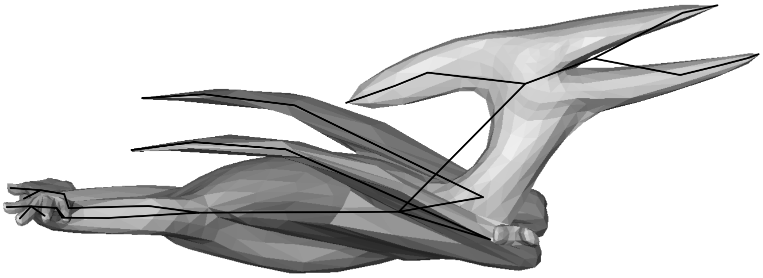
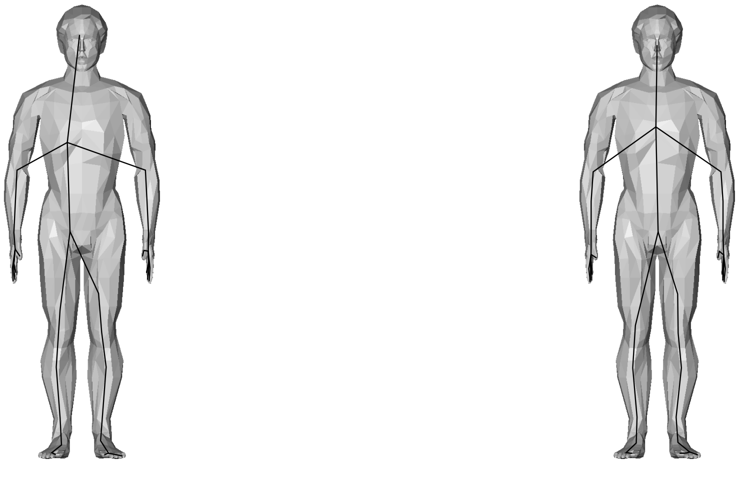


Figure 5.3: Example for a skeleton created with the edge collapse method where parts are outside of the object. In particular, see the neck and the beak of the dinosaur.

is dependent on the tessellation of the object rather than its overall shape. In fact, skeleton vertices might "drift" while performing the skeleton extraction.

Stabilization of the drift behavior can be achieved by averaging the final position of skeleton vertices by considering the original vertices that make up the skeleton vertex. This reduces the dependency on the order of edge collapse steps. RAAB argues that a weighted sum of the positions to minimize the quadratic error should be used [Raa98]. However, since minimizing the quadratic error has not been solved efficiently, he suggests using homogeneous coordinates for the purpose of finding the central point for a set of points. When determining the point in the middle between the vertices of the edge that is to be collapsed, the two positions just have to be added using homogeneous coordinates. Since adding homogeneous coordinates is an associative operation the sum of the positions of a set of points yields the position in the middle. Figure 5.4 displays an example for the stabilization of the drift behavior. In Figure 5.4(a) it is clearly visible that although the original shape is symmetric the skeleton is not. Figure 5.4(b) shows the almost perfectly symmetrical skeleton after stabilization.

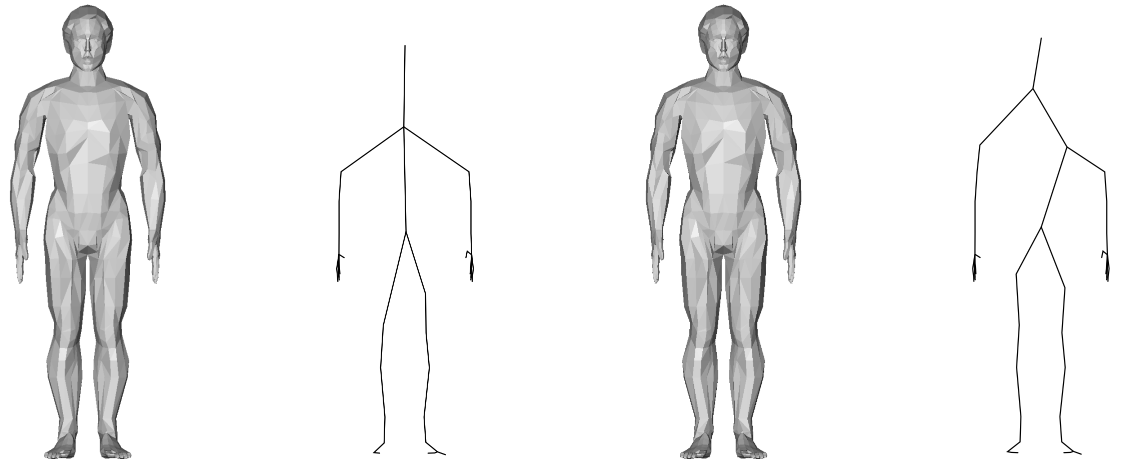


(a) Shape and not stabilized skeleton.

(b) Shape and stabilized skeleton.

Figure 5.4: Stabilization of the drift behavior which is caused by a certain succession of edge collapse steps by averaging the final positions of the skeleton vertices.

Unfortunately, one important property of edge collapse based algorithms—that the decomposition of the skeleton graph into skeleton edges is dependent on the sequence of edge collapse operations—cannot be removed by this modification. In fact, the sequence of edge collapse steps is highly dependent on the specific triangulation (topology of the triangulation and positions of the vertices) of the mesh. Only small changes in the triangulation can lead to significant and global changes in the skeleton graph (see example in Figure 5.5).



(a) Shape one. (b) Skeleton of (a). (c) Shape two. (d) Skeleton of (c).

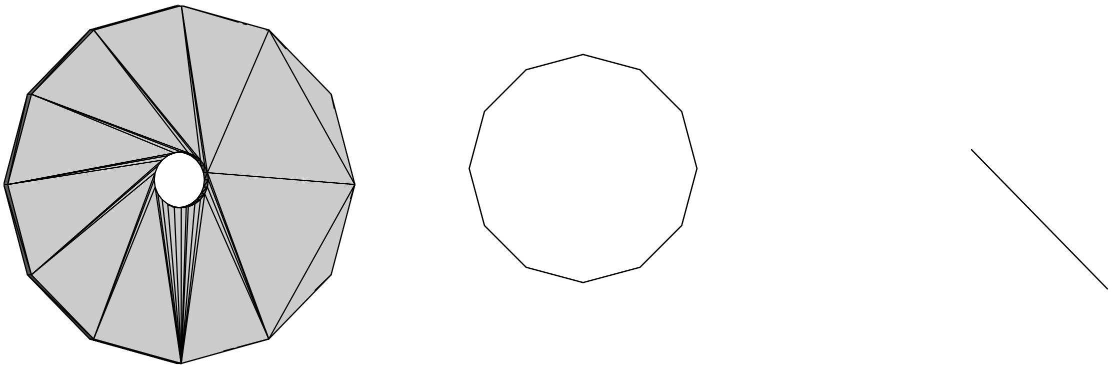
Figure 5.5: Sensitivity of the edge collapse based internal skeleton to slight changes in the triangulation of the shapes they are computed for. Being visually equivalent, the two shapes (a) and (c) only differ in the scales of their underlying geometric models.

5.1.4 Sensitivity to Errors in the Mesh

In some cases geometric models contain errors that are due to the process of how they are constructed or obtained. Errors that occur most often are normals with wrong orientation, double edges or double faces as errors in the tessellation or even missing faces that cause holes.

The edge collapse based skeleton extraction algorithm is robust against models with errors in the normals, e. g., it is not necessary to have a model with correct classification of “inside” and “outside”. In addition, errors in the triangulation can be handled very well, e. g., holes or double edges do not cause problems for the algorithm. However, it has to be noted that the algorithm may alter the topological genus of the model during the process. It may happen that holes in the original model are closed (see example in Figure 5.6). Nevertheless, since no new holes can appear the topological genus can never increase but only decrease.

Similarly, the point cloud based skeleton extraction technique by Verroust and Lazarus [VL97a] is robust against these errors since triangulation errors or normal orientation do not apply to point clouds. The medial axis based skeleton extraction



(a) Object: disk with hole.

(b) Expected skeleton in form of a closed loop.

(c) Resulting skeleton: only one edge is created.

Figure 5.6: Problem of edge collapse based skeleton extraction with the genus of the mesh. Since the hole of the disk has short edges compared to the outer edges they are being collapsed first. Thus, instead of the expected skeleton ring a single edge is the output of the algorithm.

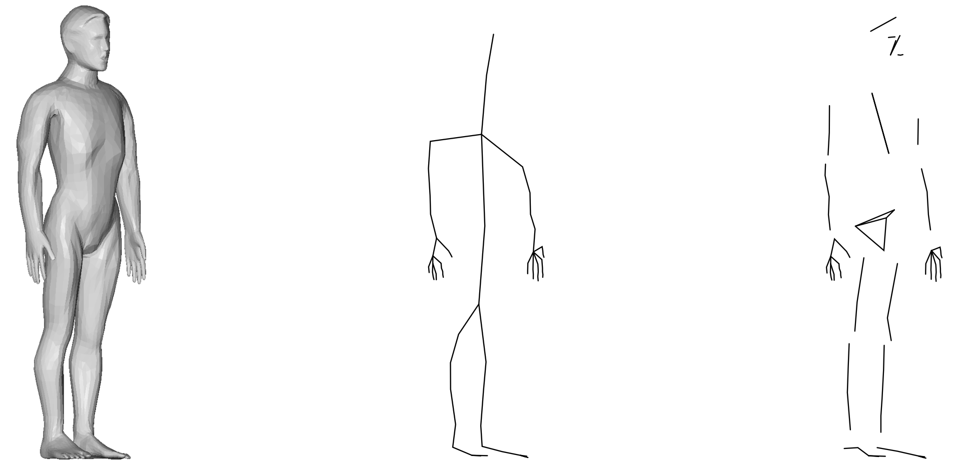
algorithms should generally also be robust against wrong normals or double edges. However, holes will most likely cause additional skeleton edges to be computed. As an exception, the method by WADE and PARENT [Wad00] needs correct normals and closed meshes in order to derive a correct voxel representation which is needed to find the discrete medial surface from which the skeleton is computed.

5.1.5 Skeletons of Segmented Meshes

Another problem for skeleton extraction algorithms is posed by segmented models. In 3D model libraries (e. g., the VIEWPOINT catalog [Vie99]), models that are intended particularly for character animation are sometimes available in segmented form instead of as a single skin. Although visually they seem to be connected, the different parts of the model are logically separated from each other. This segmentation is intended to support the efficient animation of the characters using underlying bone hierarchies.

Some skeleton extraction algorithms can deal with this type of models without problems (e. g., WADE and PARENT's algorithm [Wad00] or the point cloud based algorithm by VERROUST and LAZARUS [VL97a]). However, the result of applying certain other skeleton extraction algorithms to one of these models is a logically and almost certainly also visually separate skeleton graph for each object of the segmented model as shown in an example in Figure 5.7.

On the other hand, it is sometimes required to extract a connected skeleton graph from a model. As already discussed in Section 3.3.2, one option for obtaining such a connected skeleton is to use an adapted skeleton extraction algorithm (e. g., using GARLAND and HECKBERT's edge collapse algorithm [GH97]). A different way for



(a) Traditional rendition of the segmented model.

(b) Expected skeleton.

(c) Resulting skeleton due to segmentation of the model.

Figure 5.7: Skeletons extracted from segmented models (a) are segmented as well (c). For obtaining a connected skeleton graph as shown in (b) the patches have to be welded together prior to starting the edge collapse process.

obtaining a connected skeleton is to have a model with connected objects or patches to start with, i. e., the model must contain just one connected object.

For shapes that originally comprised one connected mesh and that were only segmented for animation this is not difficult. The separated patches still share vertices at the same location. Hence, the vertices that share a location can be determined (e. g., using a hash based approach) and welded together. In the example above, the patches that form the shape in Figure 5.7(a) were merged prior to generating the skeleton in Figure 5.7(b).

Solving the problem for models for which it is not possible to use the procedure outlined above is also possible. For example, in addition to segmented or not segmented *single-skin* models in character animation, there are also so-called *ball-jointed* (sometimes simply called *jointed*) forms of models [Tra94]. In this form of models, the parts of the character (e. g., the forearm and the upper arm) are independent from each other and the joints are modeled as half spheres for each of the body parts. Thus, although being logically independent, the parts overlap at the joints. This allows easy hierarchical animation with bones because the individual parts just have to be assigned to the associated bone and transformed accordingly. This means that no simulation of bending limbs is necessary. Thus, although being not as visually pleasing due to seams at the joints and rigid body parts, this type of animation is easier than for single-skinned models.

When computing internal skeletons for this type of models, segmented skeletons are generated because a simple welding of the individual parts is not possible. The generation of a single-skinned version of these shapes for skeleton extraction can be achieved, for example, by employing α-*shapes* [EM94]. A similar procedure can also be applied to avoid problems with models that have small holes that ought to be filled.

5.1.6 Necessity for a New Concept

Table 5.1 summarizes the results of the analysis of internal skeletons. From this analysis it can be concluded that they do have some serious flaws according to the requirements named in Section 2.3. Most notably, internal skeleton extraction methods are not suitable for all types of shapes in terms of the form of their surface (Section 5.1.1). In particular, compact shapes can only poorly be represented by internal, line or curve based skeletons. In addition, being internal ESR schemes, internal skeletons capture the overall form of the shape. They do not preserve certain, possibly important surface features. However, in some applications (e. g., shape storage and reconstruction) it is not sufficient to capture the general form of the whole shape. Hence, the form of the surface has to be included in an essential shape representation as well.

This shows the necessity of a new concept of external skeletons. In the remainder of this chapter, such a concept will be developed and properly defined. In addition, an algorithm will be described which extracts external skeletons using DoI functions that have been discussed in Chapter 4 as the significance criterion.

5.2 External Skeletonization using Wave Propagation

The review of existing algorithms for extraction of internal skeletons that was given in the previous section showed the necessity for a new concept for capturing the essence of shape. In order to avoid the problems of internal skeletons, the following sections introduce the new concept of external skeletons. Based on a wavefront propagation algorithm and using DoI functions as discussed in Chapter 4, skeleton edges are extracted in form of extreme ridges as a subset of the polygonal mesh.

5.2.1 Definition and Algorithm Outline

Based on the definitions in the context of DoI functions in Section 4.1.1, the notion of *external skeletons* can now be formally defined as follows:

Definition 5.1 *An* external skeleton *(ES) is the network of extreme ridges on a surface S. The extreme ridge terminators (ERTs) together with the local maxima of DoI that are not ERTs are the nodes in this network. The external skeleton is dependent on the specific DoI that is used to determine the extreme ridges.*

Notice that the network of extreme ridges need not be connected. In general, it consists of multiple connected components.

Thus far, extreme ridges have been defined so that they can be determined for arbitrary surfaces S. In the remainder of the chapter, however, only polygonal meshes MS will be considered (similar and relating to the discussion of DoI functions in Chapter 4). Thus, the network of extreme ridge on these meshes consists a number of separate parts with each part being a connected set of edges.

Criterion	medial surface or Voronoi algorithms (e. g., [TT98])	voxel based method (e. g., [Wad00])	edge collapse algorithm ([Raa98])	RBF method ([MWO03])	point cloud algorithm ([VL97a])
Suitable for long tubular shapes	yes				
Suitable for compact shapes	no				
Location of the skeleton with respect to the object	almost centered		in most cases not centered	almost centered	not always centered
User interaction required	yes	no	no	no	yes
Sensitivity to tessellation	high; medial axis property	low	high; drift behavior	low	n/a
Sensitivity to wrong normals	none	none	none	high	n/a
Sensitivity to holes	high	high	no problem when small	high	n/a
Preservation of the genus	yes	not guaranteed	not guaranteed	yes	n/a
Skeletons created from segmented models	segmented	sometimes segmented	segmented (with special modifications not segmented)	segmented	n/a
	modifications for creating connected graphs exist (e. g., welding vertices or using alpha shapes [EM94])				

Table 5.1: Summary of analysis of internal skeletons.

In order to establish the network of extreme ridges, the first step is to search (all) local maxima of the selected DoI function. These local maxima will form the set LM. The rationale is that, according to Lemma 4.3, LM yields at least one point in each of the connected components of ES. In order to establish the entire network of extreme ridges, all of S has to be surveyed. To do this a strategy based on propagating waves is employed (see the following section). This process starts in each of the points in LM and follows local maxima of the chosen DoI function.

The process of determining wavefront edges on the mesh and finding the external skeleton is divided into the following four steps:

1. locate the local maxima of the chosen DoI function on the 2D mesh surface (Section 5.2.3),
2. create the wavefronts originating from those local maxima (Section 5.2.4),
3. regularize the determined set of wavefront edges (Section 5.2.5), and
4. follow the local maxima of the chosen DoI function (Section 5.2.6).

Before these steps will be explained in more detail in the following, a short discussion of the rationale is given that allows using wave propagation in an algorithm for extracting extreme ridge curves.

5.2.2 Rationale of Wave Propagation

Wave propagation is a physical process that occurs when oscillations spread in a continuous medium [Fle80, page 150]. It can occur in different dimensional media (e. g., one-dimensional waves on a spiral spring, two-dimensionally moving waves on a water surface, and three-dimensional sound waves in the air). For a medium with constant velocity of wave movement, the wavefront is perpendicular to the direction of wave movement [GKV89, page 150]. In such a case a single point wave initiator would cause two-dimensional wavefronts to form concentric circles on a plane.

The propagation of wavefronts over the mesh is used as a means to traverse the mesh in an organized way. This, in turn, allows to trace the extreme ridge curves starting at the wavefront initiators (local maxima) from which they emerge and follow them over the mesh.

The process of wavefront propagation starts with the identification of local maxima of the DoI function as wave initiators. Features in form of local maxima curves emerge from such wave initiators. These features (i. e., extreme ridges or external skeleton curves) are to be detected but are not known at this point. Thus, wavefronts are initiated by the isolated local maxima and are propagated across the mesh. It can be supposed that the features are initially parallel to the wave movement if the waves are approximately circular. This approximately circular form of the wavefronts can only be achieved by a constant or nearly constant velocity of wave propagation [GKV89] causing the wavefronts to be perpendicular to the extreme ridges. This means for the mesh's tessellation that it should be balanced in order to allow for a near-geodesic metric. For unbalanced tessellations the velocity is not constant and thus the features are not perpendicular to the wavefronts. The property that the extreme ridges are perpendicular to the wavefronts only holds for wavefronts close to the initiators because with growing distance of the wavefronts from their initiators (age of the wavefronts) the extreme ridge curves may turn. Thus, in order to minimize the growing errors with growing distance from the local maxima, wavefronts are generated emerging from all local maxima. This also means that the extreme ridge curves that are traced by different wavefronts are expected to meet where the wavefronts meet.

The wavefronts are not only used for traversing the mesh but are also used to identify the extreme ridges (see Section 4.1.1). The ERs have initially been defined as local maxima of the DoI function along a direction $\pm\varepsilon\, n_P$. This can now be replaced by the wavefront since it is approximately perpendicular to the ER as outlined above.

The technique of wave propagation is based on HUYGENS' principle. It was formulated by the Dutch physicist Christiaan HUYGENS in his book "Traité de la Lumière" in the late 17th century [Huy90]. The principle says that the propagation of a wave can be understood by regarding every point at a wavefront as a new wave source. This means that the individual waves initiated by all points on the wavefront interfere with each other and, consequently, their union forms the new collective wavefront. This principle will later be used in the wave propagation algorithm when performing the actual propagation.

5.2.3 Finding Local Maxima

As discussed in Section 5.2.1, the first part in the process of finding the external skeleton ES is to determine where the wave propagation is supposed to start on the polygonal mesh. This is done by finding all vertices V representing local maxima of the chosen DoI function.

For finding the set of maximum vertices LM, all vertices of the mesh MS have to be examined. It is tested whether a vertex' interest value $\mathrm{DoI}(V)$ is greater than the DoI value of all vertices in the set $neighbors(V)$ of its direct neighbors in the mesh:

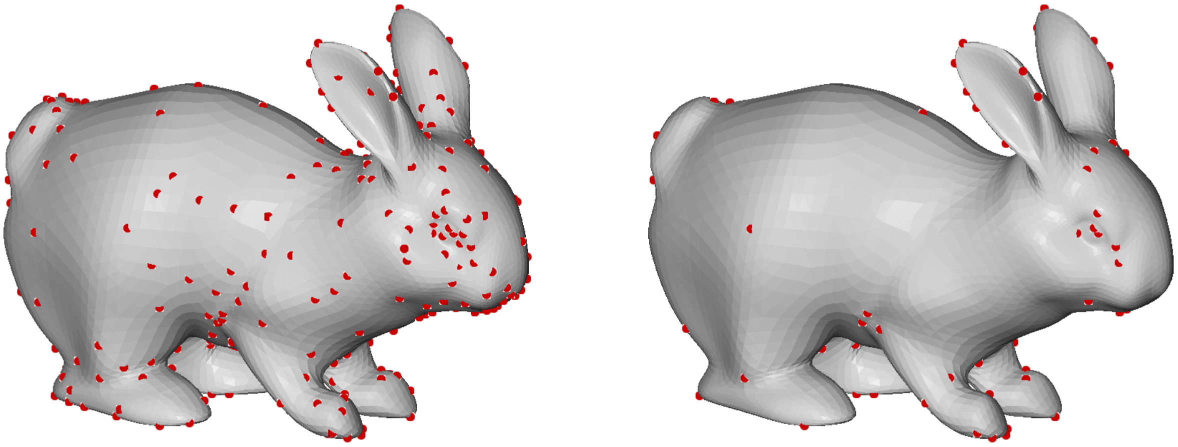
$$LM = \{V \in MS \mid \forall V_i \in neighbors(V) : \mathrm{DoI}(V) > \mathrm{DoI}(V_i)\} \tag{5.1}$$

Thus, LM comprises maxima of the DoI function in two dimensions because the DoI value of a vertex is compared with all its neighbors in the mesh.

All local maxima of the degree of interest function become wavefront initiators in the next step. Alternatively, a thresholded local maximum can be used. In order to have a normalized threshold, its computation is based on the difference Δ_{DoI} between maximal and minimal DoI value for the specific shape:

$$LM = \{V \in MS \mid \forall V_i \in neighbors(V) : \mathrm{DoI}(V) > t_{\mathrm{2D}} * \Delta_{\mathrm{DoI}} + \mathrm{DoI}(V_i)\} \tag{5.2}$$

Thus, positive threshold factors t_{2D} reduce the number of vertices that are local maxima on the mesh in the DoI function. In particular, this reduces the number of insignificant local maxima where the DoI value of a local maximum vertex is only slightly bigger than one or more of its neighboring vertices (see example in Figure 5.8). Further influence on the algorithm can be taken by limiting the set of wavefront initiators to the k most significant local maxima.



(a) No threshold.

(b) With positive threshold of 0.005 applied.

Figure 5.8: Reducing the number of insignificant local maxima by employing a threshold. The local maxima are indicated by red dots; the fold index was used as the DoI function.

5.2.4 Wave Propagation on Polygonal Surfaces

Once all local maxima LM have been found, the actual wave propagation process starts (see illustration in Figure 5.9 and pseudo-code in Listing 5.1). This process identifies the list of wavefronts (LoW) on the surface of the mesh. The initial wavefront $LoW[0]$ comprises these local maxima LM. To move from one wavefront to the next, all direct neighbors of vertices in the current wavefront $V \in LoW[i].vertices$ are detected. If these new vertices have not been visited before they are stored as the next wavefront $LoW[i+1].vertices$.[1] These steps are repeated iteratively until all vertices in MS have been visited. Two examples are shown in Figures 5.10(a) and 5.10(b).

Edges between two consecutive wavefronts are called *inter wavefront edges*. In order to distinguish them from *wavefront edges*, both types of edges are marked using specific flags while proceeding with wave propagation. In addition, the wavefront edges are stored in the data structure $LoW[i].edges$ that keeps the list of wavefronts.

When two wavefronts meet in the real world they interfere with each other. This interference effect was not modeled for the external skeleton extraction algorithm. In

---

[1] At this point the algorithms makes use of the HUYGENS' principle [Huy90]. The algorithm regards every point on the wavefront as a new wave initiator. Since the new vertices represent the one full propagation step, the union of all vertices that have not been visited to that point make up the new wavefront.

```
list_of_wavefronts
wavefront_propagation(mesh MS) {
  list_of_wavefronts LoW;
  wavefront W;
  set_of_vertices WV, WV_new;
  set_of_edges WE;
  vertex P;
  int id = 0;

  // initial wavefront from local maxima
  WV = local_maxima(MS);
  for (∀ P ∈ WV) {
    P.visited = true;
    P.wavefront_id = 0;
  }
  W.vertices = WV;
  W.edges = (wavefront_edges(WV))
  LoW.add(W);

  while (not all P ∈ MS visited) {
    id++;
    for (∀ P ∈ WV)
      for (∀ neighbors_in_MS N of P)
        if (!N.visited) {
          WV_new.add(N);
          N.visited = true;
          N.wavefront_id = id;
        }
    W.vertices = WV_new;
    W.edges = wavefront_edges(WV_new);
    LoW.add(W);
    WV = WV_new;
    WV_new.empty();
  }
  return LoW;
}

set_of_edges
wavefront_edges(set_of_vertices WV) {
  set_of_edges WE;

  for (∀ P ∈ WV)
    for (∀ neighbors_in_MS N of P)
      if (N.wavefront_id == P.wavefront_id)
        WE.add(edge(N, P));
  return WE;
}
```

Listing 5.1: Wavefront propagation.

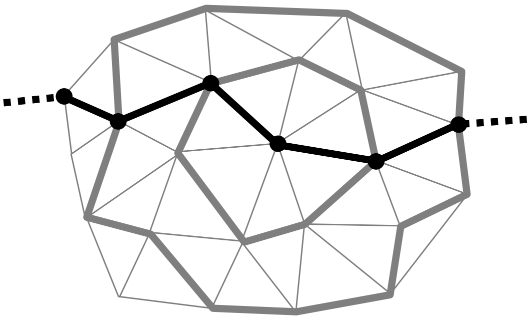
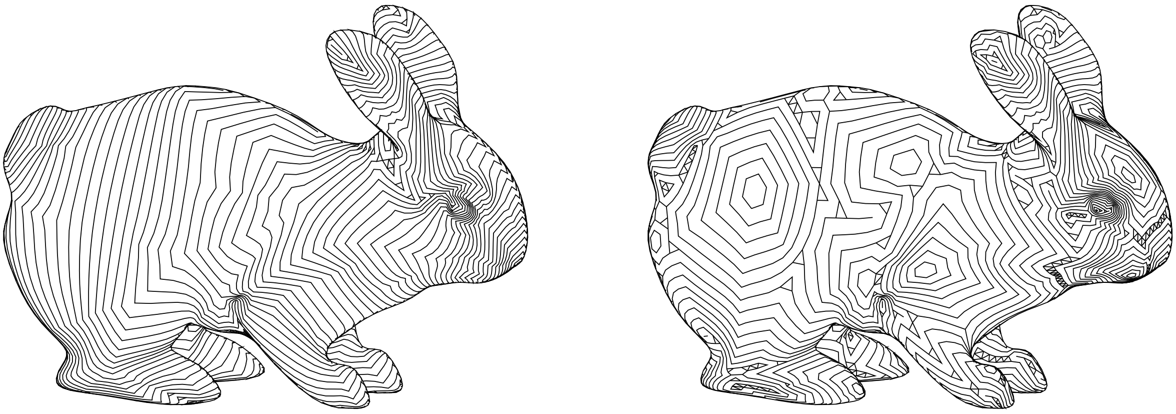


Figure 5.9: Schematic view of wavefront propagation on a triangular mesh. Wavefront edges are the thick gray edges whereas the inter-wavefront edges are thin gray. The black dots represent vertices with local maxima in the DoI function. The black edges are part of the extreme ridges.



(a) One starting vertex at the tip of the nose.

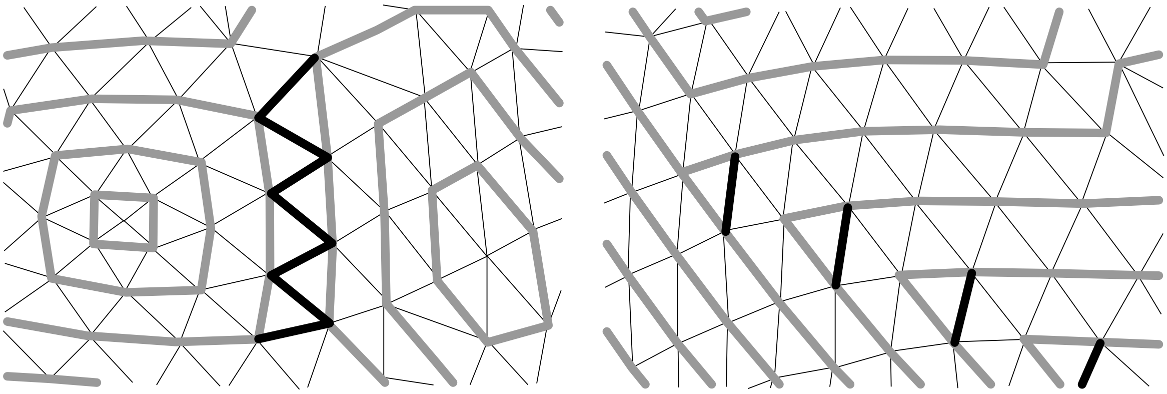
(b) Many starting vertices.

Figure 5.10: Propagation of wavefronts on the surface of a triangular model, without correction. Note the wrongly marked edges between meeting wavefronts and at merged wavefronts.

fact, this is not necessary at all since the wavefront propagation only serves as a means to traverse the mesh *MS*. Interfering wavefronts would assign two different wavefront levels to the same vertex. Instead, when wavefronts meet they are merged and only the outer border of the merged fronts is continued as a new wavefront.

5.2.5 Regularization

During the process of wave propagation, unfortunately, too many edges are marked to be part of a wavefront. This occurs, e. g., when two wavefronts meet (see examples in Figure 5.10). In such cases, two neighboring vertices are part of wavefronts that have the same age (the same distance from their initiators) but originated from different wavefront initiators (see schematic illustration in Figure 5.11(a)). This causes the edge between both vertices to be marked as a wavefront edge although it is actually an inter wavefront edge. A similar case can occur when two wavefronts from different wavefront initiators meet and consequently merge (see schematic illustration in Figure 5.11(b)). If this merging of wavefronts occurs at a sharp angle this can also cause edges to be marked as wavefront edges which are actually neither wavefront not inter wavefront edges. Thus, in order to ensure proper behavior and correct results of the remaining part of the algorithms, those erroneously marked edges have to be eliminated. This process is called *regularization*.



(a) Erroneously marked edges when two wavefronts meet. Both wavefronts, although starting from different initiators, have the age three. Thus, the edges in the mesh between them are erroneously marked.

(b) Erroneously marked edges when two wavefronts meet at sharp angle and are merged. If in such cases two edges of a triangle are part of the merged wavefront, the third edge of this triangle is erroneously marked as well.

Figure 5.11: Schematic illustration of cases for which regularization of wavefront edges is needed. Regular mesh edges are thin black lines, wavefronts are thick gray lines, and erroneously marked edges are shown as thick black lines. See also examples in Figure 5.10.

In order to regularize the wavefronts, a set of *regularization edges* is determined. Regularization edges are those edges that are direct neighbors—clockwise and counterclockwise around any vertex $V \in LoW[i].vertices$—of inter wavefronts edges. Thus, this set of regularization edges comprises the theoretically possible wavefront edges. All correct wavefront edges have such an inter wavefront neighbor edge and the wrongly marked edges do not. Hence, finding the regularization edges is very straightforward: during the process of finding the new wavefront vertices all direct neighbors of edges between two wavefronts—clockwise and counterclockwise around any $V \in LoW[i].vertices$—are marked with a flag (see Figure 5.12).

The regularization edges (Figure 5.12(a)) are finally compared with the initially determined set of wavefront edges $LoW[i].edges$ (Figure 5.10(b)). The intersection of both sets (i. e., edges which are part of both—the initial set of wavefront edges and the set of regularization edges) yields the regularized set of wavefront edges with the unwanted edges eliminated (see Figure 5.12(b)).

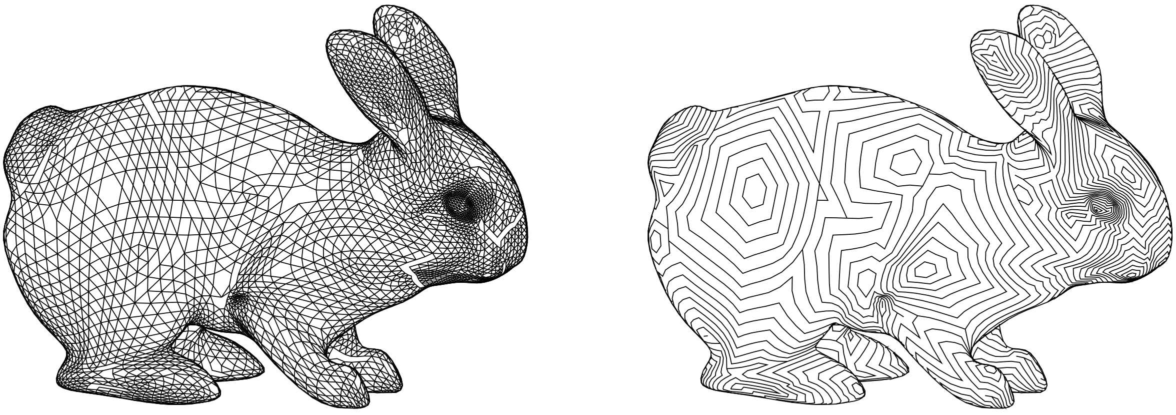

(a) Regularization edges.
(b) Regularized wavefront edges.

Figure 5.12: Regularization edges for use with the edges in Figure 5.10(b) and resulting regularized wavefront edges.

5.2.6 Following Local Maxima Curves On Wavefronts

After the regularization process has been completed, the wavefronts can be used for identifying the extreme ridges on the mesh. This process starts at the local maximum vertices LM as the initial vertices of the external skeleton ES. Then, the extreme ridges are traced by following local maxima of the DoI values from one wavefront to the next (see Listing 5.2).

```
set_of_edges
follow_maxima(mesh MS) {
  list_of_wavefronts LoW;
  set_of_edges ES;
  set_of_vertices WV, LM1D, LM1D_new;
  vertex P;

  LoW = wavefront_propagation(MS);
  LM1D = LoW[0].vertices;
  for (int i = 1; i < LoW.size(); i++) {
    LM1D_new = local_1D_maxima(LoW[i]);
    for (∀ P ∈ LM1D)
      for (∀ neighbors_in_MS N of P)
        if (N ∈ LM1D_new) ES.add(edge(N, P));
    LM1D = LM1D_new;
  }
  return ES;
}
```

Listing 5.2: Following the locally maximal DoI value curves.

The goal is to determine extreme ridges and their connection. This is accomplished by identifying local maxima of DoI on the wavefronts by comparing them to their direct neighbors on the same wavefront. Other vertices that they might be connected to in MS are not looked at in this step. This means that vertices are identified where the DoI function is locally maximal while restricting the domain of DoI to the current wavefront. These maxima are called *1D maxima* (or *restricted maxima*), as opposed to the elements of LM, which are called *2D maxima* (or *unrestricted maxima*).

In a second step, the 1D maxima are connected using inter wavefront edges (see the red edges and vertices in Figure 5.9). This can easily be accomplished by testing each inter wavefront edge whether it has two vertices which both are restricted local maxima. Due to triangulation artifacts, while looking only for direct edge links between two local maxima one can miss a connection in certain cases (see Figure 5.13(a)). Thus, for connecting the local maxima in such cases the algorithm also searches sideways on the wavefront. Hence, one wavefront edge on each wavefront is allowed to be part of the extreme ridge (see Figures 5.13(b) to 5.13(g)). Since some of these options are equivalent, only one can be used in a specific case.

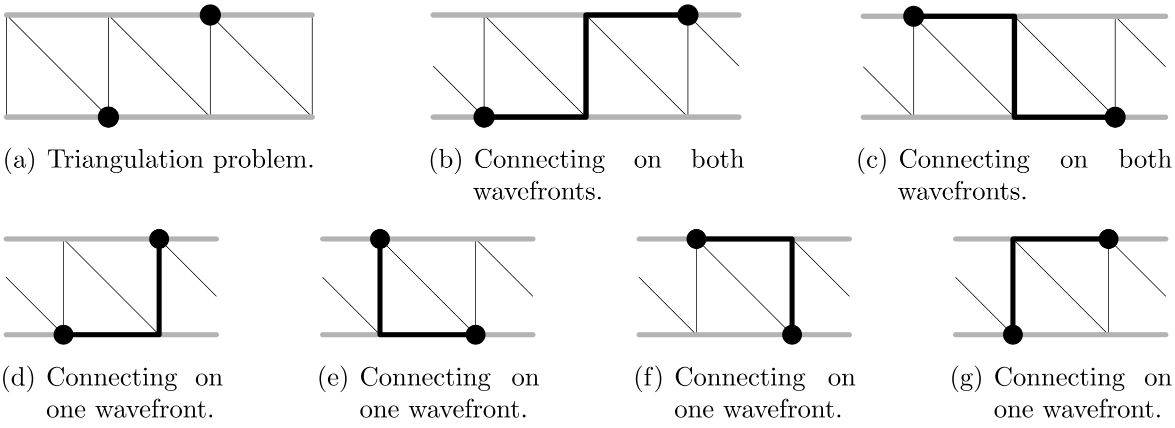
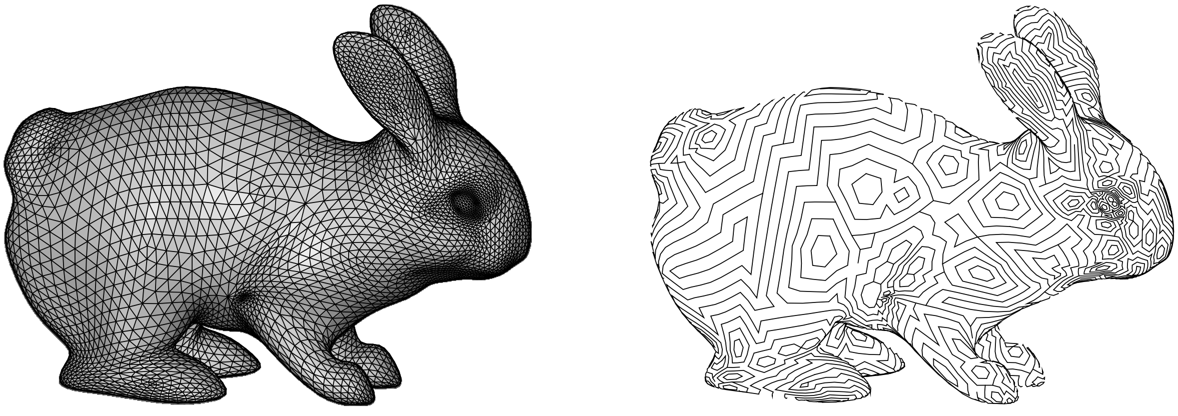

(a) Triangulation problem.
(b) Connecting on both wavefronts.
(c) Connecting on both wavefronts.
(d) Connecting on one wavefront.
(e) Connecting on one wavefront.
(f) Connecting on one wavefront.
(g) Connecting on one wavefront.

Figure 5.13: Potential problem when connecting restricted local maxima and how to avoid them. Wavefronts are visualized by thick gray lines and restricted local maxima by black dots. The proposed connection of the local maxima is shown by thick black lines.

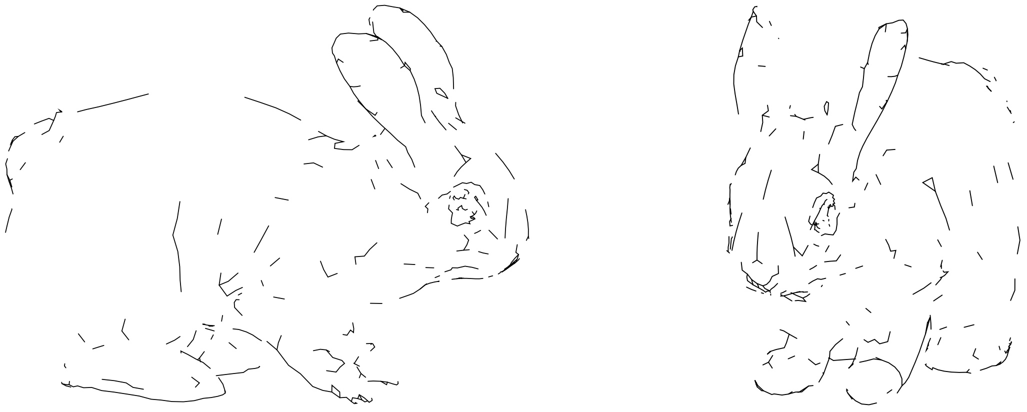
Similar to using a threshold for limiting the number of unrestricted maxima in Section 5.2.3, the number of insignificant restricted maxima can be limited by employing a one-dimensional threshold t_{1D}. Applying a one-dimensional threshold renders certain restricted local maxima to be too insignificant. Thus, during the process of following extreme ridges on the surface it may happen that one ridge ends since no connecting restricted local maxima could be found. However, it may occur that a few wave propagation steps further a chain of restricted local maxima can be found again. This does not pose a problem to the algorithm since the process of linking restricted local maxima is not dependent on having a chain of extreme ridge edges to begin with. Instead, only two local maxima on adjacent wave fronts are necessary.

An example where the process is applied to a specific model using the fold index as the DoI function is shown in Figure 5.14. Figure 5.14(a) shows the original shape for which the external skeleton is to be computed. Figure 5.14(b) shows the regularized wavefronts and Figure 5.14(c) shows the derived external skeleton.



(a) Original object.

(b) Regularized wavefronts.



(c) Resulting external skeleton.

(d) Front view of the skeleton.

Figure 5.14: External skeleton consisting of a network of extreme ridges computed for the bunny model. The DoI function used in this example is the fold index with function f_2.

5.2.7 External Skeleton Graph

At this point the network of extreme ridges has been identified. This network is called the external skeleton of the shape. After it has been found, the edges of the external skeleton can be displayed and stored by traversing the edges of the mesh and selecting the skeleton edges. Alternatively, they can also be assembled into a skeleton graph for further processing. This is also done by traversing the edges of the shape and selecting the skeleton edges. However, once a skeleton edge is found, its neighbors are examined to determine whether they are also skeleton edges. Thus, connected sets of skeleton edges can be identified by a recursive search. By assigning a certain flag to each edge that has already be found the whole skeleton graph is extracted.

5.3 Algorithmic Considerations

In this section, considerations in terms of efficiency of the algorithm will be presented. In addition, some potential modifications of the method an their impact on the results will be discussed.

5.3.1 Efficient Computation

The wave propagation algorithm essentially consists of the four steps outlined in Section 5.2.1. The first step (identifying the local maxima) requires comparing the DoI value of each vertex in the mesh with those of its direct neighbors. This means that every edge (and its associated vertices) has to be touched twice: once for each of its two vertices in order to check the value of the opposite vertex. Hence, the computational complexity of this step is $\mathrm{O}(n_v + n_e)$. The second step (establishing the wavefronts) also is based on examining the direct neighbors of vertices and all vertices are touched in this step as well. Hence, the same complexity $\mathrm{O}(n_v + n_e)$ applies. During the following step of wavefront regularization every edge is examined and corrected based on the regularization information derived previously (this only required constant time per edge). Finally, the local maxima are traced across the wavefronts. For this purpose local restricted maxima are detected which also visits every vertex and compares it to its direct neighbors requiring $\mathrm{O}(n_v + n_e)$. Afterwards, all edges are examined again to detect those inter wavefront edges that connect two vertices that are restricted local maxima. This yields the external skeleton edges in an additional $\mathrm{O}(n_e)$. Therefore, the total computational complexity of the algorithm is $\mathrm{O}(n_v + n_e)$.

For the computation of local maxima being computationally feasible, a data structure that allows for a fast search for local information in the polygonal mesh is required. In particular, the DoI values of neighboring vertices in the mesh need to be compared in order to determine local maxima (both restricted and unrestricted) during the computation of the external skeleton. This requires finding the neighbor of a specific vertex for accessing its DoI value. The same operation is also required for assembling the skeleton edges into a skeleton graph after the wave propagation process. A data structure that provides this local neighborhood information in constant time $\mathrm{O}(1)$ is required.

5.3.2 Winged Edge Data Structure

In order to fulfill the requirements mentioned above, the Winged Edge data structure as introduced by BAUMGART [Bau75] was chosen (see also [Gla91]). As the name of the data structure suggests, it is based on an (global) edge list and consists of locally linked edges. For every shape, the data structure additionally stores a list of faces and vertices (see Figure 5.15). Each face and each vertex maintains a (local) list of edges that are associated with it. Thus, each edge will be stored more than once. Consequently, in order to store the data associated with edges only once, each edge in an edge list—global and local—only holds a pointer to a unique edge data object. This edge data in turn stores the local information for in edge. This includes, at least, links to both the adjacent faces, the associated two vertices, and—hence the name of the data structure—links to the wings of an edge, i. e., the edges clockwise and counterclockwise around each of the two faces (see Figure 5.16). In addition, this edge data object may store other information regarding the edge such as, for example, flags.

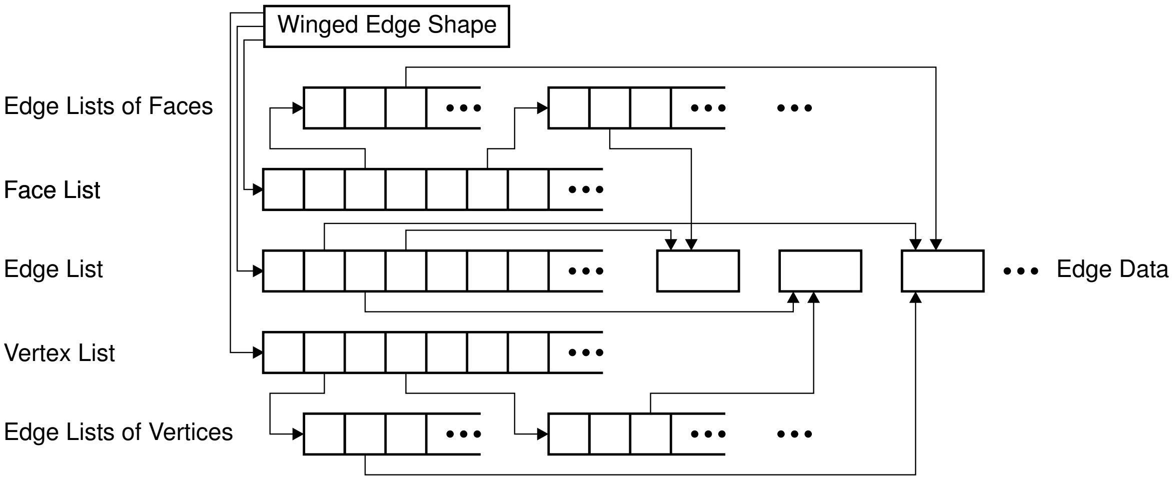


Figure 5.15: Schematic illustration of the Winged Edge data structure.

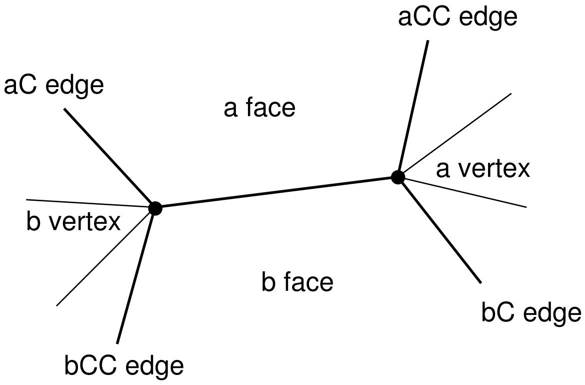


Figure 5.16: Schematic illustration of the neighborhood relations stored in the edge data for every unique edge within the Winged Edge data structure.

The (local) edge lists around each vertex and around each face on the one hand and the linked edge wings on the other hand allow the local traversal of the mesh starting at any vertex, edge, or face. In addition, the number of edges in an edge list of a face is always three for triangular meshes.[2] The number of edges in an edge list (degree) of a vertex, however, may vary. On the other hand, the degree of most vertices in a regularly triangulated mesh is six.[3] Hence, most queries for connectivity information

---

[2] Note, however, that the Winged Edge data structure is defined for arbitrary meshes.

[3] A regular triangulation is generated, e. g., by applying subdivision techniques such as LOOP's scheme [Loo87]. In such meshes, all but a few extraordinary vertices have the degree of six.

can be completed in constant time (i. e., O(1)). Thus, the Winged Edge data structure provides the local connectivity information which is needed for efficient computation of the external skeleton.

5.3.3 Modifications of the Algorithm

So far, the only parameter for adaptation to the area of application is the choice of a DoI function. Thus, in order to allow for a wider variety of adaptations, some possible modifications of the wave propagation based algorithm for external skeleton extraction will be discussed in this section.

Significance-Driven Wavefront Initiators

In the algorithm as it has been described above, all identified unrestricted local maxima are used as wavefront initiators at the beginning of wavefront propagation. This, however, gives all local maxima the same significance. Local maxima that have a high DoI value are treated the same as those with low DoI values. This behavior might not be a desired trait in every case because they have not the same significance (i. e., DoI value).

The algorithm can be modified in that the DoI value of the unrestricted local maxima is considered when selecting the wavefront initiators for the wave propagation. After the unrestricted local maxima have been identified, they are sorted using their DoI value. The following wave propagation phase is only started with the vertex (or vertices if there are more than one) that have the highest DoI value of all local maxima. Then, after completing each individual wave propagation step, the lowest DoI value on any wavefront visited during this step is determined. All local maxima that have a higher DoI value than this lowest DoI value on a wavefront and that have not been visited thus far are also added to become new wavefront initiators.

This modified procedure allows for further influence on the way the skeleton is extracted in that it favors unrestricted maxima with high DoI values over those with low DoI values because this allows to track more significant features first.

Using Local Minima Instead of Local Maxima

Up to now, only maxima of the DoI function have been considered in order to find the wavefront initiators as well as tracing the skeleton on the wavefronts. Some functions, however, do not only classify features through high DoI values but also through low DoI values. Among theses functions that have this property are, for example, average curvature, regular and modified shape index, and the fold indices. Thus, depending on the chosen DoI function, in some cases it is advisable to also look for minima of this DoI function instead of just using maxima.

When local minima are used instead of maxima, the rest of the algorithm has to be adapted accordingly. For example, the sign of the thresholds that are used has to be modified. In addition, the sorting order has to be reversed when using significance-driven wavefront initiators.

Multiple Runs

One single run of an algorithm might not always yield the desired result, even with the many possible ways of parameterization. For example, only using maxima or only using minima when extracting the external skeleton may only detect some of the features. Similarly, one single DoI function may not be able to capture all the features a user is interested in. One option is to modify the DoI function to include all the desired traits. This will often not only prove to be difficult to achieve but also the resulting function will be difficult to handle in terms of complexity and/or numerical stability. Rather than coming up with such a very complex DoI function, the easier way is to employ multiple runs of the algorithm, each with a different parameterization (e. g., different DoI functions or different local extrema to be followed). After the separate runs have been completed, their results are combined into the final external skeleton.

An example for the result generated with multiple runs is shown in Figure 5.17. In a first run, the external skeleton of the shape in Figure 5.17(a) was generated using κ_2 as the DoI function and looking for local minima (see Figure 5.17(b) where wavefronts along with the unrestricted local maxima are shown). Figure 5.17(c) shows the combined skeleton of the first run and the second run which used κ_1 as the DoI function and a local maxima based wave propagation.

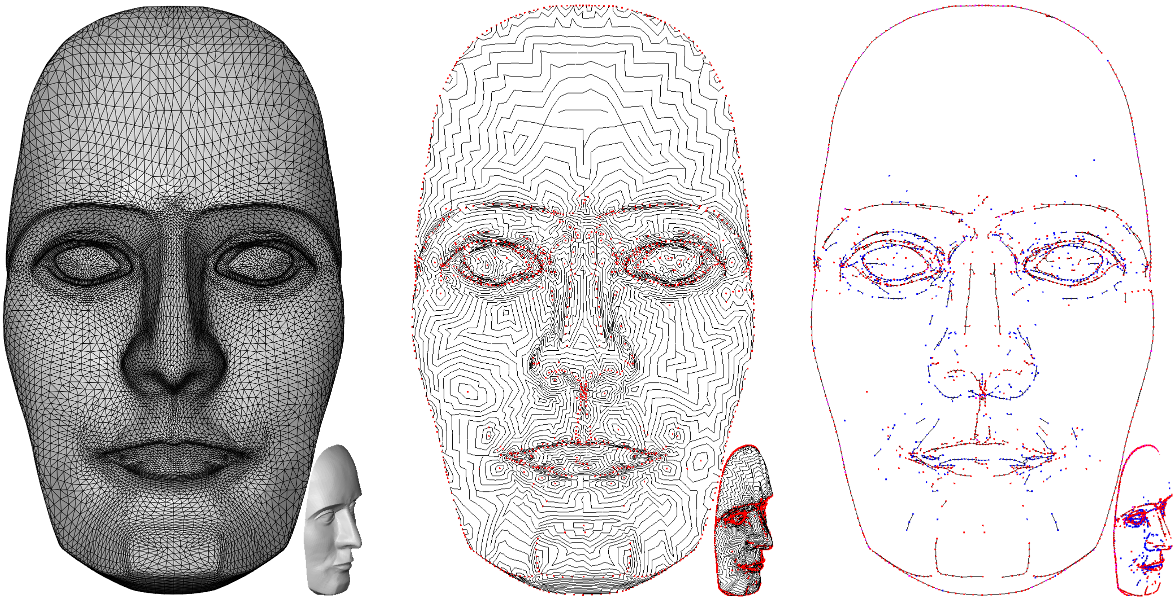
5.4 Analysis of the External Skeleton

After the algorithm for extracting an external skeleton from a polygonal mesh has been presented, the external skeleton itself will be examined. In particular, two properties of essential shape representations will be looked at in detail at this point. On the one hand, it will be shown that external skeletons facilitate the representation of different scales of objects. On the other hand, it will be demonstrated how the shape can be reconstructed from the external skeleton.

5.4.1 Scale

One of the important requirements named in Section 2.3 is that an essential shape representation has to allow for multi-scale representation of shape (Requirement 4). Therefore, the external skeleton should provide means to code and decode multiple scales of a shape. The following methods can be employed for this purpose:

- prune the external skeleton,



(a) Original object.

(b) Wavefronts and external skeleton, minima of κ_2 only.

(c) Resulting final external skeleton, maxima and minima combined.

Figure 5.17: External skeleton curves computed for a face model. The local extrema are visualized as dots (red for maxima and blue for minima). The insets are the same models viewed from a different angle.

- prune the set of local maxima LM,
- low-pass filter the original mesh, or
- any combination of these.[4]

Figure 5.18 illustrates this procedure. The different paths that are possible for getting from the initial mesh MS_0 to the final skeleton ES_i are shown. In the diagram, the vertical arrows indicate the two mayor stages of the algorithm: the identification of the maxima though exhaustive search and the wave propagation for constructing the external skeleton graph. Each of the methods named above constitutes one way of achieving multi-scale for external skeletons. These are indicated by the horizontal arrows. Low-pass filtering MS_0 happens before staring the algorithm, pruning LM_i after the maxima have been identified, and pruning the ES after the algorithm has been completed.

As previously discussed in Sections 5.2.3 and 5.2.6, the pruning of the set of wavefront initiators and of the external skeleton was implemented by applying thresholds to determine the significance of the local maxima. Hence, for both unrestricted as well as restricted local maxima a threshold value can be determined for which the specific maximum is going to be pruned yielding their specific scale value.

---

[4] In addition to these, scale could also be incorporated in a DoI function as mentioned in Section 4.3.2. This could be done, as mentioned before, by defining a footprint using a scale value that determines the scale of features that are emphasized.

$$\begin{tikzcd}
MS_0 \arrow[rrrr, "low-pass\ filter"'] \arrow[d, "exhaustive\ search"] & & & & MS_1 \arrow[d, "exhaustive\ search"] & & & \\
LM_0 \arrow[rr, "prune"'] \arrow[d, "graph\ construction"] & & LM_1 \arrow[d, "graph\ construction"] & & LM_2 \arrow[rr, "prune"'] \arrow[d, "graph\ construction"] & & LM_3 \arrow[d, "graph\ construction"] & \\
\mathrm{ES}_0 \arrow[r, "prune"'] & \mathrm{ES}_1 & \mathrm{ES}_2 \arrow[r, "prune"'] & \mathrm{ES}_3 & \mathrm{ES}_4 \arrow[r, "prune"'] & \mathrm{ES}_5 & \mathrm{ES}_6 \arrow[r, "prune"'] & \mathrm{ES}_7
\end{tikzcd}$$

Figure 5.18: Diagram showing the different possibilities for introducing multi-scale capability to external skeletons. This is achieved by combinations of low-pass filtering the original mesh MS_0 or pruning the set of local maxima LM_i or the external skeleton ES_j.

Incorporating the lowest level of multi-scale from Figure 5.18 into external skeletons is very straight-forward. Starting with the external skeleton without thresholding applied, all vertices of the skeleton are assigned their specific scale values that are stored along with the geometry and adjacency data. When the data structure is organized progressively using the scale values of the vertices as indicators, a skeleton at a specific scale level can easily be extracted.

The two higher levels of scale in Figure 5.18 (low-pass filtering of MS_0 and pruning of LM_i) can be used for generating separate starting conditions for the process outlined above. It is not possible to unify, for example, the scale values generated when pruning LM_i with those of pruning ES_j because the former changes the starting conditions for the wave propagation process which might lead to a different external skeleton. The skeleton generated from a pruned LM_{i+1} will most certainly not simply be a subset of the skeleton generated from the not pruned LM_i. Thus, using the low-pass filtering of MS_0 and pruning of LM_i it is possible to not only construct a linear succession of progressively stored skeleton edges but instead constructing a series of those with growing starting scale levels.

Figure 5.19 illustrates the multi-scale character of the external skeleton. In the example different levels of scale of the same object are shown. It can be observed that indeed the most significant features are preserved even at high scale levels while more insignificant features are disregarded at lower scale levels.

For the skeletons shown in Figure 5.19, Table 5.2 gives the numbers of vertices and edges of each external skeleton compared with the original model from Figure 5.17(a). The numbers show that even if low levels of scale are used the amount of vertices and edges that are significant for the important features of scale is small compared to those in the original mesh.

5.4.2 Reconstruction

Of the requirements discussed in Section 2.3, one of the most fundamental ones is Requirement 1. It states that the ESR must not only be constructed from a shape but

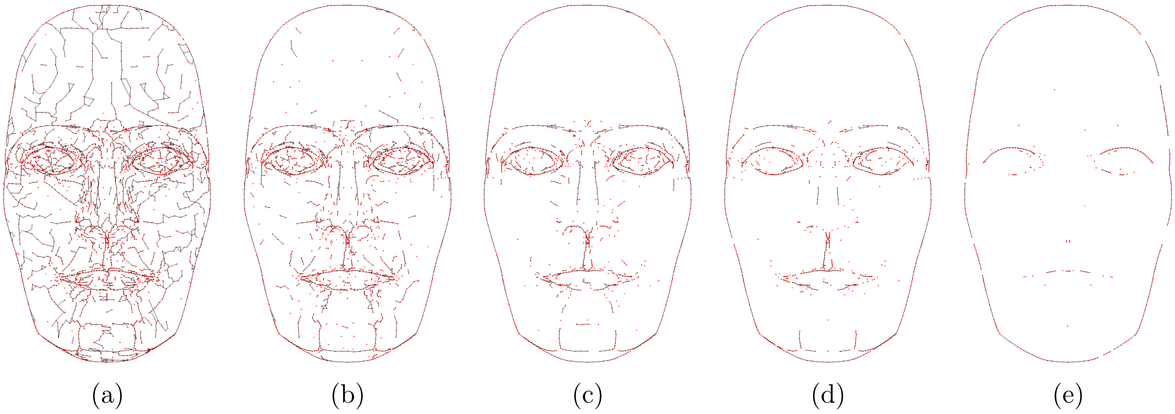


Figure 5.19: Growing level of scale of an external skeleton of the same model achieved through pruning LM and ES. The skeleton was computed from the shape shown in Figure 5.17(a) using κ_2 as the DoI function and looking for minima.

Model	Original	External Skeletons				
Figure	5.17(a)	5.19(a)	5.19(b)	5.19(c)	5.19(d)	5.19(e)
vertices (#)	11,710	2,955	1,967	1,274	938	389
vertices (%)	100	25.2	16.8	10.9	8.0	3.3
edges (#)	35,124	3,402	1,943	1,161	774	339
edges (%)	100	9.7	5.5	3.3	2.2	1.0

Table 5.2: Numbers of vertices and edges of the external skeletons shown in Figure 5.19 compared with the original model shown in Figure 5.17(a).

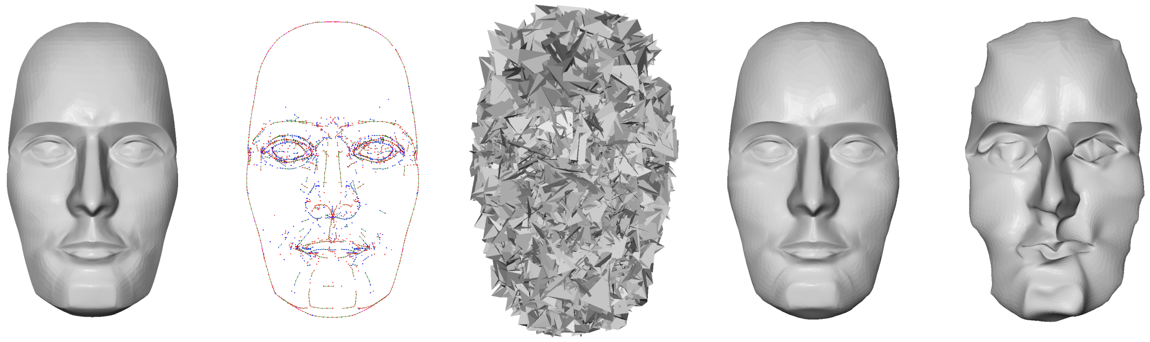
that it must also be possible to reconstruct the shape from the ESR. A reconstruction of the shape from the external skeleton basically requires two steps:

1. establishing a topology for a triangle mesh and
2. establishing the geometric location of the vertices.

In order to demonstrate step 2, the process of establishing the location of the vertices for the reconstruction will be discussed first. For this purpose the topology will be used that is available in the original model (see Figure 5.20). Afterwards, step 1 will be demonstrated by showing how to establish a topology to be used for the shape reconstructed from the external skeleton.

Establishing the Location

For demonstrating step 2 of the reconstruction, in a first stage white noise is applied to all vertices except the ones in the skeleton (see Figure 5.20(c)). Afterwards, relaxation [vO95] is applied to get a smooth surface at the regions not specified by the skeleton. To ensure that the skeleton does not change during this process the vertices on the skeleton are frozen while applying the relaxation. Figure 5.20(d) shows the result after freezing the output and applying the relaxation whereas Figure 5.20(e) demonstrates what happens without freezing the external skeleton vertices.



(a) Original shape. (b) External skeleton. (c) Original mesh with noise. (d) Relaxation applied. (e) Relaxation, no freezing.

Figure 5.20: Simulating the reconstruction of the external skeleton by employing white noise.

Since reconstruction *without* taking the external skeleton into account produces the badly deformed result in Figure 5.20(e), it can be concluded that the external skeleton captures the essence of the shape in Figure 5.20(a), and the large similarity between the shapes in Figure 5.20(a) and Figure 5.20(d) shows the effectiveness of the method. In order to get an even more accurate reconstruction, one could store the local curvature and curvature directions of the skeleton vertices to guide the process of generating a new mesh. This might additionally improve the mesh quality of the reconstruction.

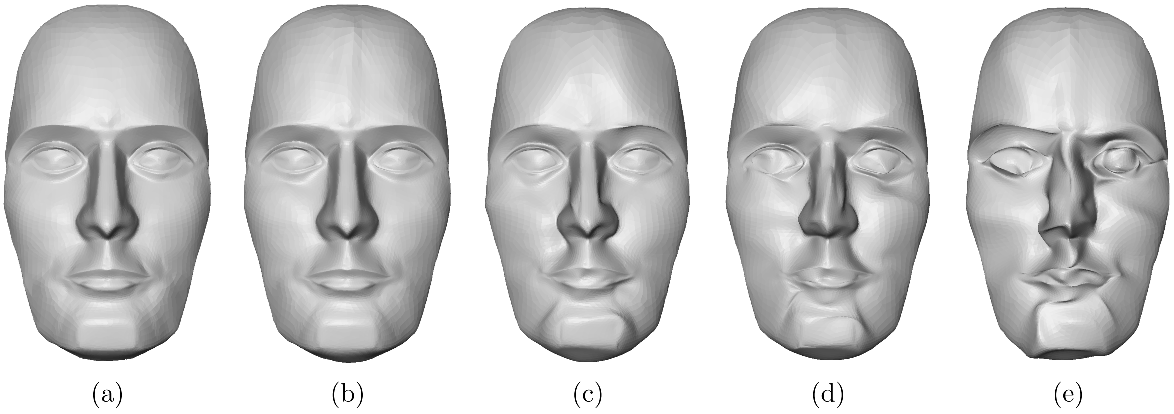


Figure 5.21: The effect of scale on the reconstruction. The reconstructed shapes were generated from the external skeletons shown in Figure 5.19.

Figure 5.21 shows the effect of scale on the reconstruction. The reconstructions were generated from the external skeletons shown in Figure 5.19. As can be observed, the more detail is preserved in the external skeleton the better the reconstruction matches the original shape. It is interesting, though, that a significant reduction of the number of vertices and edges in the ES can be tolerated before visible differences between the reconstruction and the original shape occur. Of course, this should be investigated more thoroughly by means of rate distortion assessments, but it may be interesting to study the merits of external skeleton as a compression device.

Establishing and Refining a Mesh Topology

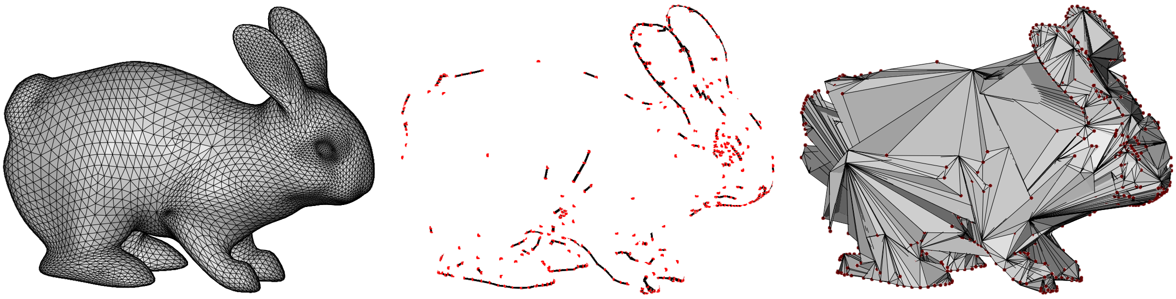
Step 1 of the reconstruction—establishing the necessary topology of a triangle mesh—can be attempted by employing known techniques for reconstructing a shape from a set of points on the shape's surface. Those were developed, for example, by Veltkamp based on Delauney triangulation [Vel93, Vel94] or by Amenta, Bern, and Kamvysselis using Voronoi diagrams [ABK98, AB99]. These types of algorithms, however, require a fairly dense set of points for all parts of the surface of the shape. This is not the case for external skeletons. Thus, point based algorithms may not be appropriate for external skeletons.

Other algorithms, e. g., as used in the system Teddy [IMT99], exploit curve based reconstruction methods. In the Teddy system, the mesh is established from a polygon drawn in 2D by first computing the chordal axis, raising it, and tessellating the surface which spans between the polygon and the axis. This process is repeated for the opposite side of the shape. The advantage of such an algorithm is that it can work with polygons rather than an unorganized point set. However, it needs a closed polygon to start with. This, too, is not always given for external skeletons. For a subclass of shapes (e. g., the face model which contains an almost closed polygon even in high scales as shown in Figure 5.19(e)) this might be an option to pursue. However, for general shapes it will not be feasible.

Finally, one might try to apply an approach using the projection of the skeleton onto a sphere to reconstruct the shape. A sufficiently densely tessellated sphere is constructed and centered to the skeleton. From the center of the sphere, the skeleton is projected onto the tessellated sphere and the vertices and edges of the skeleton are matched with the ones of the sphere. This way a topology is assigned to the skeleton. Unfortunately, this approach can only reconstruct shapes of genus zero. Also, the matching process of the sphere with the skeleton can not always guarantee to find a plausible topology.

This discussion shows that re-establishing a topology from the external skeleton graph is difficult. However, reestablishing the topology is not really necessary because it is not lost during the wave propagation process. After the skeleton has been determined, the original topology is still available and can be used. On the other hand, it is neither necessary nor advisable to keep the entire tessellation of the mesh. The original mesh is by far too detailed for the purpose of storing the topology. Thus, the mesh's complexity has to be reduced.

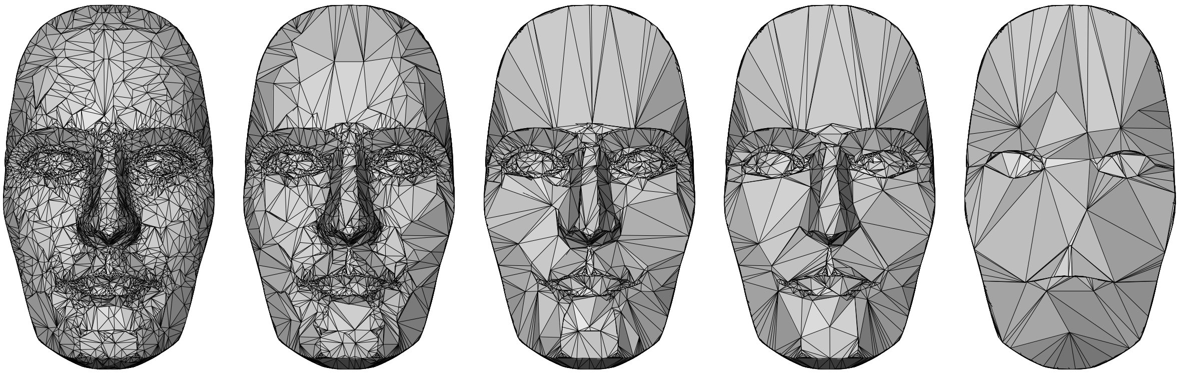
For this purpose, an algorithm based on edge collapse operations as it was introduced in Section 3.3.2 can be employed. It has to be modified such that vertices and edges that are part of the skeleton are not moved or collapsed, respectively. However, a simple length based scheme for selecting the edges that are to be collapsed next as it has been applied for internal skeleton extraction is not advisable in the current case. Using such a scheme can possibly result in a degenerated mesh with most non-skeleton edges and vertices collapsed into one or few final vertices as shown in Figure 5.22.



(a) Original shape. (b) External skeleton. (c) Degenerated collapsed mesh.

Figure 5.22: Possible degenerate result when collapsing of edges is not restricted to those starting from the external skeleton. Although the skeleton is fairly densely tessellated, the simplified mesh is degenerated (in this example a different model was used to show the effect because in the face model the sharp edge around the face prevents the effect to some degree).

Those edges are to be selected for an edge collapse operation that share exactly one skeleton vertex. This scheme ensures that the mesh is simplified starting from the skeleton. The mesh's complexity is reduced at approximately the same speed starting equally from all skeleton edges and vertices. In addition, before each edge collapse step it has to be tested that this operation will not create a degenerated mesh or reduce its genus. Only then the edge collapse operation is actually performed. When repeating this process iteratively until all non-skeleton edges have been collapsed the simplified mesh is created. Figure 5.23 shows the resulting simplified meshes for all skeletons previously shown in Figure 5.19.



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

Figure 5.23: Mesh simplification for preservation of the mesh's topology using edge collapse operations for the external skeletons shown in Figure 5.19.

For a complete reconstruction it is necessary to re-establish a mesh with a balanced tessellation where every polygon has approximately the same size. This is not the case after the mesh decimation step. In order to re-create a mesh with smaller, regular triangles in a balanced tessellation, adaptive subdivision can be applied (for details of the process of adaptive subdivision see Section 4.6 and [IHK03]). For this purpose, the subdivision is applied only to those faces that are too large. This it iteratively repeated until the faces reach the desired size. This can be measured by comparing the lengths of the edges of a face with the average edge length before the external skeleton extraction process. If, for example, the average length of the edges of a face is longer than the average length of all edges before skeletonization, this edge is subdivided according to LOOP's scheme [Loo87]. Because LOOP subdivision is a scheme based on edge split steps this will, however, also cause the skeleton edges to be split. Since these are already relatively small (they have not been touched during the mesh decimation step) this will cause too small edges to be generated. In order to prevent this from occurring, after each step of adaptive subdivision those divided edges are merged again by applying edge collapses to those edges only.

Alternatively, for establishing a smooth mesh with a balanced tessellation, other mesh smoothing algorithms as, for example, the *beautification* method presented by IGARASHI and HUGHES [IH03] can be applied. However, the modifications to the mesh introduced by this method will also cause the removal of the specific location and connectivity of the external skeleton edges.

5.4.3 Robustness of the Method and Potential Problems

Meshes may contain a number of errors which are caused by the way the geometric mesh data is created or acquired (e. g., creating 3D models using a modeling suite such as 3D Studio MAX or acquiring it through 3D scanning). Errors in a modeling suite can occur, for example, due to mistakes of the modeling artist or because of incorrect computation of modeling operations in the software. Wrong parts of meshes can be created in 3D scanners due to certain physical phenomena such as reflections of laser rays. Although these mistakes can be reduced (see, e. g., NEUGEBAUER [Neu96] or TEUTSCH [Teu03, TITW04]) they can never completely be avoided.

One of the properties of an ESR that is important in terms of usability is how sensitive it is towards errors in the mesh it is applied to. Errors in the mesh include, but are not limited to, wrong normals, double edges, double faces, holes (i. e., missing faces), and open patches (those are usually not considered to be errors but are a special case of a geometric mesh).

Wrong normals are no problem for the wave propagation based skeleton extraction since it does not use normals at any stage. Even the edge collapse operations or adaptive subdivision used for preserving the topology do not use normals at any point. In fact, it is easier to check the model for correct normals at the simplified level—either manually or automatically—because this representation contains far less faces.

Double edges do not pose a problem to the algorithm either. If they are part of features on the mesh it just means that this condition will be preserved in the skeleton. However, they can easily be removed in a pre-processing stage. Similarly, double vertices and double faces do not cause problems and can also be easily removed by pre-processing.

Holes in the mesh and meshes consisting of open patches are similar in that they are characterized by edges that have only one face attached. However, holes are most certainly errors in the mesh while open patches have to be preserved. Missing faces in general are no problem for the algorithm because it is edge-based, i. e., it only uses edges and their neighboring edges so that missing faces are ignored. However, during the mesh decimation stage the missing faces may disappear or may also be enlarged. Thus, it is advisable to try to close the holes prior to skeleton extraction. On the other hand, the edges at the end of the patches can be marked as being part of the skeleton in order to preserve the open patches.

One potential problem with the presented method for extracting external skeletons is that only two-manifold shapes[5] can be used. This technical limitation is caused by using the Winged Edge data structure [Bau75] for shape description that can only describe two-manifold shapes. From an algorithmic point of view, the two-manifold character is not required since the wave propagation can easily handle non-two-manifold singularities due to the HUYGENS' Principle (see Section 5.2.2). This, on the other hand, would require one to use a different mesh data structure. However, most geometric models relevant in rendering are two-manifold or can be modified to be two-manifold without significant visual changes.

As already argued in Section 5.4.2, the topological genus of a shape cannot be captured when the skeleton is reduced to the graph only. If the edge collapse based topology preserving method is used instead, the topology is unlikely to change. Unfortunately, the edge collapse algorithm might collapse holes when no specific precautions are taken. However, the degeneracy check discussed previously to avoid the creation of degenerated meshes (e. g., to avoid reducing a tube to a single internal skeleton edge) can also be used to prevent the algorithm from closing holes in the shape. This ensures that the topological genus is preserved.

As could be seen in experiments, the algorithm works very well for approximately evenly tessellated meshes. If the tessellation is not balanced for the mesh, the fact that the distance of two wavefronts is one edge length and not a certain constant Euclidean distance causes problems (the reasons for this were discussed in Section 5.2.2). The problem of uneven tessellation could be solved, however, by applying some adaptive subdivision steps that subdivides larger triangles rather than smaller ones prior to the feature detection (similar to the adaptive subdivision in Section 5.4.2). Alternatively, re-meshing algorithms could be applied prior to skeleton extraction such as, for example, the method introduced by ALLIEZ, MEYER, and DESBRUN [AMD02]

Some difficulties may arise when using the external skeleton ESR in applications where connectivity of the skeleton is essential. There are cases when branches of the external skeleton graph originating from different neighboring wavefront initiators may not

---

[5] Two-manifold shapes are those shapes for which there is an ε environment around every point on the surface which is topologically equivalent to a disk.

meet. This may be due to numerical problems with the DoI function or to an uneven tessellation of the mesh.

Other problems may arise for shapes and a DoI function that produces the same values everywhere (or for a certain part of the mesh). For example, think of a regularly tessellated tube and a curvature based DoI function. In such cases the algorithm cannot find meaningful local maxima. This happens also when the DoI function yields noise instead of a constant value. In fact, in cases with constant DoI values the application of some noise may help to create (random) maxima to be used in the algorithm. These in turn will then be used for mesh simplification. This will not yield a meaningful skeleton graph but will make topology-preserving mesh simplification possible.

5.5 Summary

In this chapter, a new ESR scheme has been introduced for application to polygonal meshes. It extracts an essential shape representation that is called external skeleton that represents the essence of the shape with respect to a chosen ESR criterion. The algorithm is based on wave propagation using a DoI function as ESR criterion in order to traverse the mesh and trace local maxima of the chosen DoI function. The resulting graph comprises local shape information based on the DoI function. It has been shown that this method can be used to capture not only local information but also global properties such as the general form of the mesh. It has been shown how this representation can be used to derive different levels of scale as well as how the original shape can be reconstructed.

With respect to the requirements posed in Section 2.3, it can be seen that an external skeleton, defined as the network of extreme ridges, is a plausible option. Indeed, it allows for coding and decoding (Requirement 1). Instead of storing the whole mesh only the external skeleton is stored (with or without the topology depending on the specific application). When decoding, the positions and connectivity of the extreme ridges can be used. A reconstruction was discussed based on storing the topology along with the extreme ridges. This produces a mesh similar to the original one. Through this the algorithm also facilitates shape compression (Requirement 2; see Figures 5.19 and 5.21 as well as Table 5.2). An external skeleton does allow for multiple scales (Requirement 4) and progressive encoding in that the extreme ridges can be pruned based on the significance (DoI value) of the feature edges.

Moreover, an external skeleton is represented independently from the specific triangulation and allows for resampling (Requirement 5) in that the extreme ridges can be thought of as a connected graph rather than a subset of the triangle mesh. By manipulating the external skeleton instead of the specific mesh, the external skeleton also facilitates the editing of the shape at feature level (Requirement 7). Finally, it seems easily possible to use the external skeleton for shape matching (Requirement 3) by going stepwise from high to low scale versions of two external skeletons instead of using the triangle mesh. However, this has not been tested and remains for future work.

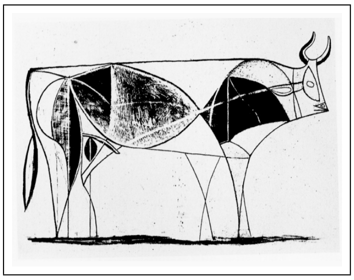
Due to employing a DoI function in order to classify what is to be regarded as important in a shape, the essential shape representation approach has a major advantage over previous approaches to feature extraction (e. g., [RKS00, HBK01, HBK02, WB01]). This means that it is a more general method because it is not limited to geometric features. The algorithm is not restricted anymore in terms of a specific concept of importance for detecting features. Instead, an appropriate DoI function can be defined that fulfills the specific requirements of the application which extracts the ESR.

Compared to algorithms based on thinning (for example, Rössl and Kobbelt [RKS00]) the ESR approach is able to find the actual location of the features. Thinning based algorithms, in contrast, first use segmentation based on a criterion and than extract a central axis of the segmented part of the surface. This axis, however, does not necessarily have to be identical with the actual local maximum of the criterion. The wave propagation based ESR, on the other hand, locates this local maximum on the wavefronts. Since no thinning is used in the process, the algorithm also does not have to cope with unwanted short curves emerging from the main skeleton curve that are due to the thinning process.

Belyaev, Bogaevski, and Kunii indicate that in order to represent global properties one has to employ internal skeleton structures (e. g., internal skeletons, linear or planar) while local properties are best represented by external structures (for example, ridges, external skeletons, feature lines) [BBK96, Section 3]. It has been argued in this chapter, however, that not all internal structures are well suited for representing global properties (see Section 5.1 and specifically Section 5.1.1). In addition, it could also be shown that a set of local properties (the wavefront propagation algorithm captures local properties) can in fact be used to capture aspects of global shape of the object (see Section 5.4.2).

Unfortunately, the presented essential shape representation cannot be used to capture all global shape aspects present in an object. Thus, it seems advisable to combine internal and external skeletons in one essential shape representation. Hisada, Belyaev, and Kunii have previously discussed such a combination based on an internal Voronoi structure [HBK01, HBK02] while Watanabe and Belyaev use an internal focal surface [WB01, BAK97]. However, all these approaches are based on planar internal structures and they are used to derive the external representation. Using a linear internal representation might add the missing global properties to the external skeleton while at the same time avoiding the problems discussed in Section 5.1.

In the following chapter, a different type of essential shape representation will be examined that also detects curves on the surface of a polygonal mesh. However, it captures the second aspect of shape that was mentioned in Section 2.1—the form of an object's silhouette. It differs in many factors from the ESRs discussed so far and requires a separate method of computation due to its specific properties and applications.



Chapter 6

ESRs in 2D: Silhouettes

Line drawings have played an important role in the area of artistic expression (e. g., any pencil or pen-and-ink drawing), scientific visualization (for example, medical or technical illustrations; see, e. g., HODGES [Hod89]), or entertainment graphics (such as in comics) for a long time. Hence, the automatic generation of such line drawings has been studied extensively in computer graphics. In particular, the area of *non-photorealistic rendering* (NPR) [GG01, SS02, FM03] has focused on two main directions of research in this respect: the generation of hatching to convey illumination as well as texture in an image, and the computation of outlines and silhouettes. In particular, silhouette drawings allow artistic expression even with a very limited number of strokes. One can also emphasize specific parts of an image, by using stylized lines, in order to focus the viewer's attention [ST90, SPR$^+$94].

However, a silhouette can also be regarded as an essential shape representation in 2D. In contrast to the previously discussed ESRs, silhouettes are dynamic and depend on how an object is viewed. In addition, not all parts of the silhouette are visible at all times. This means that there are other requirements for algorithms that compute silhouettes. Most notably among these are a fast detection of possible silhouette edges and the visibility culling that is necessary for most applications, in particular those for interactive line rendering.

After a definition of the concept of silhouettes and a more thorough discussion of silhouettes as ESRs, this chapter first gives a comprehensive survey of silhouette detection algorithms followed by the discussion of a new algorithm for efficient visibility culling of silhouettes. This is complemented by methods for generating long smooth silhouette strokes and for post-processing them. Combining a fast silhouette detection algorithm with the new silhouette visibility culling yields a method of generating interactive line renderings, one of the most popular applications for silhouettes.

6.1 Definition of Silhouette Concept

The *silhouette* S of a free-form object is typically defined as the set of all points on its surface where the surface normal is perpendicular to the vector from the viewpoint

[HZ00]. Mathematically, this means that the dot product of the normal $\overrightarrow{n_i}$ with the view vector at a surface vertex P's position p_i is zero:

$$S = \{P : 0 = \overrightarrow{n_i} \cdot \overrightarrow{(p_i - c)}\} \tag{6.1}$$

with c being the position of the center of projection. In case of orthographic projection $\overrightarrow{(p_i - c)}$ is exchanged with the view direction vector $\overrightarrow{v}$.

Unfortunately, for polygonal objects this definition cannot be applied because normals are only well defined for polygons and not for arbitrary vertices on the surface. Hence, typically there are no points on the surface where the normal is perpendicular to the viewing direction. However, silhouettes can be found along those edges in a polygonal model that lie on the border between changes of visibility for the surface. Thus, the *silhouette edges* of a polygonal model are defined as edges in the mesh that share a front-facing polygon and a back-facing polygon.

Some authors refer to the term *contour* rather than the term silhouette. A contour is taken to be the subset of the silhouette that separates the object from the background. Note, that the subset of the silhouette that—in 2D—divides one portion of the object from another one of the same object is not part of the contour. Those lines are sometimes called *internal silhouettes.*

Other lines that are also significant in the context of silhouettes are *creases*, *border lines*, and *self-intersection lines.* Creases are edges that should always be drawn, for instance the edges of a cube. Typically, they are identified by comparing the angle between its two adjacent polygons with a certain threshold. Border lines only appear in models where the mesh is not closed and are those edges with only one adjacent polygon. Lastly, self-intersection lines are lines where two polygons of a model intersect. They are not necessarily part of the model's edges but are important for shape recognition. In the literature, these three additional categories of lines are often called *feature lines.* Some silhouette detection methods make no algorithmic distinction between these feature lines and silhouette lines. This is because the visibility of all these line types must be determined for generating an image. Hence, the visibility aspect of an algorithm usually treats them the same way. The visibility test yields *visible segments* of silhouette edges and potentially also feature lines. Sometimes visible segments are joined together into *visible strokes.*

6.2 Silhouettes as ESRs in 2D

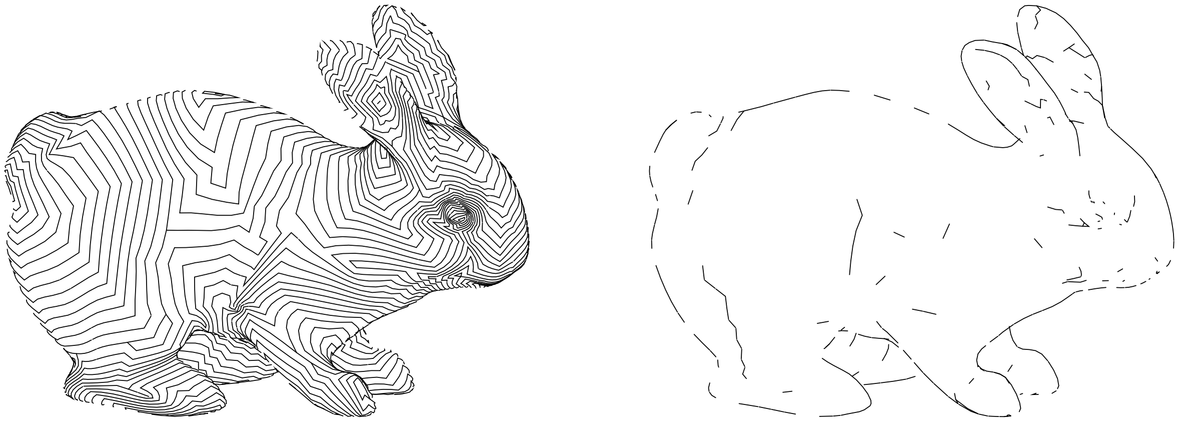
Psychological research has shown that silhouettes play an important role for the recognition of shapes since they are one of the main cues for figure-ground segregation [Gol02, page 158 f.]. In fact, they have proven to be almost as efficient in human shape recognition as regularly shaded objects [BR94]. Moreover, when they are displayed in addition to regularly shaded objects they also significantly improve the figure-ground segregation and, thus, the recognition of shapes [HMH⁺03]. This means that they carry essential information concerning the shape of objects.

From a geometric point of view, KOENDERINK discusses what conclusions can be drawn from the shape of the contour[1] of an object concerning the local shape of the surface of the object [Koe84]. He shows that a convex part of the visible contour corresponds to a locally convex surface of the object. Concavities of the contour, in contrast, correspond to locally saddle-shaped patches of the surfaces of the object.

With respect to the concept presented in this dissertation, it can be concluded that silhouettes are in fact essential shape representations as defined in Section 6.1 and they can be generated by employing the external skeleton algorithm discussed in Chapter 5. This can easily be demonstrated by using a DoI function similar to the one for silhouette based adaptive subdivision (Equation 4.32). This DoI function is modified in order to use the interpolated normal of a vertex instead of the normal of a face (i. e., the equation is derived from Equation 6.1):

$$DoI(V) = \overrightarrow{n} \cdot \overrightarrow{v} \,. \tag{6.2}$$

Using this DoI function in the wave propagation algorithm will generate local maxima at the silhouette of the mesh (see example in Figure 6.1). Consequently, the extracted external skeleton will be the silhouette of the mesh. However, there are additional edges extracted which are not part of the silhouette. This is caused by restricted local maxima which are detected on the wavefronts.[2] In addition, the errors in the silhouette are due to the unbalanced tessellation.



(a) Wavefronts. (b) Silhouette skeleton.

Figure 6.1: Computing the silhouette using the wave propagation based external skeleton extraction algorithm and the DoI function from Equation 6.2.

---

[1] KOENDERINK uses a different notion in his paper: what has been defined as a silhouette in this work he calls a *rim* whereas his notion of *contour* refers to the union of all rays originating in the viewpoint and touching the object perpendicularly in a point on the rim (this forms a *generalized cone*).

[2] They may be considered to be feature lines. However, the computation of these edges is highly unstable. Thus, this is not an appropriate way to deal with such lines.

Thus, silhouettes are a special case of the essential shape representation concept. When comparing them to the group of previously discussed examples of ESRs for feature detection, it can be noted that they have many properties in common. For example, in both cases curves are computed in 3D on the surface of the polygonal mesh of objects. The silhouette curves on the surface of the objects mark the border between visible and invisible parts of the surface while the feature detection ESRs denote interesting curves on the surface, e. g., in terms of the geometry. Both properties can be detected by a specific DoI function.

However, silhouettes and previously discussed ESRs also differ in a number of points. Most previous external skeleton curves are static since they are based on static DoI functions (with the exception of the silhouette based DoI functions discussed in Section 4.6.1). In contrast to the skeletons, silhouettes are view-dependent and, thus, dynamic. This means that they need to be recomputed when the viewpoint changes, e. g., for every frame of an animation. Related to this point, while previous ESRs can be computed without defining a view on them, the silhouette curves do not make much sense unless they are projected onto the two-dimensional viewing plane. When projected onto this plane, the silhouettes border the projected object from the background. When this background is part of the object itself, the curve is an *internal silhouette*. Otherwise it is called an *external silhouette*.

Both groups of ESRs describe abstractions of objects. The previous external skeleton uses a DoI function describing a specific property to abstract from the 3D shape of the object as shown in Chapter 5. In contrast, silhouettes abstract from the projected shape of objects. Interestingly enough, the silhouettes can be used to reconstruct the form of objects in terms of normal data of the surface as shown by JOHNSTON [Joh02]. This is used by JOHNSTON to simulate real-world illumination situations for hand-drawn silhouette cartoon characters placed into filmed movie sequences.

The previously mentioned dynamic character of silhouettes brings about that algorithms for detecting the silhouette of an object have to fulfill different requirements than those for static essential shape representations. In particular, it is essential to use an efficient and fast method to detect silhouettes which is non-trivial to derive.[3] In addition, it is necessary to distinguish between silhouette edges and feature lines.[4] Indeed, there is a variety of different algorithms that compute silhouettes for geometric objects (and potentially feature lines). A classification of the various techniques and how they differ from each other is shown in Section 6.3.

Parts of the silhouette may be occluded while the remaining parts are visible. Again, this depends on how the object is seen by a viewer and changes when the viewpoint moves. In most applications it is necessary to determine the visible subset of the

---

[3] For a discussion of the computational bounds for silhouette computation see EFRAT et al. [EGHHZ00]. However, they impose tight restrictions on the objects they consider (only convex polytopes) and the movement of the viewpoint (only a straight line or an algebraic curve). This means that they do not have to be detected dynamically.

[4] Feature lines, that may be extracted as well, have to be identified by a stable algorithm. They should be the same for every view of the object and must not change their position on the surface.

silhouette. Most notably are silhouette line renderers that treat the visible subset of the silhouette differently from the invisible part (e. g., the visible part is rendered using solid lines while the invisible part is rendered dashed or left out altogether). Section 6.4 discusses a few approaches for visibility culling of silhouettes and introduces a new algorithm that combines the advantages of previous approaches: speed and analytic presentation of the results.

6.3 Approaches for Detecting the Silhouette of Polygonal Meshes

In order to support the following silhouette visibility culling approach, a comprehensive survey of the existing literature for silhouette extraction was conducted [IFH$^+$03]. This survey has shown that the are numerous algorithms for silhouette detection but relatively few works for the necessary visibility culling.

Due to the scope of this analysis, the following discussion of related work on algorithms for silhouette detection is restricted to the discussion of only those algorithms that can be applied to polygonal models.[5] Thus, all algorithms that are discussed here can be used to take a polygonal mesh as input and produce the visible part of the silhouette. Some algorithms, however, can also be used to compute the silhouette only, i. e., without additional visibility culling. The representation of the silhouette may vary depending on the class of the algorithm, it may be in the form of a pixel matrix or as a set of analytic stroke descriptions.

As noted above, every silhouette algorithm has to solve two major problems. First, silhouette edges need to be detected. The second step is to determine the visible subset of the detected silhouette edges (visibility culling). For the purpose of this survey, the approaches are classified according to how they solve each of these problems. First, with respect to how the edge detection problem is solved it can be distinguished between image-space algorithms, hybrid algorithms, and object-space algorithms. Image-space algorithms only operate on image buffers (see Section 6.3.1), hybrid algorithms perform manipulations in object-space but yield the silhouette in an image buffer (see Section 6.3.2), and object-space algorithms do all calculations in object-space with the resulting silhouette represented by an analytic description of silhouette edges (see Section 6.3.3). The second problem—visibility determination—is inherently solved within the algorithm for both image-space and hybrid approaches of silhouette detection. However, for object-space approaches, the visibility culling has to be done separately. The algorithms that do this visibility check can be categorized into image-space, object-space, and hybrid approaches. A discussion of these techniques as well as a newly developed method will be presented in Section 6.4.

---

[5] For an algorithm to compute the silhouette for free-form surfaces see, for example, [EC90].

6.3.1 Image-Space Algorithms

The main idea behind image-space algorithms is to exploit discontinuities in the image buffer(s) that result from conventional rendering. Thus, these methods use image processing algorithms to extract these discontinuities. The result of these methods is a silhouette represented as features in a pixel matrix.

The easiest way to find significant lines would be to detect edges in the color buffer. This, however, does not necessarily detect silhouettes since changes in shading and texturing can cause edges to be detected erroneously. Saito and Takahashi suggest using the z-buffer instead and to apply an edge detection operator such as the Sobel operator to it [ST90]. This has the advantage that only object-relevant edges are found, such as silhouette lines including contours, because in most cases at places where silhouette lines are in the image there is a (C^0) discontinuity in the z-buffer (see Figure 6.2). In [Her99] Hertzmann extends this method by using a normal buffer instead. This can also detect C^1 discontinuities. Both approaches combined yield a pleasing result (see Figure 6.3).

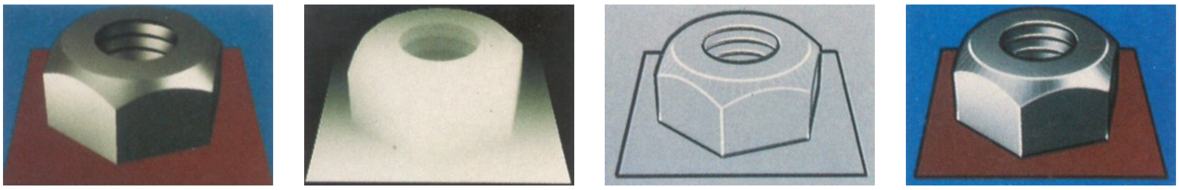


Figure 6.2: Image-space silhouette detection based on edge detection operators on the z-buffer (original image, z-buffer, detected edges, and composed image). From [ST90].

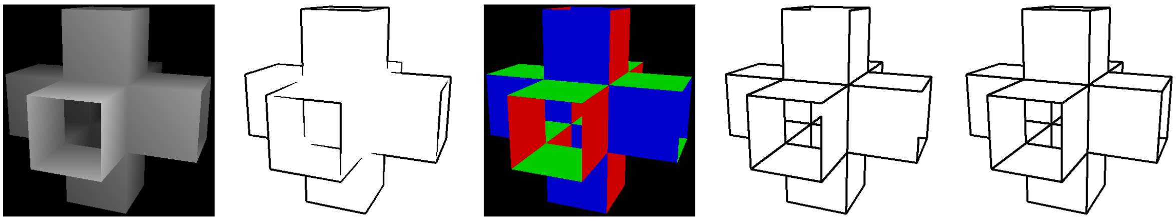


Figure 6.3: Image-space silhouette detection using edge detection operators on the z-buffer and the normal buffer (z-buffer, edges extracted from the z-buffer, normal buffer, edges extracted from the normal buffer, and combination of both edge buffers). From [Her99].

Deussen and Strothotte use a simplified version of this algorithm specifically to compute pen-and-ink illustrations of trees [DS00]. They render the foliage of the tree as primitives (such as oriented disks) only to the z-buffer and look for discontinuities larger than a specified threshold. Depending on the primitives' size and the depth threshold, this achieves the special look of pen-and-ink.

The advantage of using image-space algorithms is that existing graphics hardware and rendering packages can be used to generate the buffers on which the algorithms operate. This makes these algorithms relatively simple to implement. The computational complexity of silhouette detection in this manner depends on the number of pixels that comprise the image, rather than the number of polygons in the model,[6] and thus is constant provided that the image resolution remains constant. Thus, these algorithms are usually fast. Moreover, the algorithms can be greatly accelerated by appropriate graphics hardware. Mitchell suggests using the *pixel shader* technique on newer graphics cards for hardware acceleration of image-space silhouette detection [MBC02]. Finally, an image-space algorithm inherently deals with silhouette edges the same way as with feature lines. In contrast to other methods, even self-intersections are found automatically. These also share similarities with z-buffer rendering in that the algorithms are very generic and robust to errors in the models.

The main disadvantage of image-space algorithms is that the user has very little control over the attributes of the resulting lines. The only ways to directly influence the resulting lines are the choice of an edge detection operator and the choice of a source buffer on which to apply it. A second disadvantage is the fact that the silhouette is not available in the form of analytic line description. Hence, the silhouettes can typically not be stylized or used for further processing. Thus, many techniques simulating traditional drawing or painting utensils cannot be readily applied because they usually require this analytic information. On the other hand, there are approaches that extract curves from the silhouette pixels such as presented by Loviscach who fits Bézier curves to the pixels [Lov02]. This does allow for subsequent stylization of the resulting curves. However, the process of fitting curves to the pixel silhouettes might introduce new artifacts and inaccuracies. In addition, the approach is too slow for interactive or real-time applications.

Inherent in the use of image processing operators is the fact that the resulting silhouettes in general do not have distinct borders. The intensity—the gray value—of a silhouette pixel usually depends on the derivative and thus on the intensity of features detected in the original buffer. However, this also means that the resulting images tend not to have significant aliasing problems. Another characteristic of these algorithms is that they are limited to pixel precision. This means that some fine details with less than the size of a pixel may be hidden. However, for visual appearance this usually makes no difference.

6.3.2 Hybrid Algorithms

Hybrid algorithms are characterized by operations in object-space that are followed by the rendering of the modified polygons in a more or less ordinary way using the z-buffer. This usually requires, however, two or more rendering passes. The result is similar to image-space algorithms, in that the silhouette is represented in a pixel matrix.

Rustagi presents a simple algorithm using the stencil buffer [Rus89] that delivers the contour—not the complete silhouette—of an object. The object's mesh is rendered

---

[6] Note, that whereas silhouettes are not computed from polygons, these polygons must be rendered first into the required buffers.

four times, each time translated in screen coordinates by one pixel in the positive or negative x or y directions. During each pass the stencil buffer is incremented where the object fills the viewport. After these four passes, the object's interior pixels will have a stencil value of four, the perimeter pixels will have values of two or three, and the exterior will have values of zero or one. Finally, setting the stencil function to pass if the stencil value is two or three and rendering a primitive larger than the object will result in only the outline pixels being changed.

Rossignac and Emmerik present a method based on z-buffer rendering that yields silhouettes and not only contours [RvE92]. They show four algorithms that differ slightly from each other and that render polygonal objects either in wireframe or silhouette mode with the hidden lines either removed or dashed. For generating an image with visible silhouette and feature lines only, they first draw the faces of the objects into the z-buffer. Then they translate the mesh away from the viewer by a small distance and render a wireframe representation using wide lines. This ensures that only silhouette edges will be visible since the other lines will be occluded by the previously rendered faces in the z-buffer. Finally, they translate the object forward again by twice the former distance and render the feature lines of the objects in regular line width.

Raskar and Cohen generalize this approach in their work [RC99]. For automatically determining front and back facing polygons that define the silhouette they use traditional z-buffering along with back-face culling or front-face culling, respectively. Similar to the work by Rossignac and Emmerik, by first rendering all polygons in white on a white background with back-face culling enabled, they fill the z-buffer with the depth data of front-facing polygons. Afterwards, they render all polygons again, but this time in the desired silhouette color and using front-face culling. Hence, only the back-facing polygons effect the frame buffer during this second pass. By employing the "equal to" depth function, Raskar and Cohen achieve the result that only the edges where the two groups of polygons meet are drawn and thus yield the silhouette of the model. Similar results are achieved by instead rendering the wireframe representation in the second pass. This allows rendering of silhouettes with a certain line width by just using thicker lines for the wireframe rendering.

This general technique can be improved by performing additional transformations before rendering the back-facing polygons. A translation of the back-facing polygons towards the viewer yields thicker silhouette lines in the resulting image. However, these lines do not have a constant line width. For silhouettes with adjustable but constant width, the back-facing polygons are enlarged depending on the distance from the viewer and on the angle with the viewing direction (see Figure 6.4).

A similar technique that also uses hardware accelerated rendering is presented by Gooch et al. [GSG$^+$99]. Instead of rendering directly to a color buffer, lines are drawn into a stencil buffer. This stencil buffer is used as a mask for drawing the back-facing polygons in a second pass. Also, creases can be rendered in a different color than silhouettes.

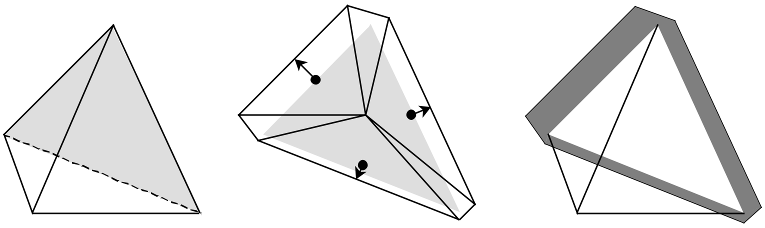


Figure 6.4: Enlargement of a back-facing polygon to achieve wide silhouette lines. From [RC99].

Raskar proposes a one-pass hardware implementation that basically adds borders around each triangle [Ras01]. These borders are arranged in such a way that they disappear between two neighboring front-facing polygons during rendering. This method also inherently includes the generation of crease lines where the angle between the faces exceeds a given threshold.

A different technique introduced by Gooch et al. uses environment mapping in addition to regular shading [GSG+99]. By using a partially darkened environment map, the vertices with normals that are almost perpendicular to the viewing direction are assigned dark values whereas other vertices are not changed. This method achieves a stylistic effect (see Figure 6.5).

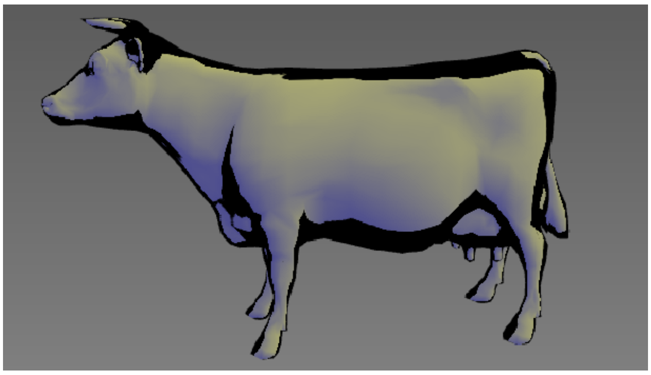


Figure 6.5: Silhouette generated with a hybrid algorithm. From [GSG+99].

The advantage of hybrid algorithms over image-space algorithms is the typically higher degree of control over the outcome. This control is achieved through the choice of an algorithm and through parameters such as translation or enlargement factors that control line width. The visual appearance of the generated images tends to be a more stylistic one. Also, in contrast to image-space algorithms, the silhouettes inherently have distinct borders which may be desired. On the other hand, the distinct borders can give rise to aliasing artifacts in the image. However, these artifacts can be avoided by employing well-known anti-aliasing techniques.

The computation times of hybrid algorithms, generally, do not differ much from those of image-space methods and are in the realm of interactive to real-time frame rates.

Some need two or more render passes but, in return, do not require an additional manipulation of any of the generated buffers. Additional computation time besides the number of rendering passes is required by the object-space manipulations that are needed for some of the algorithms. However, the algorithms can easily make use of commonly available graphics hardware to speed up the rendering process.

The drawbacks of a pixel matrix representation of the found silhouette lines, as mentioned for image-space algorithms (see Section 6.3.1), apply to hybrid algorithms just as well. Similarly, the silhouette lines have pixel resolution and do not facilitate further stylization. In addition, the algorithm may have some numerical problems due to limited z-buffer resolution.

6.3.3 Object-Space Algorithms

In the above discussions of algorithms the disadvantage of a pixel matrix representation of the computed silhouette was mentioned. In order to do further processing of the lines it is necessary to have an analytic representation of the silhouette. A good way to achieve this is to employ an object-space algorithm. The computation of silhouettes in this group of algorithms, as the name suggests, takes place entirely in object-space. In contrast to hybrid and image-space algorithms, object-space algorithms deal with the problems of edge detection and edge visibility in separate stages.

A straightforward way to determine the silhouette edges of a model follows directly from the definition of a silhouette (see Section 6.3.3). There are approaches to speed-up the process by using some pre-computed data structures (see Section 6.3.3), while other algorithms achieve faster execution by employing stochastic methods (see Section 6.3.3).

Trivial Method

The trivial algorithm is based on the definition of a silhouette edge. It consists of two basic steps. First, it looks at all polygons of the mesh and determines whether they are front-facing or back-facing as seen from the camera. The second step of the algorithm examines all edges of the model and selects only those that share exactly one front-facing and one back-facing polygon. These two steps have to be completed for every frame.

Buchanan and Sousa suggest using a data structure called the *edge buffer* to support this process[7] [BS00]. In this data structure they store two additional bits per edge, F and B for front-facing and back-facing, respectively. When going through the polygons and determining whether they are front-facing or back-facing, they XOR the respective bits of the polygon's adjacent edges. In case an edge is adjacent to one front-facing

---

[7] The edge buffer is especially intended to optimize for platforms such as game consoles that have certain hardware restrictions.

and one back-facing polygon the FB bits are 11 after going through all polygons and can thus be used for silhouette rendering. Buchanan and Sousa extend this idea to include support for border edges and other feature edges.

This simple algorithm, with or without using the *edge buffer* data structure, works for both perspective and orthographic projections. Also, it is guaranteed to find all silhouette edges in the model. It is easy to implement and well suited for applications that only use small models. However, it is comparatively computationally expensive for common hardware-based rendering systems and the models of the size typically used with them.[8] Even if an effective data structure is used that delivers local connectivity information in constant time, every face has to be looked at, its orientation has to be determined (which can be done with one dot product per face[9]), and every edge has to be looked at as well. The time complexity of this trivial algorithm is linear in the number of edges and faces but still too slow for interactive rendering of reasonably sized models. When it is additionally considered that there is typically only a small number of edges that are in fact silhouette edges, testing each one also seems unnecessary.

To speed up the brute force algorithm, user-programmable vertex processing hardware can be used as suggested by Card and Mitchell [CM02]. For every potential silhouette edge in the model a separate quad is generated along the edge. In addition, each vertex of such an edge stores the normals of both faces adjacent to the edge. When rendering the quads, a vertex program tests whether the normals are front-facing or back-facing. The quad is only actually drawn where the result is different for both normals.

Sub-Polygon Silhouettes

Because a polygonal mesh is usually only an approximation of a free-form object, silhouettes of polygonal meshes usually have some artifacts (such as, for example, zig-zags or silhouette edge clusters). Hence, the expected silhouette of the real object can differ significantly from the silhouette the trivial algorithm yields. Therefore, besides the quest for a fast algorithm, there are approaches that try to find a more exact silhouette similar to the one of the real object.

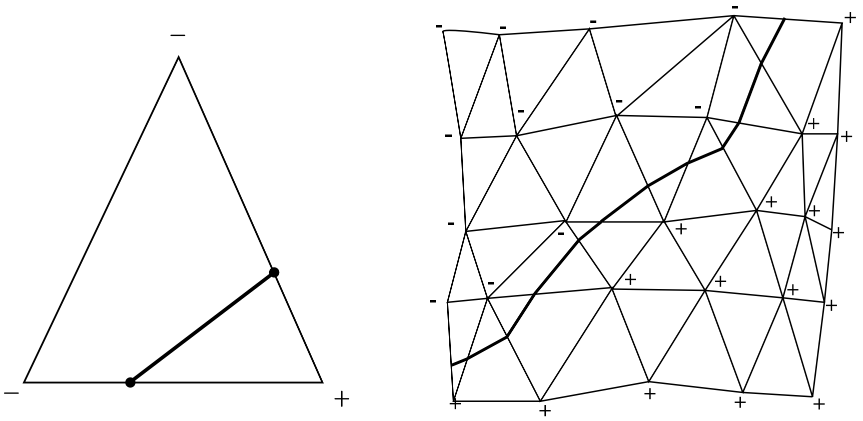
Hertzmann and Zorin suggest considering the silhouette of a free-form surface that is approximated by a polygonal mesh [HZ00]. To find this silhouette, the normal vectors of the approximated free-form surface are re-computed in the vertices of the polygonal mesh. Using this normal its dot product with the respective viewing direction is computed. Then, for every edge where the sign of the dot product is different at both vertices, linear interpolation along this edge is used to find the point where it is zero. These points are connected to yield a piecewise linear sub-polygon silhouette line (see Figure 6.6). The resulting silhouette line is likely to have far less artifacts

---

[8] For software-based rendering systems, however, the silhouettes computed with the brute-force approach come at little extra cost.

[9] For perspective projection the view vector has to be re-computed for every face as well.

in the form of zig-zags or silhouette edge clusters. Also, the result is much closer to the "real" silhouette than the result of a traditional polygonal silhouette extraction method. Hence, it is well suited for later stylization of the lines. For an example see Figure 6.7.



(a) Sub-polygon silhouette for a single triangle.

(b) Concatenated sub-polygon silhouette for a mesh.

Figure 6.6: Computation of a sub-polygon silhouette for a single triangle and a mesh. The dot product between normal vector and view direction is positive at vertices with plus signs and negative at vertices with minus signs. Between a positive and a negative vertex linear interpolation is used to find the silhouette. From [Her99].

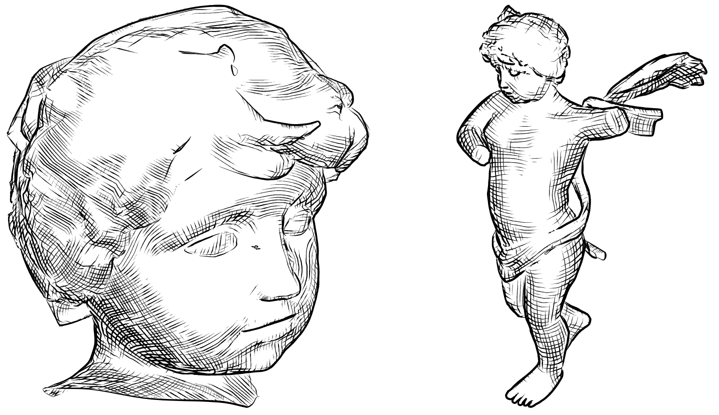


Figure 6.7: Two images generated with the sub-polygon method by Hertzmann and Zorin. The silhouettes are combined with a hatching technique. From [HZ00].

Pre-Computation Methods

To speed up the process of silhouette edge determination, various authors have developed methods that do some kind of pre-processing. The pre-processing stage sets up data structures that are used to find silhouette edges more quickly.

Gooch et al. [GSG⁺99] as well as Benichou and Elber [BE99] present a procedure for pre-processing that is based on the projection of face normals onto a Gaussian sphere. Here, every edge of the mesh corresponds to an arc on the Gaussian sphere which connects the normals projections of its two adjacent polygons (see Figure 6.8). For orthographic projection, a view of the scene is equivalent to a plane through the origin of the Gaussian sphere. The authors further observe that every arc that is intersected by this plane is a silhouette edge in the corresponding view. Applying this observation to silhouette edge extraction removes the need to check if every face per frame is front-facing or back-facing. The arcs are computed in a pre-processing step and at run-time only the intersections with the view-plane are tested. An example rendering is shown in Figure 6.9.

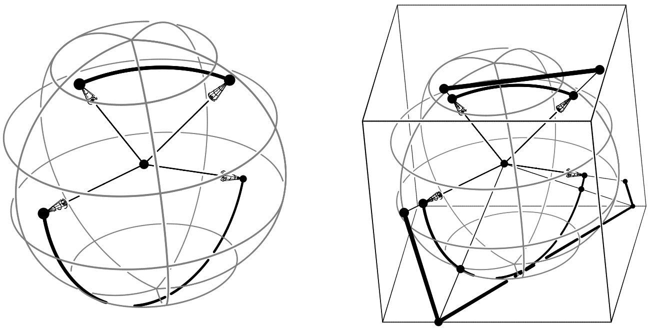


Figure 6.8: Pre-processing using the projection of the normals of the two faces adjacent to an edge onto the Gaussian sphere (left) and its surrounding cube (right). From [BE99].

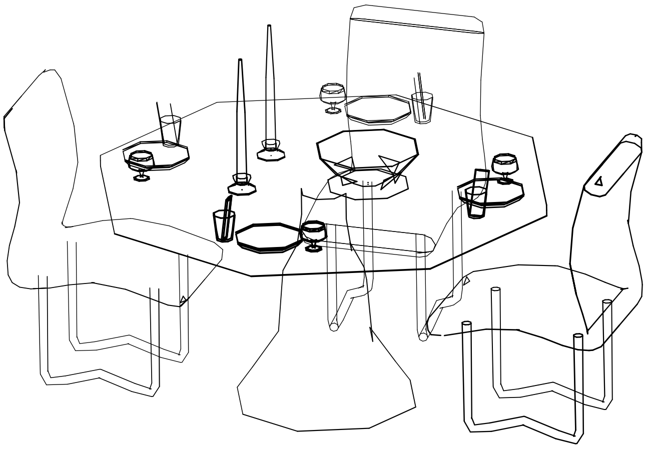


Figure 6.9: Example scene rendered with the Gaussian sphere pre-processing algorithm. Note that no visibility test was used. Courtesy of Gershon Elber and Fabien Benichou.

To further limit the number of arcs to be tested, Gooch et al. use a hierarchical decomposition of the sphere, starting with a platonic solid, and consecutively applying subdivision to its sides [GSG⁺99]. The arcs are stored in the lowest possible level. Benichou and Elber, on the other hand, map the Gaussian sphere as well as the arcs to a cube surrounding the sphere [BE99]. The arcs are equivalent to line segments on the cube. The great arc of the view plane is also mapped onto the cube resulting in a set of line segments which simplifies the intersection test. To reduce the number of arcs that have to be tested the sides of the cube are decomposed into a grid and only edge line segments in grid cells that are touched by a view-plane line segment are tested for intersection.

In [HZ00], HERTZMANN and ZORIN present a method that uses a data structure also based on a dual representation which is, in fact, similar in principle to the one by BENICHOU and ELBER. In this approach, however, a dual representation of the mesh is constructed in 4D space based on the position and tangent planes of every vertex. The viewpoint's dual (a plane in 4D) is intersected with the dual of the mesh triangles. Beforehand, the dual vertices are normalized using the L_∞-norm so that they end up on one of the unit hypercube's sides.[10] Thus, the problem is reduced to intersecting the triangles on a hypercube's sides with the viewpoint's dual plane. This means that triangles in eight 3D unit cubes (the eight hypercube sides) must be intersected with a plane. Here, again, the speed-up is achieved by employing space partitioning, namely by using an octree for each of the hypercube sides. At runtime, only the dual plane of the viewpoint is computed and intersected with each of the hypercube's sides resulting in edges that intersect the silhouette. The major advantage over the methods discussed earlier in this section is that it works for orthographic as well as perspective projections. For orthographic projection the viewpoint is simply placed at infinity.

POP et al. present another algorithm that is based on a dual representation [PBD$^+$01]. Similar to the previous approach but this time in three dimensions, a dual space is constructed with vertices having planes as duals and vice versa. The silhouette problem is reduced to finding intersections of the dual plane of the viewpoint with the duals of the edges of the mesh. This, unfortunately, is still expensive to compute for every frame. The authors note, however, that only the changes of silhouettes have to be found in two consecutive frames. To do this they look at the wedge that is formed by the dual planes of two consecutive viewpoints and identify the dual edges with one vertex inside and the other vertex outside of this wedge. Similarly to HERTZMANN and ZORIN's method, to speed up this process they use an octree data structure. The advantage of this approach over the previous one is that it works in 3D only. Hence, the search for intersections can be performed only once per frame as opposed to eight times. Also, only the changes of the silhouette are detected. However, although HERTZMANN and ZORIN's approach requires eight octrees instead of just one each octree only contains about one-eighth of the faces. Also, both methods should have an expected extraction time linear on the number of found silhouette edges.

SANDER et al. use a different method for silhouette edge detection [SGG$^+$00]. In their approach, a hierarchical search tree is constructed in which the mesh's edges are stored. All the faces attached to the edges in a node and its associated sub-tree make up a *face cluster*. At run-time, the tree is searched recursively to find face clusters which

---

[10] The normalization does not make a difference because the viewpoint's dual plane goes though the origin.

are entirely front-facing or entirely back-facing. All the edges of such a face cluster can be discarded and the search for the associated sub-tree is stopped. To effectively test whether a face cluster is entirely front-facing or entirely back-facing in constant time, Sander et al. store two anchored cones in every node (see Figure 6.10). One of the cones represents those positions where the view point can be located at for the entire face cluster to be front-facing, the other cone for the face cluster to be back-facing.

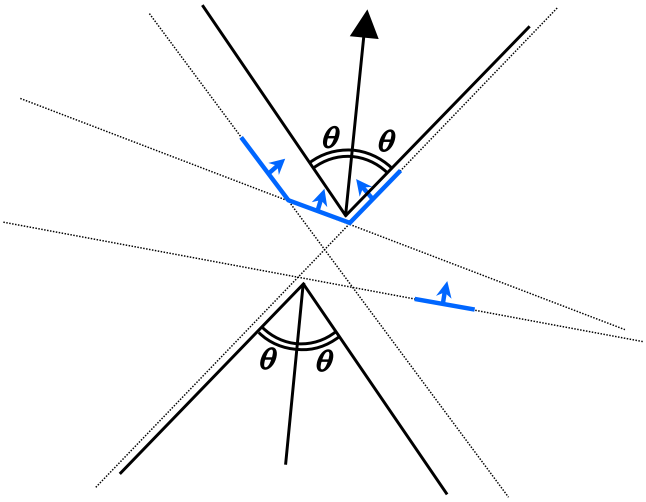


Figure 6.10: Pre-processing for faster silhouette detection through arrangement of face clusters in anchored cones (visualized in 2D) by Sander et al. From [SGG+00].

In summary, all pre-computation methods presented here reduce the number of triangles or edges that have to be checked at run-time to speed-up the silhouette detection process without having to trade in accuracy. This is accomplished by establishing an efficient data structure during pre-processing. All authors claim to achieve at least interactive frame rates for reasonably sized models. However, these methods have the drawback that they make animation of the models themselves inefficient since the pre-computed data structure stores information about the visibility of polygons to quickly identify silhouettes for the moving viewpoint. If a model would be animated beyond a flight through the scene this pre-computed data structure would become invalid which would result in new pre-computation steps for every frame. On current hardware, this would reduce the frame rate to below interactive rates. In addition, pre-computation algorithms need a separate elaborate data structure besides the actual geometry for achieving their speed-ups.

Stochastic Method

In contrast to pre-computation, Markosian et al. suggest using a stochastic algorithm to gain faster run-time execution of silhouette detection [MKT+97, Mar00]. They observe that only a few edges in a polygonal model are actually silhouette edges.[11]

---

[11] Typically $O(\sqrt{n})$ edges of the n polygons according to [SGG+00].

In the hope of finding a good initial set of candidates for front and back face culling, they randomly select a small fraction of the edges and exploit spatial coherence. Once they detected a silhouette edge, they recursively test adjacent edges until they reach the end of the silhouette line. In addition, they also exploit spatial coherence as the silhouette in one frame is typically not far from the—visually—same silhouette in the next frame.

Combining these two parts of the algorithm yields most of the silhouette edges in one image. By exploiting spatial and temporal coherence Markosian et al. achieve fast run-time execution for interactive or real-time applications. Also, the method is not restricted to static objects, so animation does not pose a problem. However, in contrast to the pre-computation algorithms that were discussed above, the algorithm cannot guarantee finding the entire set of silhouette edges for a certain view on the scene.

6.3.4 Comparison of Silhouette Detection Algorithms

Table 6.1 summarizes the silhouette detection algorithms discussed in the survey. In addition to the individual assessments given above, it compares the algorithms with respect to a number of important properties.

algorithm	image-space/hybrid/object-space	silhouette output as pixels or edges	visibility culling performed	creases and intersections detected	accuracy of the silhouettes	silhouette stylizable	uses preprocessing
[ST90, Her99]	image	pixels	yes	yes	pixel	no	no
[DS00]	image	pixels	yes	no	pixel	no	no
[Rus89]	hybrid	pixels	yes	no	pixel	no	no
[RvE92]	hybrid	pixels	yes	no	pixel	limited	no
[RC99, Ras01]	hybrid	pixels	yes	no	pixel	limited	no
[GSG$^+$99]	hybrid	pixels	yes	no	pixel	limited	no
trivial, [BS00]	object	edges	no	no	mesh	yes	no
[HZ00]	object	edges	no	no	submesh	yes	yes
[GSG$^+$99]	object	edges	no	no	mesh	yes	yes
[BE99]	object	edges	no	no	mesh	yes	yes
[PBD$^+$01]	object	edges	no	no	mesh	yes	yes
[SGG$^+$00]	object	edges	no	no	mesh	yes	yes
[MKT$^+$97, Mar00]	object	edges	no	no	mesh	yes	no

Table 6.1: Comparison of silhouette detection algorithms.

It is of particular interest to use the detected silhouettes beyond the mere drawing of simple lines in an image. Therefore, in order to successfully do a further processing of silhouettes, e. g., apply line styles across silhouette lines, the silhouettes of a scene must be represented as a series of connected visible line segments. This requires *silhouette edges* as the output of the silhouette detection process as a prerequisite. Thus, from the three groups of algorithms for silhouette detection discussed above only object-space methods can be used for this purpose. All object-space silhouette detection methods, however, have in common that they only test the silhouette property of the edges in a mesh. Image-space and hybrid silhouette algorithms, in contrast, inherently also test the visibility of the silhouettes. Therefore, the following section discusses visibility culling for edges that were generated with object-space silhouette detection methods.

6.4 From Silhouette Edges to Visible Strokes

In most applications it is essential to handle the visible subset of a silhouette (that part of the silhouette which is not occluded by other parts of the scene) differently from the invisible subset. Hence, it is essential to come up with ways to compute the visible silhouette. For example, the importance of determining visible line segments becomes apparent when considering a line renderer that uses line styles which may apply perturbations to the original line. When the lines have been tested for visibility prior to the stylization this allows lines to be rendered to the scene afterwards without additional depth testing. For example, this allows to use perturbations that overlap or cross over other objects (such as thick lines or path variations). In addition, the visible subsets of the silhouette edges have to be assembled to form long strokes. This allows, e. g., a line renderer to simulate the smooth strokes of a drawing tool that an artist may have used. Using single edges in this case would significantly distort a line rendition since the individual edges do not necessarily have all the same length. In addition, for models with fine tessellations the average edge length will be comparatively short.

Methods to do both of these steps—visibility culling and stroke handling—will be discussed in the following (for details see [IHS02]). First, a brief survey of previous techniques for visibility culling of lines will be given (see Section 6.4.1). Afterwards, a new technique is presented that uses a hybrid approach for the line visibility determination combining both the speed of image-space techniques with the analytic representation of the results found in object-space methods (see Section 6.4.2). Finally, a discussion of the creation long smooth strokes follows and how certain artifacts of the polygonal character of the silhouettes can be avoided (see Sections 6.4.3 and 6.4.4).

6.4.1 Visibility Culling of Silhouette Edges

The problem of visibility culling of silhouette edges is the classic computer graphics problem of hidden line removal. Similar to silhouette detection, there are three general ways to attack this problem. A fast way is to use an image-space approach (see Section 6.4.1). High precision can be achieved by employing an object-space method (see Section 6.4.1) that yields visible silhouette segments in an analytic description. Finally, by combining both approaches one can use a hybrid method[12] that is faster than an object-space method but still yields analytic descriptions of the visible silhouette lines (see Section 6.4.1).

Image-Space Visibility Tests

A trivial and fast way to determine the visibility of silhouette edges in image-space is to use the z-buffer. The simplest method is to render the silhouette edges and let the z-buffer do the hidden line removal. The result is comparable to Raskar and Cohen's hybrid method using wireframe rendering in the second pass (see Section 6.3.2). A more advanced algorithm renders the silhouettes in a certain line width and perpendicular to the viewing direction. Unfortunately, this generates images with thick internal silhouettes and thin contour lines because the contour lines are partly occluded by the object itself and internal silhouettes are not.

Besides the limitation to pixel accuracy, the main disadvantage of this image-space approach is that the drawn silhouette line might be partially occluded when stylized variations have been applied to it.

Object-Space Visibility Tests

In the past, many algorithms have been proposed to do hidden line removal in object-space (see, e. g., Sutherland et al. [SSS74] or Sechrest and Greenberg [SG81]), most of which also solve the problem of hidden surface removal. It would be beyond the scope of this survey to address all of these but many of them can easily be used for determining the visibility of silhouettes. One algorithm, however, deserves further attention since it has frequently been used in the context of silhouette algorithms: the visible line algorithm presented by Appel [App67]. The algorithm is based on the notion of *quantitative invisibility* (QI)—the number of front-facing polygons between a point on the edge to be rendered and the viewer—which is determined for all segments of edges (see Figure 6.11). All edge segments with a QI value of zero are visible, all others are invisible. The fact that QI value only changes when the edge to be rendered intersects a silhouette edge in the two-dimensional projection (it can also change at a vertex if the vertex lies on a silhouette edge causing some complications for the computation) allows for propagation along connected sets of edges. Therefore, for every connected set of edges that has to be tested the QI value is initially established

---

[12] Hybrid visibility culling methods must not be confused with hybrid silhouette detection algorithms. Visibility culling is inherently solved within the hybrid silhouette detection while hybrid visibility culling is used with the silhouettes generated by object-space silhouette detection. However, both types of hybrid algorithms were named such because they both combine image-space and object-space aspects.

for one point using, for example, ray-tracing. It is then propagated along the edges to be rendered determining whether the QI value increases or decreases each time the edge passes behind a silhouette edge.

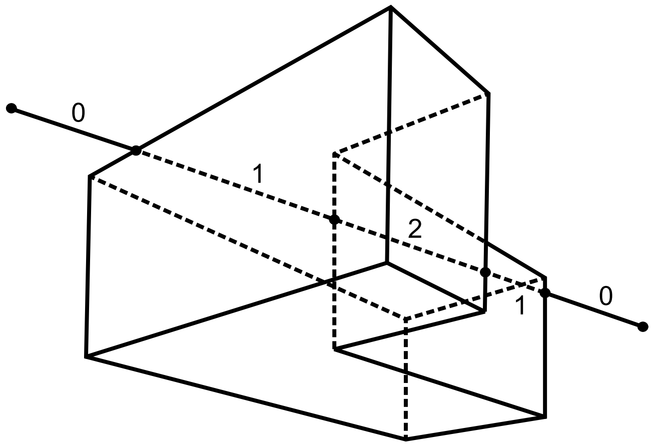


Figure 6.11: Example for Appel's notion of quantitative invisibility (QI) for a line passing behind the object. The dots denote the positions where the QI value changes. Adapted from [App67].

Markosian et al. [MKT^{+}97] use a modified version of Appel's algorithm to improve computation time. They re-define the QI value to be the total number of faces between a point and the viewer. This simplifies Appel's algorithm in that the QI value can now only change at a vertex if that vertex is a so-called *cusp vertex.* In addition, they avoid many of the required ray tests by first looking at how the QI value changes by establishing relative QI values when traversing a connected set of edges before actually executing the ray test. Sometimes, this marks the whole connected set of edges invisible and thus removes the need to establish initial QI values. In remaining cases, the algorithm avoids even more tests by inferring from QI values of one set of connected edges to those of others by using the relative QI values as determined before.

Hertzmann and Zorin apply this approach to their algorithm for sub-polygon silhouettes [HZ00]. First, they divide their silhouette curves into segments at points where the visibility can possibly change and determine visibility individually for each of these segments. Then, in addition to their sub-polygon silhouette edges they also determine regular (non sub-polygon) silhouette edges that are a subset of the input mesh. They determine edge visibility using the adapted Appel's algorithm described above. Finally, the visibility of the majority of the edges adjacent to one of the sub-polygon silhouette edge segments is used to infer the visibility of the segment. For finely tessellated objects this yields a sufficiently correct visible silhouette.

The advantage of analytic algorithms is that they typically yield high precision visibility information. However, since their computational complexity is not constant, they usually take longer to compute than image-space approaches.

Hybrid Visibility Tests

As has been demonstrated above, due to their exact solution for the hidden line problem, object-space silhouette visibility tests are usually expensive. In addition, as has been shown before, a typically fast but not as accurate way of determining the visibility of the silhouette edges is to use an image-space approach. However, in many applications it is not necessary to have more than pixel accuracy. Thus, this fact can be used to achieve a significant speed-up for the visibility test by combining object-space and image-space approaches to a hybrid algorithm.

In addition to regular rendering using a z-buffer, Northrup and Markosian use a second buffer—an ID buffer—to determine the visibility of the silhouette edges [NM00]. In this ID buffer each triangle and each silhouette edge is identified by a unique color. For each frame, this buffer is read from the graphics hardware and all pixels of the reference image are examined to extract all those silhouette edges that are represented by at least one pixel. Each of the remaining silhouette edges are then scan-converted and checked for visibility according to whether a pixel with the edge's unique color exists in the ID buffer or not. An example rendition with stylization applied to the silhouette is shown in Figure 6.12.

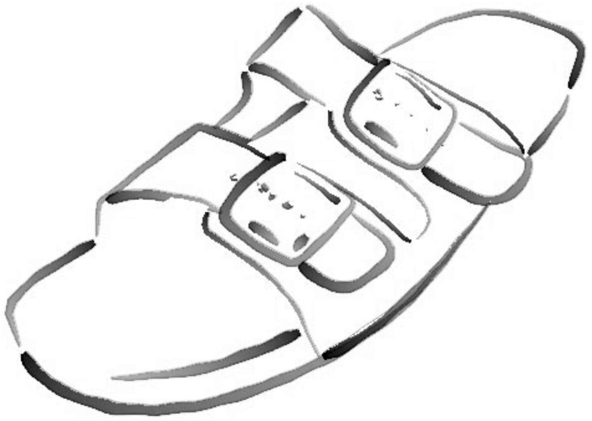


Figure 6.12: Image generated using the hybrid visibility test by Northrup and Markosian. Note the use of line styles for rendering silhouette strokes. From [NM00].

6.4.2 A New Hybrid Approach for Determining Visibility

As already argued above, from the set of silhouette edges it is necessary to determine which segments of those edges are visible to the viewer. In the following, a new technique will be introduced that does not require rendering of an ID-buffer (such as done by Northrup and Markosian [NM00]), but makes direct use of available z-buffer information. Therefore, the scene may be rendered as usual (or enable writing to the z-buffer only), and then this information can be used for visibility testing. Thus, in cases when the scene geometry needs to be rendered in combination with silhouettes, the generation of z-depth information is obtained for free.

To determine visible silhouette segments, each silhouette edge is taken separately, and the edges are scan-converted in relation to the z-buffer, testing for point visibility along the edge. In a naïve approach, to determine the visibility of a point one could just compute its z-value and compare it with the value in the z-buffer. However, the results are highly instable due to z-buffer precision and pixel quantization. Since the visibility of silhouette edges is being tested the visibility culling algorithm has to look at parts of the projected scene where many faces are almost parallel to the viewing direction. Hence, these faces may share the same x-y-location as the silhouette edges, but will overwrite the z-buffer at those locations with z-values closer to the viewer. The naïve test will fail under these circumstances, and falsely detect an edge as occluded (see Figure 6.13).

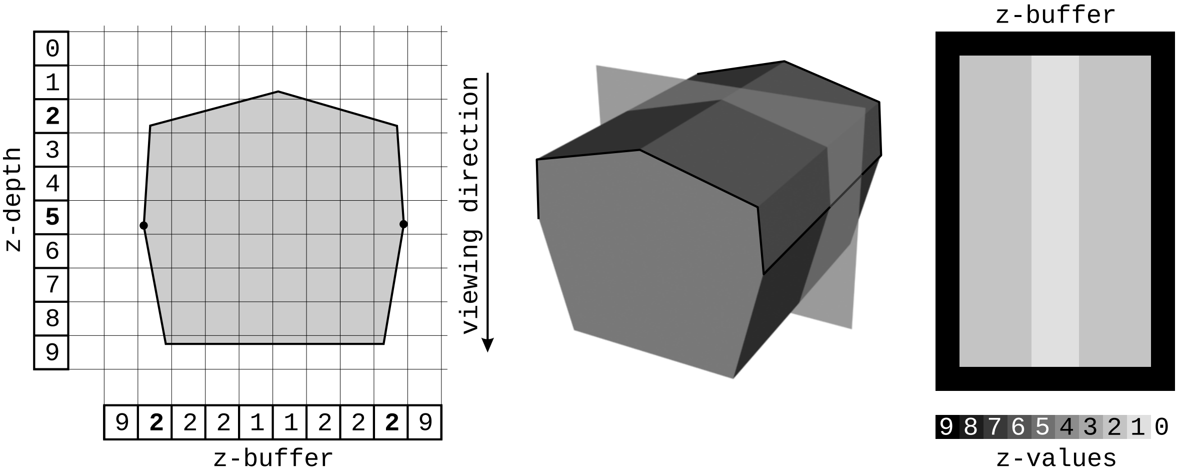


Figure 6.13: Problem with faces almost parallel to the viewing direction (indicated by the arrow). Although the z-value of the silhouettes (indicated by dots in the left image) is 5 the value of the related pixels' z-buffer is 2. The left image shows a cut through the model as shown in the middle (with silhouette lines emphasized) and right part shows the resulting z-buffer image.

In order to take advantage of the availability of the rendered z-buffer, a more reliable point visibility test along a silhouette edge has to be employed. For this purpose, z-buffer silhouette extraction techniques (e. g., Saito and Takahashi [ST90]) are adapted so that not only the exact position of a point is looked at to determine its visibility but also its 8-neighborhood. If there is any point in this 8-neighborhood where the z-buffer value is farther away than the computed z-value of the point, it is declared visible, and not visible otherwise (see Figure 6.14).

This works because silhouette edges occur in regions with a discontinuity in z-depth. Hence, it is almost certain that one of these background pixels is found by looking in the 8-neighborhood. Those silhouette edges that are occluded by larger areas will not

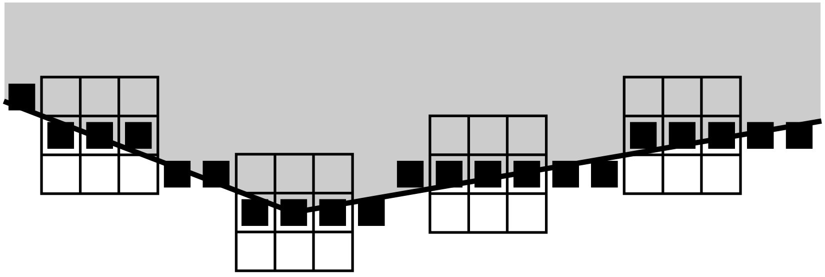


Figure 6.14: Testing the visibility by looking at the z-buffer values of the pixel and its 8-neighborhood. Also, not every pixel is tested (in this case every fifth pixel is examined).

find any pixels further away, and will be correctly identified as occluded. Therefore, the numeric instability of the z-value computation is significantly reduced.

For speeding up the process it is not necessary to test visibility for each pixel when scan converting an edge. Instead, it is possible to parameterize the number of pixels that may be skipped along this process (example in Figure 6.14). The advantage of this is that it is possible to tune a speed vs. accuracy trade-off for the specific needs. There are cases when two adjacent test points are detected in the scan conversion of an edge of which one is visible and the other one invisible. The edge segment determined by these two adjacent points then intersects a boundary between visible and occluded parts of the edge. An imprecise way to solve this problem is to clip this segment in the middle. In order to reduce visible artifacts, the scan-conversion process is subdivided until a one-pixel accuracy is reached for the location where the edge needs to be clipped. However, the consequence of skipping pixels could be that a line is drawn through an occluder that lies in-between two visibility test points in the scan conversion of an edge.

Additional speed-up is obtained through the optimization of the extraction of z-buffer values. This is done both by caching results (eliminating re-accessing the same values) and reading larger chunks at a time (minimizing function-call overhead and bus-rate transfer). The size of the chunks to read is largely dependent on the performance issues of the hardware. On common PC-based machines it has shown to be practicable to read chunks of eight by eight pixels at a time and store them into the main memory. Only in cases where the image exhibits a high density and distribution of silhouette edges it becomes profitable to read the entire area of the z-buffer that is covered by the scene.

As a result the set of clipped silhouette edges is found that is now defined as the set of visible silhouette segments. Next, the visible silhouette segments are combined into a set of silhouette strokes.

6.4.3 Finding Silhouette Strokes

A silhouette stroke is a concatenation of visible silhouette segments that share a common vertex pairwise. Therefore, strokes terminate when there are no more segments that may be joined by a common vertex. There are cases where more than two visible silhouette segments share a common vertex (e. g., a cusp vertex). In these cases an approach can be adopted that continues the stroke along the segment with an angle closest to 180° (as done by Northrup and Markosian [NM00]).

To generate these strokes, a data structure that provides local connectivity information can be used such as the Winged Edge [Bau75]. Starting with an arbitrary visible silhouette segment, and searching in both directions for additional connecting visible silhouette segments, the silhouette edges are concatenated in each iteration. Visible silhouette segments that do not comprise of a whole silhouette edge may either form a silhouette stroke by themselves (if they are located in the middle of a silhouette edge) or may start or end a silhouette stroke (if they are at the beginning or end of one silhouette edge). This is simple and efficient because the visible silhouette segments that were generated in the previous stages of the algorithm correspond directly to the original mesh with the edge connectivity information intact. Thus, no additional processing, such as rendering an edge ID buffer, is necessary to find connectivity between edges.

6.4.4 Removing 2D Artifacts

Since meshes do not provide perfect representations of surfaces and due to numerical instabilities in silhouette edge detection, one often gets overlapping silhouette edges and small zig-zags in image-space (of course, there might also be zig-zags due to the local shape of an object). These have to be removed in order to achieve good looking results when applying styles to strokes, as will be described in the following sections. The visibility-check algorithm requires that edges are scan-converted to the rendered information in the z-buffer, thus it is not possible to alter the position of the edge vertices prior to this process. Therefore, there are two stages in artifact removal—one before the visibility processing of silhouette edges that retains the original vertices for edges, and one after visible silhouette segments have been found that might merge or alter vertices. The first two artifacts outlined below are treated in the first stage, and the subsequent three are handled after silhouette stroke generation.

Triangles With Two Visible Edges

Sometimes it happens that two silhouette edges share the same triangle (depicted in Figure 6.15) such that they cause sharp zig-zag lines when projected to the image plane. Applying a style in form of a texture to these zig-zag lines will result in awkward artifacts, due to the edges turning back on themselves. However, because a triangle with two silhouette edges is most likely to be almost parallel to the viewing direction, the artifacts created may be avoided by un-marking the two formerly marked silhouette edges and marking the remaining edge of the shared triangle instead (see Figure 6.15). This yields the result that one new silhouette edge covers the same span as the previous two, but with the zig-zag artifact removed. However, before this artifact reduction is applied, the orientation of the triangle is tested whether it is oriented parallel to the viewing direction. Only when the angle between triangle and viewing direction is below a certain small angle (≈ three degrees were used) are the marked edges switched. This avoids unwanted effects when both silhouette edges have to be visible (e. g., a triangle facet of a cube).

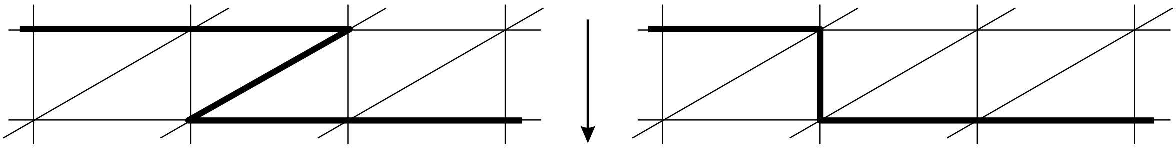


Figure 6.15: Example for handling triangles with two silhouette edges before and after the update. The arrow indicates the viewing direction.

Silhouette Edge Clusters

Another problem also arises due to small numerical errors and triangle surface approximation. It can occur when computing the visibility of faces that are almost parallel to the viewing direction. In these cases one might encounter lots of adjacent triangles that alternate between front-facing and back-facing, thereby creating clusters of silhouette edges (Figure 6.16 shows an example; one can find clusters of silhouette edges, for example, at the silhouette of cylinder-like shaped objects). This may result in many short strokes whereas one long stroke would be favorable.

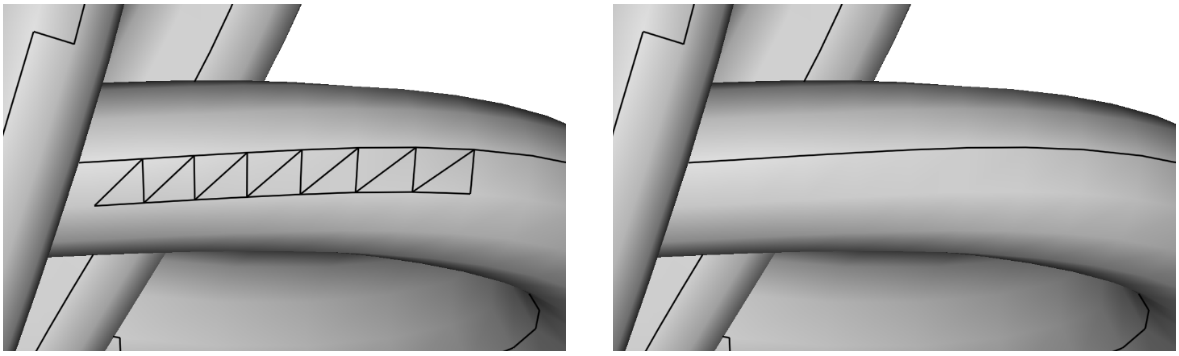


Figure 6.16: Example for treating silhouette edge clusters. In the screen-shots the upper part is visible and the lower part is not. The triangles with all edges marked alternate from not visible to visible. To remove the cluster, a path around it is detected, the inner edges are removed, and the longest segment of the path between the two leaving edges is removed. The left and right images show, respectively, the situation before and after the update.

One way to remove these silhouette edge clusters is to find each triangle that comprises three silhouette edges. Next, the closed path of outer silhouette edges around such groupings of triangles is detected. The silhouette edges that reside inside the closed path loop are eliminated. Now, because there is a loop, overlapping silhouette edges are generated when projected onto the screen. This means that one side of the loop can be removed so that a single connected path joins all the silhouette edges leaving the group of triangles (as an example, in Figure 6.16 the path comprising the most silhouette edges was removed). This is achieved by detecting the longest path (the one with the highest number of edges) between two neighboring leaving edges. Of course, if there is only one triangle in a triangle cluster (such as a tetrahedron with only one side visible) this remains untouched.

Silhouette Stroke Zig-Zags

After the projection into 2D, again due to numerical instabilities, there can be zig-zags in the generated strokes (see Figure 6.17). Parts of a stroke are classified as comprising zig-zags for each segment that has two sharp angles formed by its adjacent segments. If such a case is detected, the visible silhouette segment is replaced by just a vertex located in the middle of this segment. This typically eliminates the zig-zag.

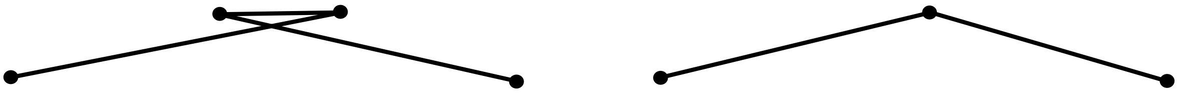


Figure 6.17: Example for removing zig-zags. The left side represents the result before and right side after the update. These zig-zags might originate from cases similar to the one displayed on the right side of Figure 6.15 due to a viewing direction which was changed just slightly in the counter-clockwise direction and minimal numerical errors during the visibility test. These can occur very easily because the viewing direction is almost parallel to the triangles.

Sharp Angles

Similar problems as with zig-zag lines can arise when there are acute angles in a stroke depending on the stroke width. Consider Figure 6.18. On the left side it can be observed that over the length of the segment the normal of the stroke texture is twisted by almost 90 degrees. Also, because of the high stroke width, the thickness at the vertex of the sharp angle is much smaller than at the other end. Both effects yield an unappealing image. Therefore, the silhouette stroke is split at the point of the sharp angle. This intends to stop the stroke in the vertex of the sharp angle and start a new one there for the remainder of the former stroke. This simulates an artist drawing one stroke, stopping, and then drawing back over his stroke.

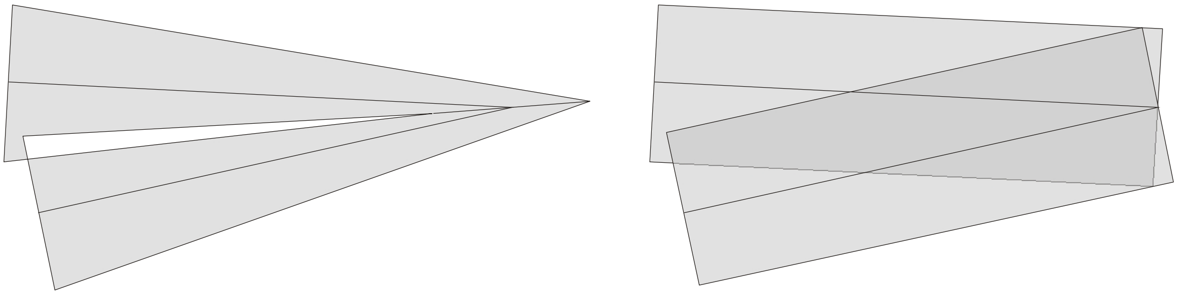


Figure 6.18: Example for splitting silhouette strokes at sharp angles. The left side shows the situation before the update where one stroke is twisted awkwardly. On the right side, after the update, the stroke has been split into two separate strokes yielding a better rendering.

Short Segments

Sometimes stroke segments with screen-projection lengths less than a few pixels are generated. The stroke quality can be improved by merging sub-pixel and short strokes together. Note that this eliminates many sharp angles occurring at the sub-pixel level that would influence the orientation changes and texturing of a stroke, whilst also acting as a level-of-detail feature by reducing geometry (see Figure 6.19).

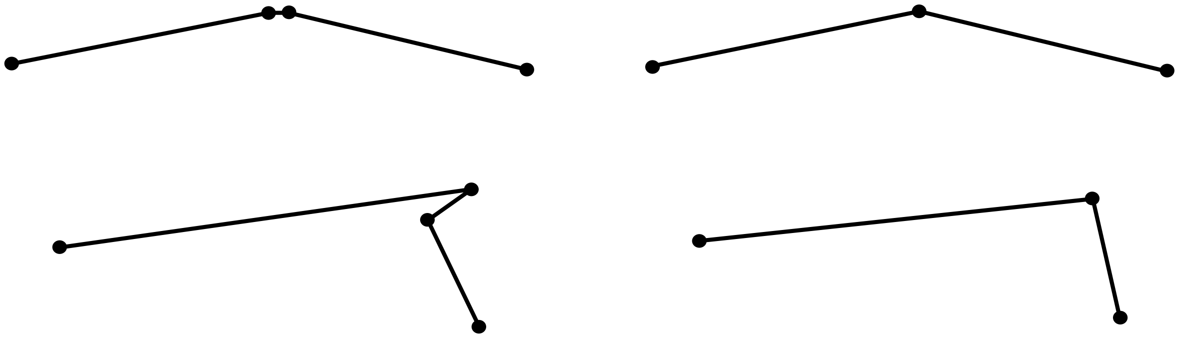
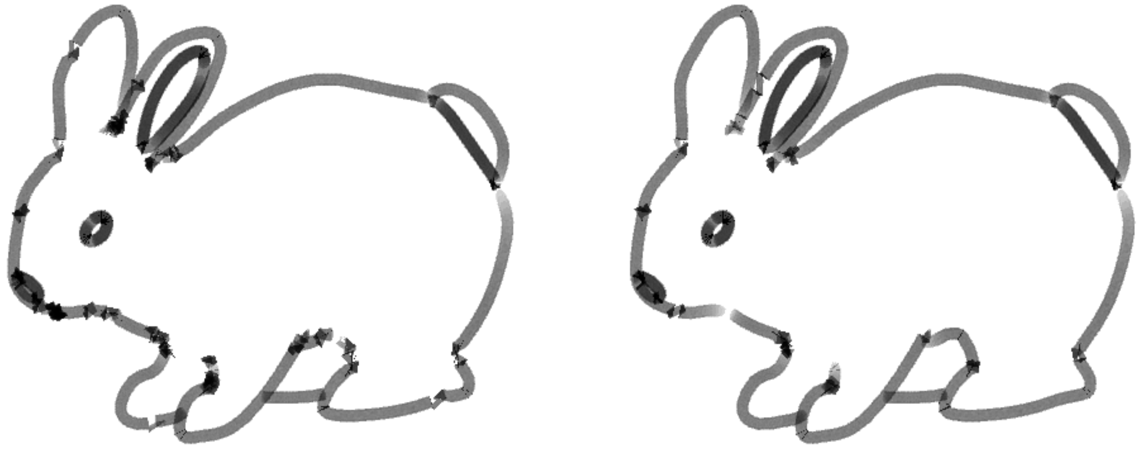


Figure 6.19: Example for removing short segments, left side before and right side after the update.

Results of Artifact Reduction

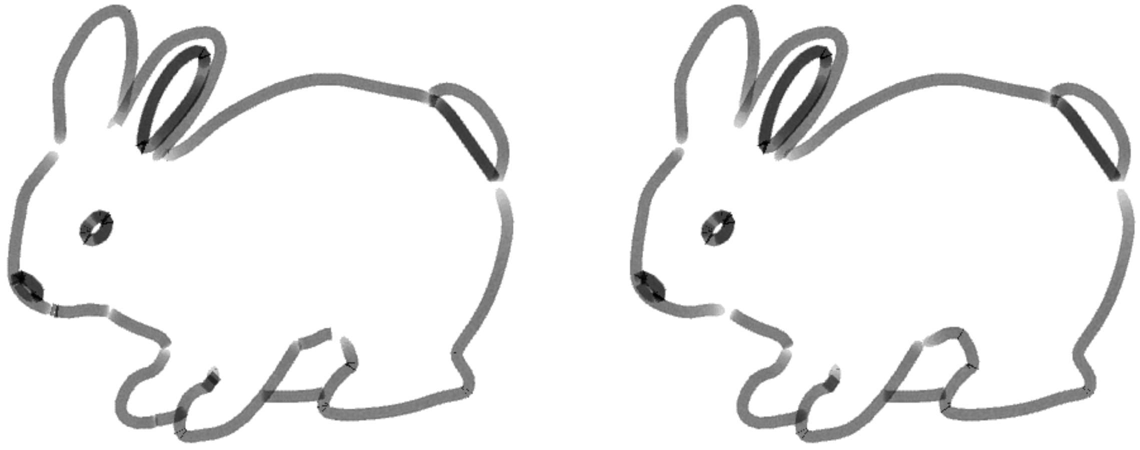
The artifact reduction techniques significantly improve the appearance of the final image. The slight displacements to silhouette strokes and segments might alter the real silhouette but do not significantly distort the line drawing. In contrast, for the application in stylized rendering they are necessary to yield appealing images (for applications that rely on an exact silhouette the artifact removal can be skipped). In Figure 6.20 the local improvements to the silhouette rendering can be seen. When no artifacts are removed at the geometric level, the projection of zig-zags and short segments create noticeable unwanted properties in the line stylization as depicted in

Figure 6.20(a). Figures 6.20(b) and 6.20(c) show elimination of visual artifacts before and after visible silhouette segment processing, respectively. However, only when both artifact removal stages are combined can the algorithm yield the smooth consistent outline present in Figure 6.20(d).



(a) No artifact removal.

(b) Pre-visibility artifact removal only.



(c) Post-visibility artifact removal only.

(d) All artifact removal techniques.

Figure 6.20: Examples for artifact removal. Remaining gaps are due to the applied line style.

6.5 Analysis of the Approach

In this section, the presented method for efficient visibility culling of silhouette edges and generation of silhouette strokes will be analyzed according to how well it is suited, in particular, for the application in interactive line rendering systems. First, a few examples are given to demonstrate the visual quality of the approach. Then, the approach is evaluated in terms of the rendering times required for the separate stages. Finally, a summary of the method is given naming the advantages over previous approaches, suggesting potential application areas, and discussing few disadvantages.

6.5.1 Examples

Figure 6.21 shows the application of different line styles to the silhouette strokes generated by the silhouette stroke based non-photorealistic rendering system OpenNPAR (see also Section B). The line styles can be modified interactively, such as by varying the line width and stroke textures. Perturbation functions (such as applying a sinusoidal wave over the strokes) are also possible, using both image-space or object-space algorithms. Image-space algorithms compute styles based on viewport projections whereas object-space algorithms can actually operate in three dimensions, such as varying thickness according to depth in order to implement depth cueing.

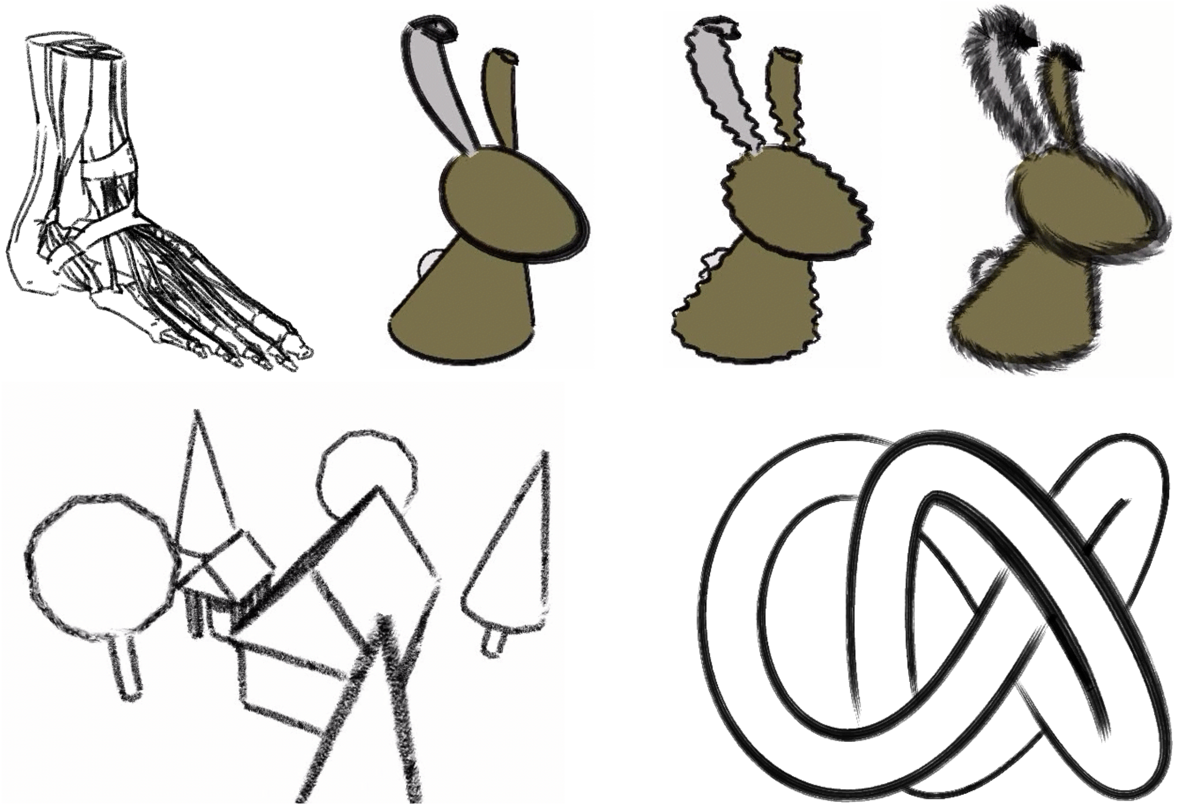


Figure 6.21: Examples for applying different line styles to the generated silhouette lines produced by the OpenNPAR system.

6.5.2 Computation Times

By providing one parameter that determines the number of pixels that are skipped in the silhouette edge visibility scan conversion process, a tuning mechanism is provided so that a trade-off between accuracy and speed can be decided upon (see Table 6.2). Determining this number depends on the resolution of the image (the higher the resolution, the more pixels may be skipped) and the geometric properties of the scene (objects that cover small areas would require reducing the number of pixels which are skipped). The extravagance of styles may also be taken into account (wavy lines, for instance, will overlap nearby objects anyway). From experiments it was found that a skipping value of six pixels has negligible visual impact for most cases.

skipping value (pixel)	1	2	3	4	5	6	7	8	9	10
time (ms)	13.1	12.1	12.0	11.7	11.5	10.8	10.4	10.7	10.5	10.6

Table 6.2: Computation times for the silhouette stroke generation (including stroke segment generation) depending on the skipping value for the bunny model with 3608 polygons and an approximate rendered size of 295 x 223 pixels.

In Tables 6.3 and 6.4 the computation times are shown for various stages of the rendering pipeline for a number of objects and polygon counts. It can be observed that the major bottleneck in the resulting frame-rate is the silhouette edge detection. However, this can be reduced by using any of the more efficient silhouette detection algorithms (e. g., using pre-processing) as reviewed in Section 6.3.

model	PN	SED		PrVAR		SSG		PoVAR		STY		total		fps	
		1P	2P	1P	2P	1P	2P	1P	2P	1P	2P	1P	2P	1P	2P
bunny	902	2	2	1	0	11	7	2	1	7	4	23	14	25.1	42.7
bunny	3608	7	7	1	1	11	6	3	2	7	4	29	20	21.7	32.0
bunny	14432	29	27	4	2	11	7	4	3	11	6	59	45	11.7	15.5
bunny	57728	114	108	8	6	14	9	9	5	11	6	156	134	4.8	5.8
knot	2880	4	4	1	0	12	7	2	1	6	3	25	15	24.9	42.7
knot	11520	21	21	2	1	12	7	4	2	11	6	50	37	14.3	21.4
foot	11244	24	21	8	7	22	13	15	7	35	17	104	65	7.5	10.7

Table 6.3: Computation times in milliseconds tested on a 450 MHz PentiumIII machine with a GeForce2MX graphics board and 256 MB RAM (1P, left columns) and a 800 MHz Dual-PentiumIII machine with a GeForce1 graphics board and 384 MB RAM (2P, right columns), both running MS Windows 2000 (PN = polygon number, SED = silhouette edge detection, PrVAR = pre-visibility artifact removal, SSG = silhouette stroke generation (including stroke segment generation), PoVAR = post-visibility artifact removal, STY = stylization). The skipping value was set to six pixels and the approximate rendered size was 295 x 223 pixels. The frame-rates given are total tested rates and less than to be expected from the computation times because they also include other activity like the standard OpenGL rendering of the model.

The computation times for the silhouette stroke generation depends on the number of 8-neighborhood tests (i. e., number and length of silhouette edges, the pixel coverage of the rendered object, and the skipping value). In Table 6.3 it is noted that computation time across the models is relatively constant, except for the foot model because the silhouette here is more complex. Higher resolution representations of models also yield higher computation times for the silhouette stroke generation because often silhouette edges are very short, and to short edges the skipping mechanism cannot be applied.

model	PN	SED		PrVAR		SSG		PoVAR		STY	
		1P	2P	1P	2P	1P	2P	1P	2P	1P	2P
bunny	902	8.7	14.3	4.3	0.0	47.8	50.0	8.7	7.1	30.4	28.6
bunny	3608	24.1	35.0	3.4	5.0	37.9	30.0	10.3	10.0	24.1	20.0
bunny	14432	49.2	60.0	6.8	4.4	18.6	15.6	6.8	6.7	18.6	13.3
bunny	57728	73.1	80.6	5.1	4.5	9.0	6.7	5.8	3.7	7.1	4.5
knot	2880	16.0	26.7	4.0	0.0	48.0	46.7	8.0	6.7	24.0	20.0
knot	11520	42.0	56.8	4.0	2.7	24.0	18.9	8.0	5.4	22.0	16.2
foot	11244	23.1	32.3	7.7	10.8	21.2	20.0	14.4	10.8	33.7	26.2

Table 6.4: Computation times in percent according to the data in Table 6.3 (abbreviations as in Table 6.3). Differences from a sum of 100% are due to rounding errors.

Altering skipping values gives computation times as shown in Table 6.2. The dependency of computation time for silhouette strokes (including stroke segment generation) on the size of the rendered image is demonstrated in Table 6.5.

size (pixel)	523 x 393	295 x 223	170 x 130	92 x 72	43 x 34
time (ms)	23.2	10.8	5.8	3.8	2.7

Table 6.5: Computation times for the silhouette stroke generation (including stroke segment generation) depending on the (approximate maximal) size of the rendered image for the bunny model with 3608 polygons and a skipping value of six.

The pre-visibility artifact removal sequentially processes all silhouette edges, hence computation times are dependent on the number of extracted silhouette edges. Post-visibility artifact removal and stylization operate over the generated silhouette strokes. Since strokes are comprised of segments this computation time depends on the number of visible silhouette segments.

6.5.3 Summary

The presented algorithm performs an efficient visibility culling of silhouette edges and generates silhouette strokes for application, e. g., in stylized rendering at interactive frame-rates (see also comparison with the visibility culling methods reviewed above in Table 6.6). Its input is the result of any algorithm that computes silhouette edges from a polygonal mesh. The introduced method performs a fast visibility check by combining analytic edge information available in the form of a Winged Edge data structure with the depth information in the z-buffer, but without having to render an additional image buffer. The expected complexity of the method depends linearly on the total projected length of all silhouette edges. This means that a few long edges take approximately the same time to compute as many short edges if both have the same projected length.

In addition, if the same set of edges is tested in a small and in a twice as big projection, the latter will take approximately twice as long to compute.[13]

algorithm	image-space/ hybrid/ object-space	accuracy	usable for stylization	computational complexity	G-buffers needed
z-buffer	image	pixel	no	$O(l_{pe} + n_e)$	z
[App67, MKT$^+$97]	object	analytic	yes	$O(n_e^2)$	none
[NM00]	hybrid	pixel	yes	$O(l_{pe} + n_e)$	ID
8-neighborhood	hybrid	pixel	yes	$O(l_{pe} + n_e)$	z

Table 6.6: Comparison of visibility culling algorithms (l_{pe} is the projected edge length and n_e denotes the number of edges).

The data structure also allows the algorithm to connect silhouette strokes from only those silhouette segments that are continuous on the object topology. The fact that this yields higher quality images becomes apparent when considering two silhouette segments that are continuous in the image plane (so that their endpoints touch or overlap each other) but are actually disjoint in object space (so that they represent different parts of the object or separate objects). The new approach would render two distinct strokes (one for each object silhouette) whereas earlier approaches that reconstruct strokes from the image buffer only [NM00] would draw a single stroke spanning over the two disjoint silhouette segments. The new approach can also easily be extended to allow multiple styles to be rendered across the scene since the analytic edges can be tagged accordingly.

The presented algorithm can now be used to generate silhouette drawings from a mesh. In combination with a pre-processing object-space silhouette detection method it is possible to generate interactive to real-time animations of line renditions using stylized strokes. It has been shown that trading the absolute accuracy of an object-space visibility culling technique for pixel accuracy allows for an enormous speedup. This pixel accuracy is sufficient for most applications which require the interactive display of silhouettes. In addition, in many cases the stylization applied to the generated strokes creates a sketchy appearance of the line renditions that hides the remaining artifacts (strokes that stop a few pixels short or after an occluding part's stroke). In such environments they appear to add to a sketchy look rather than disturb the image.

In contrast to previous hybrid approaches (such as the one by NORTHRUP and MARKOSIAN that uses an ID-buffer [NM00]), in most cases the presented new hybrid visibility

---

[13] This applies to models with most silhouette edges longer than the skipping value. If most edges are shorter than the skipping value, the expected complexity depends linearly on the number of edges because the 8-neighborhood test is performed for each vertex of the stroke.

culling method does not require additional G-buffers[14] to be generated. Instead, the z-buffer is used to determine the visibility status of the silhouette edges. The z-buffer is generated as a part of the traditional rendering process for hidden surface removal. This means that if the silhouettes are generated in addition to a traditional rendition this depth information comes at no extra cost. In addition, a number of artifact removal techniques have been introduced that filter the silhouette strokes before they are sent to the stroke stylization stage. These modified silhouettes might differ slightly from the originally computed silhouette, but will yield better images when styles are applied.

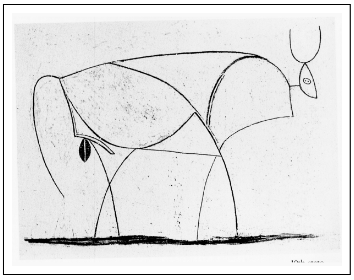
Computation times were recorded across a variety of models for each separate stage of the algorithm, demonstrating that interactive rates are achievable. When used to create animations, the algorithm performs well in maintaining strokes over successive frames, although no explicit frame-coherent measures are taken. Sometimes artifacts occur in stroke textures due to sudden varying lengths or start and endpoints of silhouette strokes. This occurs most noticeably when new silhouette segments are added to strokes in regions where the object comprises long edges in the geometry, and when strokes are split by occlusion from other objects. Popping artifacts are inherent to silhouettes generated from coarse models and can be improved by subdividing the model into finer triangles or by propagating parameterization information across successive frames. To handle sudden occlusion of strokes to maintain frame-coherency would require more investigation. In practice, however, the achieved results show that even with applied perturbations to the strokes the resulting animation maintains a degree of frame-coherency with only the occasional visual artifact.

Unfortunately, the algorithm sometimes does not perform well when trying to determine the visibility of sharp concave feature lines. In these cases the 8-neighborhood test often fails because most tested points in the adjacent faces will be closer to the viewer than the edge itself. However, this effect can be reduced by using a polygon offset (`glEnable(GL_POLYGON_OFFSET_FILL)` and `glPolygonOffset()` in OPENGL). Also, in cases where two independent silhouette edges are touching or almost touching, such that they lie at exactly the same z-distance from the viewer and separated from each other by less than one pixel in the frame-buffer, they might not pass the silhouette segment visibility test. The risk of this occurring, however, is minimal.

Stylized silhouette strokes have previously been implemented in a number of off-line systems that analytically generate high quality renditions (such as the daLi! system by MASUCH et al. [MSS97, MSS98]). However, the interactive frame-rates being achieved by the new hybrid approach are essential when working with stylized line drawings. Hence, stylized silhouettes can now be used in interactive and real-time applications [HIR$^+$03]. This means that it is now also possible for a designer to experiment with rendition in order to create a pleasing result. Appropriate interfaces to support this creative process are necessary and first attempts have been made towards this goal (see, e. g., the system by HALPER, SCHLECHTWEG, and STROTHOTTE [HSS02]).

---

[14] *G-buffers* are buffers with the size of the frame-buffer which store additional information, such as z-depth, normal data, object ID, face ID, edge ID, etc. They have been first introduced by SAITO and TAKAHASHI [ST90].



Chapter 7

Applications and Case Study

The concept of essential shape representations that was developed and discussed in this dissertation has a broad range of possible applications. In particular, the two main algorithms introduced and discussed in this dissertation—external skeleton extraction and visible silhouette stroke generation—can be employed in various different domains.

This chapter introduces a selection of potential application scenarios that go beyond the mere computation of feature or silhouette curves. First, applications that are based on the progressive coding of shape data will be examined. In this context, applications such as shape compression, mesh decimation, and network transfer are important. A second major application domain can make use of essential shape representations for shape matching. Interestingly enough, not only external skeletons but also silhouettes can be used in this area. Finally, non-photorealistic rendering is going to be analyzed as the main domain where silhouettes are used. However, the examples given show that also other ESRs such as internal and external skeletons can be used for creating non-photorealistic renditions. Moreover, the fact that both external skeletons and silhouettes are curves leads to additional applications of which some will be mentioned.

The variety of potential application areas illustrates the great potential of the introduced concept. Because of the many possible applications only a selection of the scenarios discussed in this chapter could be prototypically implemented. A number of examples are shown in the following sections in order to illustrate these scenarios. Other scenarios will be presented without having been implemented in order to point out additional application areas.

7.1 Progressive Coding of Shape Information

As one important property, the multi-scale character of the essential shape representation facilitates progressive coding of geometric models. This is not only important for storing the data structure in a file. In contrast, it also permits, among other things, lossy and lossless compression and storage, feature driven mesh decimation of geometric data, and optimized network transfer.

7.1.1 Progressive Meshes and Compression

Progressive coding of data is always very useful where large amounts of data have to be processed. It ensures that important features of the data are stored, transferred, or processed first while less important features are looked at later. Typical methods for progressive coding of polygonal meshes are based on mesh optimization techniques that use the edge collapse algorithm (e. g., [HDD$^+$93, Hop96, GH97, GH98] etc.).

These algorithms successively select edges from the mesh that are to be collapsed and, thus, reduce the complexity of the object's tessellation. The selection of edges to be collapsed is usually based on some kind of criterion that changes the shape of the object as little as possible (for example, using the shortest edge or using an edge which introduces the smallest possible error). This process is stopped once a sufficiently small mesh has been generated. This prevents the mesh from degenerating.

External skeletons can also be used in order to implement a progressive coding of meshes. In addition to the multi-scale property discussed in Section 5.4.1, the value of the used DoI function can be employed for this purpose. By setting both threshold values t_{1D} and t_{2D} to zero, a very detailed external skeleton is computed. Based on one of the previously mentioned progressive mesh algorithms, first all edges that are not part of the skeleton are collapsed (similar to the topology preserving algorithm in Section 5.4.2). Then, the skeleton edges are collapsed in the sequence of their DoI values (less significant first). Alternatively, the threshold values can be used to select which edges are to be collapsed next.

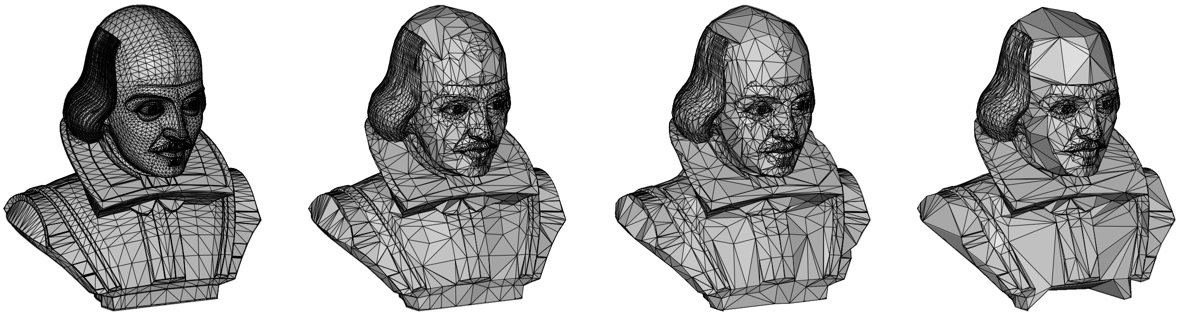
The progressive mesh can be decoded by reversing the edge collapse operations to vertex split operations [Hop96]. During this process, the most significant features are reconstructed first. Besides this lossless coding of mesh data, this also allows for the generation of different scale levels of the same object (Requirement 4). This process can be used in order to generate models that have the appropriate size for a specific application.

In addition, when disregarding the less important subset of the data, progressive coding can be used for lossy data compression (Requirement 2). The progressive mesh is then reduced to a certain percentage of the most significant vertex split operations. This means that insignificant details will be eliminated. Since these do not have to be stored anymore this leads to a compression of the shape representation with information loss, i. e., lossy data compression.

7.1.2 Feature Driven Mesh Decimation

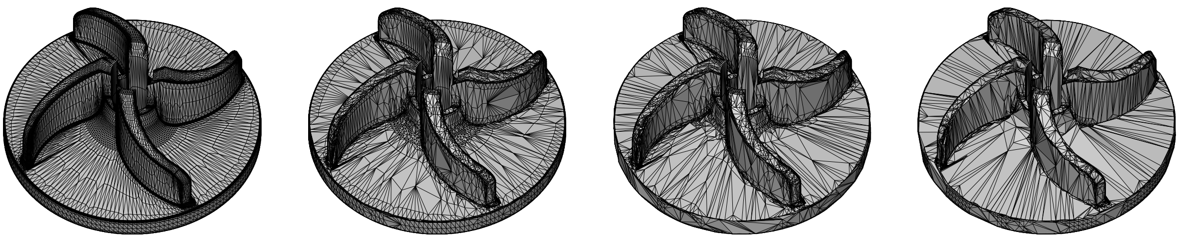
Closely related to the progressive storage of meshes and lossy compression are algorithms for mesh decimation. In this case compression as an aspect of data storage is not the desired trait. Instead, the reduction of the complexity of the mesh is the goal since this allows for faster processing of the data. However, the important features of the mesh should be preserved in this process. Previous approaches (e. g., [WB01]) used a geometric criterion to define these important features. External skeletons are very well suited for feature driven mesh decimation since they use DoI functions in order to define the features. This makes this approach more flexible than previous algorithms.

The application of feature driven mesh decimation is demonstrated in Figures 7.1 and 7.2 as well as in the previously shown example in Figure 5.23 on page 102.

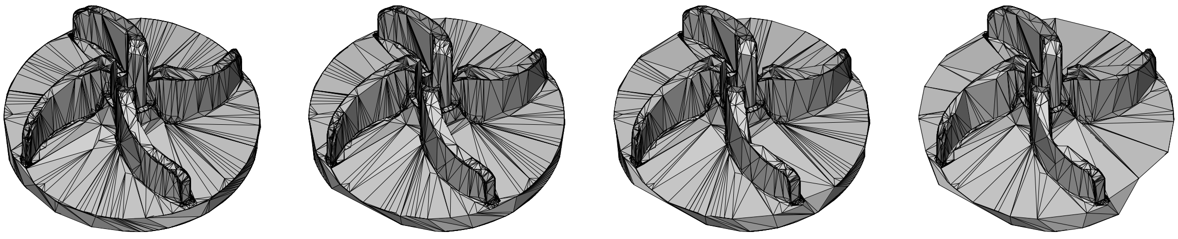


(a) 20 247 polygons. (b) 7 867 polygons. (c) 6 325 polygons. (d) 5 355 polygons.

Figure 7.1: Example for feature driven mesh decimation using the fold index. As can be observed, the features important in the shape are preserved even at high decimation rates.



(a) 30 840 polygons. (b) 19 020 polygons. (c) 14 580 polygons. (d) 10 806 polygons.



(e) 7 710 polygons. (f) 5 596 polygons. (g) 4 780 polygons. (h) 3 862 polygons.

Figure 7.2: Technical example for feature driven mesh decimation using the fold index. See also the visualization of the fold index DoI function for this model in Figure 4.20(k).

7.1.3 Linearizing the Mesh Data for Network Transfer

Both the progressive mesh representation of shapes and feature driven mesh decimation can be applied in networked transfer of object geometry data. Progressive meshes have an inherent ordering that can be used in networked transfer. In addition, the lossy compression or feature driven mesh decimation may be applied in order to reduce the amount of data that has to be sent. In any case, due to the progressive character of this transfer it is possible to reconstruct the mesh on the receiver's side using all the information that is available at a specific time. The receiver does not have to wait until all data has been received. In addition, the external skeleton graph itself also allows for linearization. The individual edges and vertices in the graph can easily be assigned importance values based on the selected DoI function. Using these DoI values the edges can be sorted and sent over a network connection using this sequence.

All mentioned application domains that are based on progressive coding of meshes benefit from the external skeleton approach's flexibility. Better than in previous algorithms it is now possible to define which features are important for a specific application rather than to be limited to a certain geometric property.

In addition to the application of essential shape representations for shape modifications in a broad sense, the structures extracted from shapes can also be used as descriptors of these shapes. This leads to the application domain of shape matching that will be discussed in the following section.

7.2 Shape Matching and Database Search

In the previously discussed shape processing using progressive coding it was important to manipulate shapes depending on the extracted external skeleton. In addition, essential shape representations can also be applied to the domain of shape matching database search for shapes. In this case shapes are not manipulated but rather a simplified representation is needed to represent a mesh.

The two main aspects of essential shape representations presented in this dissertation—skeletons and silhouettes—are both important in this application area. Both can be used to compare shapes in order to determine their geometric similarity. In the following, some examples for this application domain will be given. In addition, how the algorithms presented above can be applied in this context will be discussed.

7.2.1 Similarity Measure Based on Shape Abstractions

In general, when trying to match two shapes it is necessary to compare the geometric properties of the objects. Hence, it is necessary to compute a representation of the 3D shape that captures the geometric essence in some way.

Skeleton-like structures have previously been applied to matching of 3D shapes. HILAGA et al., for example, used Reeb graphs in order to compare two 3D models [HSKK01] (for details see Section 3.1.2). In a different approach, OSADA et al. discuss a method to compute shape histograms called *shape distributions* that also allow for comparing 3D shapes [OFCD02]. Their method can also deal with partially erroneous geometries which is very advantageous when dealing with arbitrary shapes.

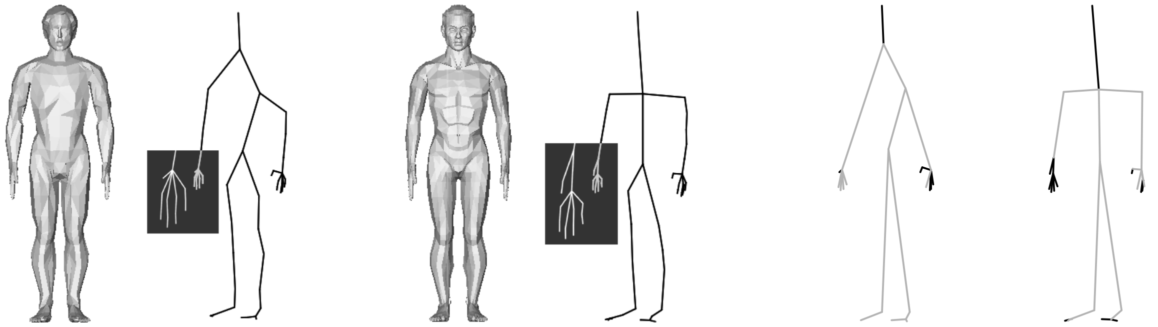
Funkhouser et al. use a shape descriptor based on spherical harmonics [FMK+03]. Although it cannot be used for object reconstruction, it can be used to determine the distance between two shapes and has the advantage of being orientation invariant.

A problem for such shape matching algorithms arises when comparing connected and non-connected shapes. Methods that compute different abstractions for similar shapes—one with connected components (e. g., topologically equivalent to one sphere) and one with logically separate components (e. g., topologically equivalent to many spheres)—are not well suited for shape comparison. Instead, the algorithms should provide the same abstraction for both type of objects.

7.2.2 Shape Matching with Edge Based Skeletons

In order to compute the similarity of two shapes, skeletons as discussed in this dissertation can be employed. Both internal and external skeletons are graphs. These can be used for comparing shapes instead of using their geometry for this purpose. Unfortunately, there is no known polynomial time algorithm for the graph isomorphy problem. However, skeleton graphs are usually not very large structures. In addition, they can be pruned in order to remove less important structures.

A small empirical study of shape matching based on skeleton graphs was conducted using internal skeletons (see details in [Bre03, BI04]). In this study, a selection of models were compared on the basis of their internal skeletons. Using a feature vector approach a scalar similarity value was computed. This was accomplished using a back-tracking algorithm that tries to find the best match of the two skeleton graphs that are being compared (see Figure 7.3). In order to simplify this search, nodes with a degree of two whose adjacent edges were approximately co-linear were removed in a preprocessing stage since they usually do not carry much information about the shape.



(a) Low detail model and skeleton.

(b) High detail model and skeleton.

(c) Skeleton comparison, common subgraph marked gray.

Figure 7.3: Comparing two similar shapes using simplified internal skeletons.

The empirical study showed that it is in fact possible to derive a similarity measure for shapes from matching their skeletons. Figure 7.4 shows the result of comparing a number of selected models. It can be seen that the algorithm performs quite well for piecewise tubular shapes. However, for compact or flat objects very similar skeletons are computed. This means that the algorithms can match their respective skeletons very well although the shapes may differ significantly. However, this is not a problem of the matching algorithm but of the edge collapse based internal skeleton extraction technique as discussed in Section 5.1.1.

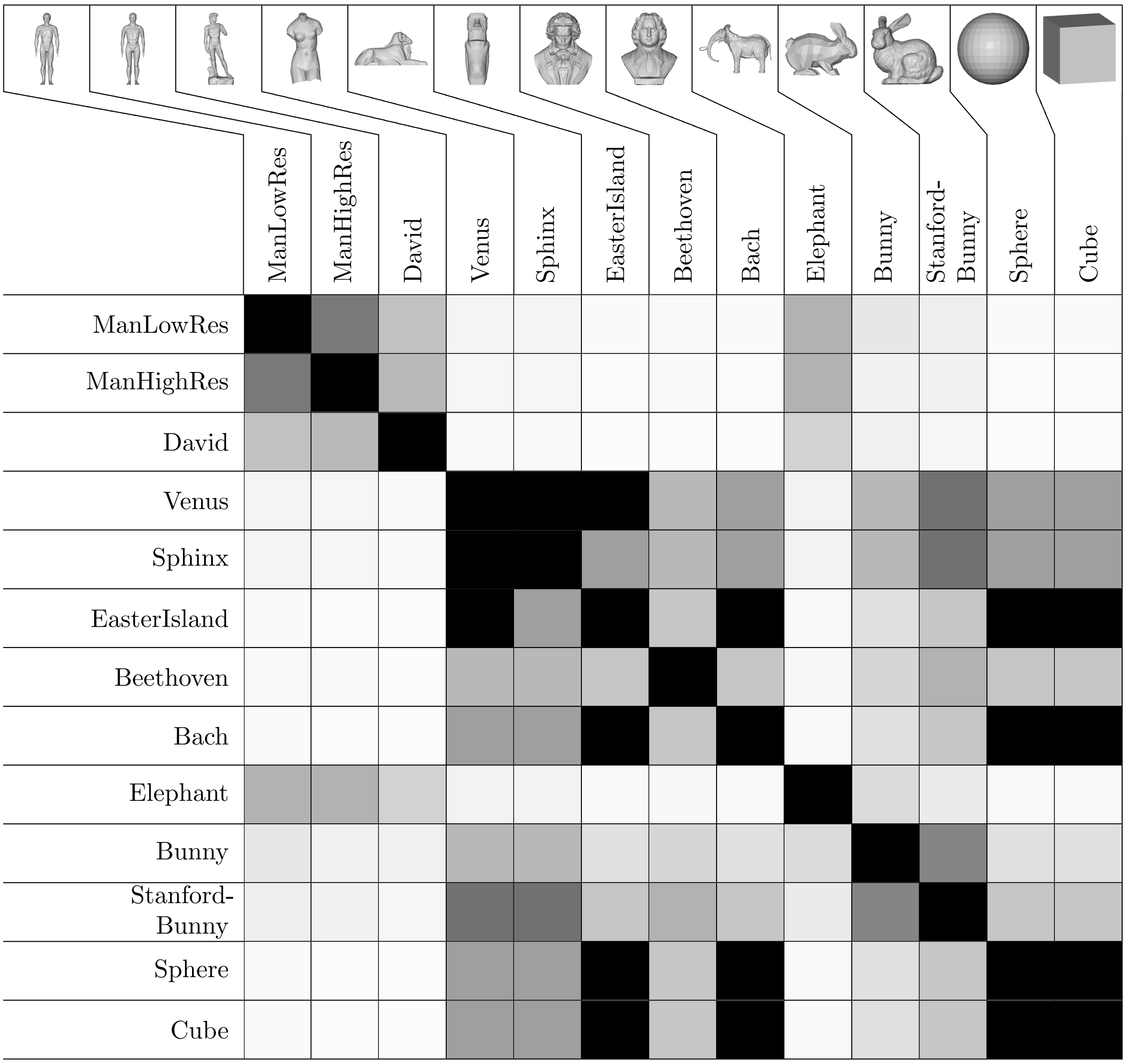


Figure 7.4: Comparison matrix for a number of example shapes. The gray value indicates the quality of the match, black indicates a perfect match and white denotes no match at all.

Although the study has been conducted with internal skeletons, external skeletons could just as well be used because both types of skeletons are represented as graphs. In fact, it would be easier to use external skeleton graphs since they additionally contain information about the importance of vertices in form of DoI values. This would simplify the back-tracking search for a match because the progressive coding can be used (Requirement 3). For this reason it is to be expected that external skeletons will perform better than internal skeletons. Fortunately, external skeletons provide similar graphs for connected and non-connected shapes. This is another reason why they will perform better than internal skeletons. However, a combination of both types can potentially be used in order to improve the results even more.

7.2.3 Shape Matching with Silhouettes

In addition to using 3D shape abstractions in order to match 3D shapes, it is also possible to use 2D shape abstractions, i. e., silhouettes. For example, MIN, CHEN, and FUNKHOUSER describe a method for matching hand-drawn silhouettes of shapes with 3D models [MCF02]. They note that users tend to use a fairly limited set of views on objects when asked to sketch silhouettes. In order to compute the match of the silhouettes with 3D models, they compute a series of traditional 2D projections of the models and try to match them individually with the user sketches. An individual match score is based on transformation into the frequency domain using Fourier analysis. This technique frequently used in two-dimensional shape comparison makes the matching process rotation invariant. The total match is combined from single ones.

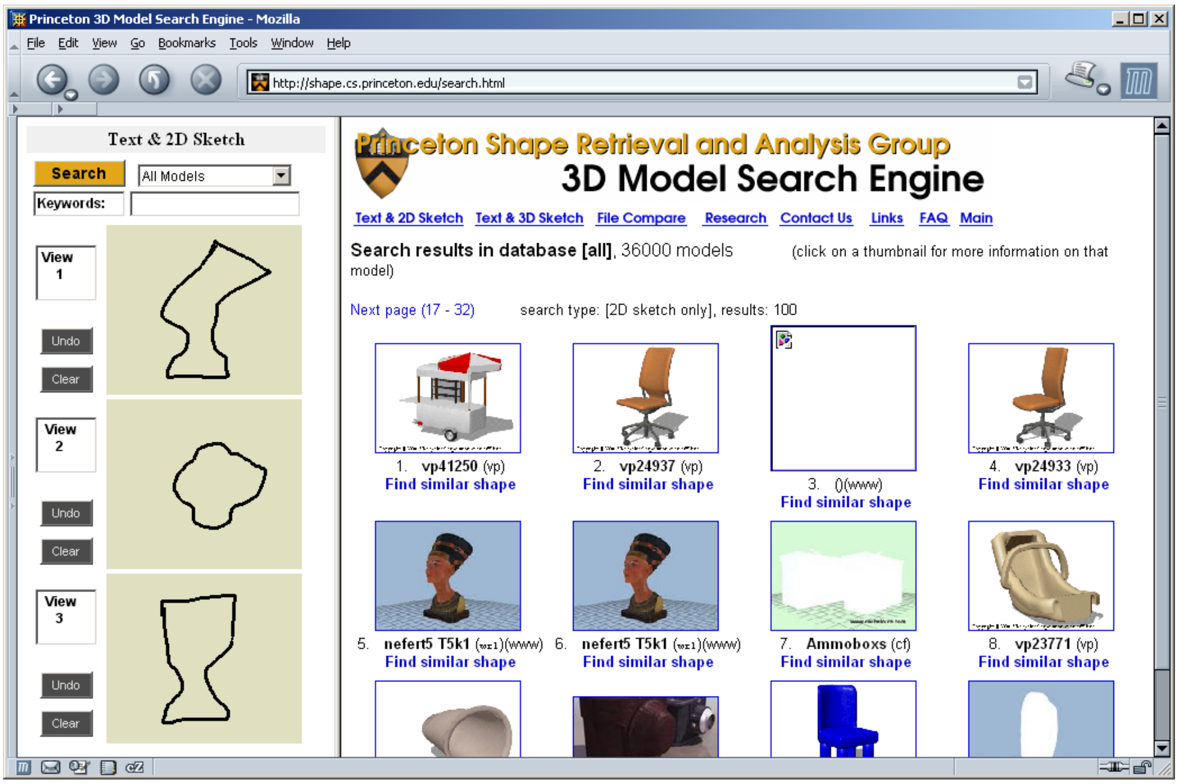
In order to improve this process, real silhouettes computed from the required views of the shapes can be used. This would create images by far more similar to the query images. Although this has not been tested, it can be expected that this would improve the quality of the shape matching. Similar to MIN, CHEN, and FUNKHOUSER's work [MCF02], it can then be applied to silhouette based shape database queries. These are more flexible and more user-friendly than keyword based searches. The search in shape databases as a potential application will be examined in the following section.

7.2.4 Model Database Search Using Silhouettes or Skeletons

Shape matching techniques based on shape abstractions such as skeletons and silhouettes can be employed in a shape database search. For example, the multi-resolution Reeb graph based shape matching technique by HILAGA et al. [HSKK01] has been applied in a 3D object retrieval system by CHEN and OUHYOUNG [CO02] as previously mentioned in Section 3.1.2. FUNKHOUSER et al. introduce a different system in [FMK$^+$03]: the PRINCETON 3D MODEL SEARCH ENGINE [RG03]. In addition to well-known annotation based keyword search the system can use both—the query by 2D silhouettes and the query by a shape descriptor as described in Section 7.2.2.

In both cases it is important what interface is presented to the user. For the 3D search it would be possible to allow the user to "sketch" a model using a known modeling tool. However, since this is a very technical interface it would be much too complicated for most users. Thus, the authors use the 3D sketching interface TEDDY as introduced by IGARASHI, MATSUOKA, and TANAKA [IMT99] (see Figure 7.5). It lets the user specify a shape from which the system computes a shape descriptor to query the database.

The silhouette based interface is easy to use as well. The user sketches the silhouette from up to three views on the model (see Figure 7.6). These sketches can be directly applied for querying the database.



Search results in database [all], 36000 models (click on a thumbnail for more information on that model)

search type: [2D sketch only], results: 100

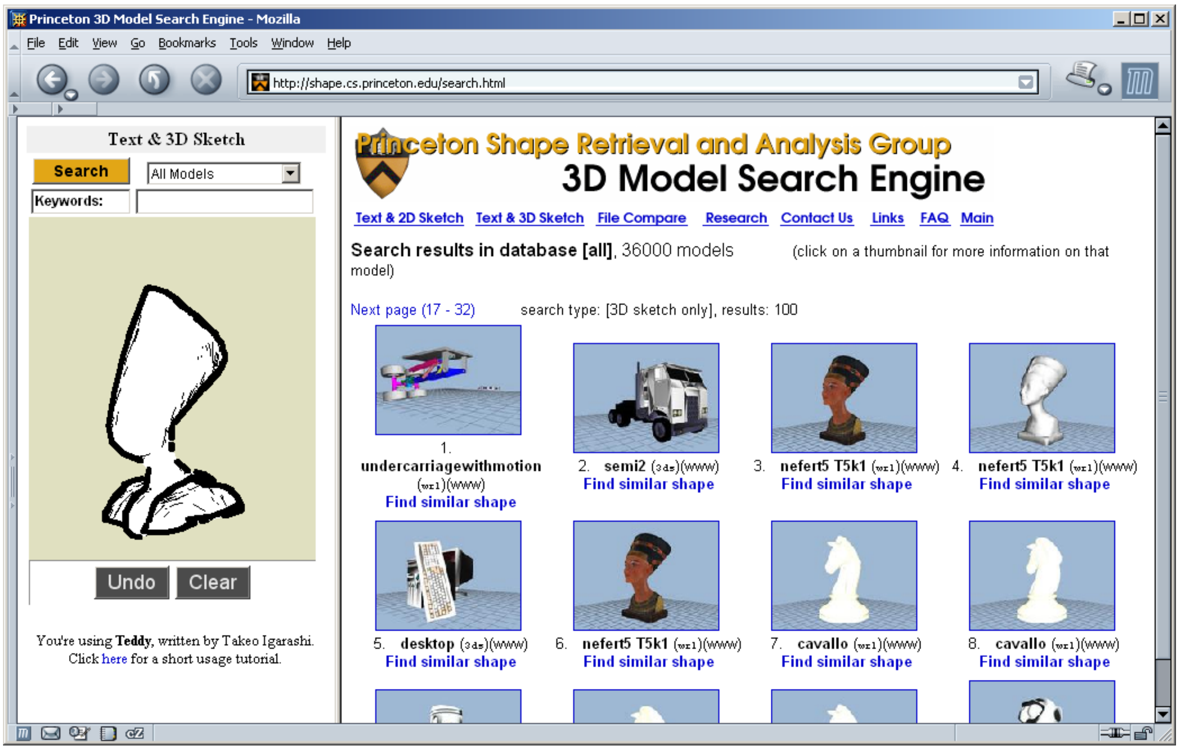
Figure 7.5: Searching for a model of Nefertiti's bust by sketching the silhouette using the PRINCETON 3D MODEL SEARCH ENGINE [RG03].

In both mentioned types of search queries, the concepts introduced in this dissertation can be employed. Potential applications include, for example, the use of external skeletons for a 3D based shape query. The fast silhouette algorithm, on the other hand, can be used for searching in a database by the shape of projections of the stored models. In fact, a combined version also seems to be possible: the user would sketch the shape using a system similar to TEDDY [IMT99]. From this 3D sketch both a silhouette and a 3D representation can be computed for searching the database.

In contrast to the shape manipulation and shape description applications for essential shape representations that were discussed so far, additional potential for applications of ESRs lies in using the extracted structures for the creation of renditions. In particular, the following section will examine non-photorealistic depictions created using ESRs.

7.3 Non-Photorealistic Rendering

The previously discussed shape manipulation and shape matching applications are the primary area where skeletons are used. However, this is not the case for silhouettes. Non-photorealistic rendering is certainly the main application for silhouette rendering. Nevertheless, skeletons can also be used in this domain. The following sections discuss



Search results in database [all], 36000 models (click on a thumbnail for more information on that model)

search type: [3D sketch only], results: 100

You're using **Teddy**, written by Takeo Igarashi. Click here for a short usage tutorial.

Figure 7.6: Searching for a model of Nefertiti's bust by sketching the shape using the PRINCETON 3D MODEL SEARCH ENGINE [RG03].

possible applications of not only silhouettes but also skeletons as non-photorealistic rendering techniques. Within the framework of the unifying system OPENNPAR within which the techniques discussed in this dissertation have been implemented (see Appendix B), new application domains for silhouettes and skeletons have emerged, some of which will be described as well.

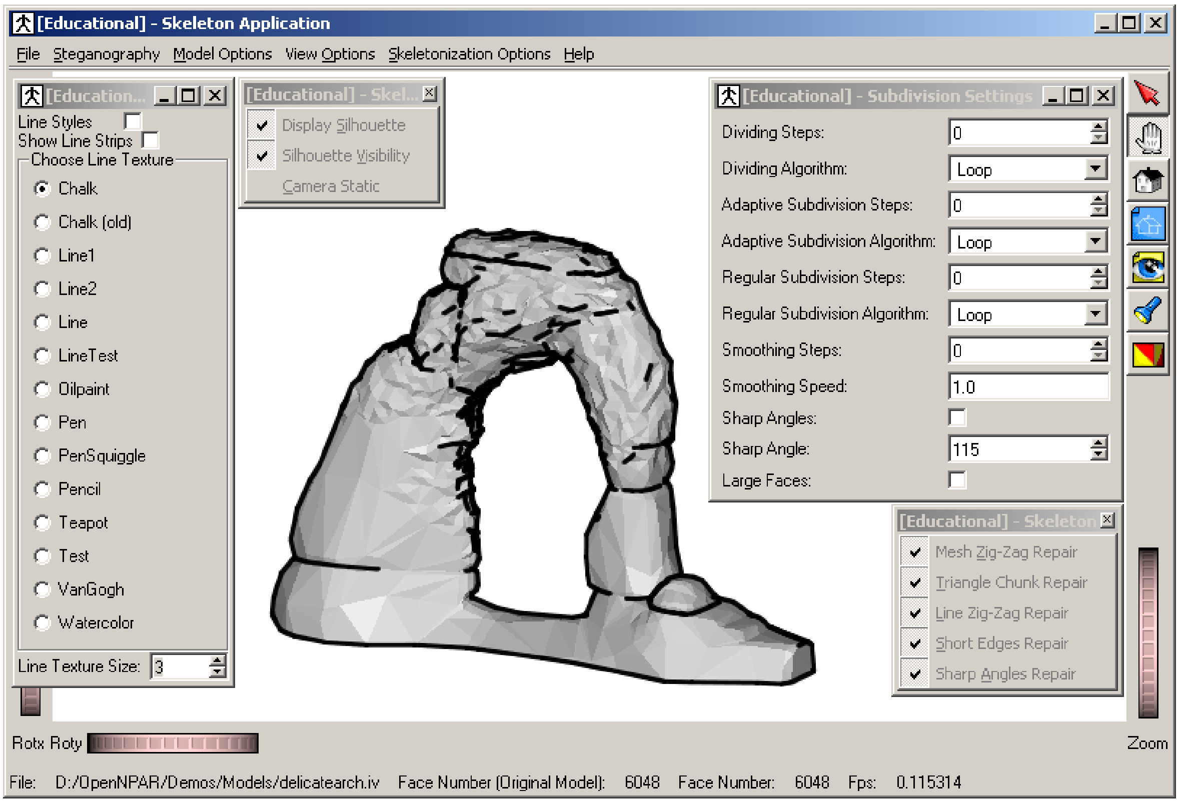
7.3.1 Silhouettes and Skeletons in Non-Photorealistic Techniques

A very obvious application for silhouette generation is to use them in non-photorealistic rendering by applying line styles to the generated strokes. In addition, combinations of silhouettes and skeletons promise to allow for new and interesting ways of creating non-photorealistic renditions. Some of these techniques will be outlined below.

Interactive Silhouette Line Rendering

The self-evident application for fast silhouette algorithms (a silhouette detection method using preprocessing combined with the hybrid visibility culling technique), as noted above, is the creation of interactive silhouette line renditions. They can be used, for example, for the creation of medical or technical illustrations or for comic style rendering. In particular, in application domains such as CAD and architectural design it is advantageous to use stylized line drawings instead of photorealistic renditions when presenting designs to customers as a study has shown [SPR+94, SSRL96].

In all of these cases it is beneficial to be able to render the images with interactive frame rates using, for example, the algorithm presented in Chapter 6 (see, e. g., Figure 7.7). The possibility for interaction with renditions reduces the limitations for designers who create non-photorealistic visualizations. This means that they can easily experiment with certain views and parameterizations which would hardly be possible with off-line rendering. However, the interactivity poses new problems to the rendering algorithms. In particular, frame coherency has to be maintained, especially when using line styles. Some of the attempts to maintain a good frame coherency in computer-generated silhouette line animations have been introduced in Section 6.4.4, others are discussed by Hertzmann and Zorin [HZ00] or Kalnins et al. [KDMF03].



Line Styles ☐
Show Line Strips ☐
Choose Line Texture
● Chalk
○ Chalk (old)
○ Line1
○ Line2
○ Line
○ LineTest
○ Oilpaint
○ Pen
○ PenSquiggle
○ Pencil
○ Teapot
○ Test
○ VanGogh
○ Watercolor
Line Texture Size: 3

☑ Display Silhouette
☑ Silhouette Visibility
Camera Static

Dividing Steps: 0
Dividing Algorithm: Loop
Adaptive Subdivision Steps: 0
Adaptive Subdivision Algorithm: Loop
Regular Subdivision Steps: 0
Regular Subdivision Algorithm: Loop
Smoothing Steps: 0
Smoothing Speed: 1.0
Sharp Angles: ☐
Sharp Angle: 115
Large Faces: ☐

☑ Mesh Zig-Zag Repair
☑ Triangle Chunk Repair
☑ Line Zig-Zag Repair
☑ Short Edges Repair
☑ Sharp Angles Repair

File: D:/OpenNPAR/Demos/Models/delicatearch.iv Face Number (Original Model): 6048 Face Number: 6048 Fps: 0.115314

Figure 7.7: Example application created with the OpenNPAR library that demonstrates interactive line rendering.

External Skeletons as Feature Lines

One problem with silhouette line drawings generated using object-space methods is that only the actual silhouettes are found by the silhouette detection algorithm itself.

However, line drawings usually also include lines other than the silhouette—commonly referred to as feature lines (also see Section 6.1).

Typically, the feature lines for line renditions generated from polygonal models are detected using the angle between two adjacent faces and comparing it to a user-defined threshold. Although this works well for models with sharp edges where feature lines ought to appear, it fails when the models have rounded corners. In such cases the feature lines can be computed using external skeletons and an appropriate DoI function (see example in Figure 7.8 where the maxima of κ_1 and the minima of κ_2 have been used in separate runs). This allows one to find local extrema for rounded corners of models as well as those at sharp edges. The algorithm may be modified in order to limit local extrema where their value is below a certain threshold which avoids insignificant feature lines.

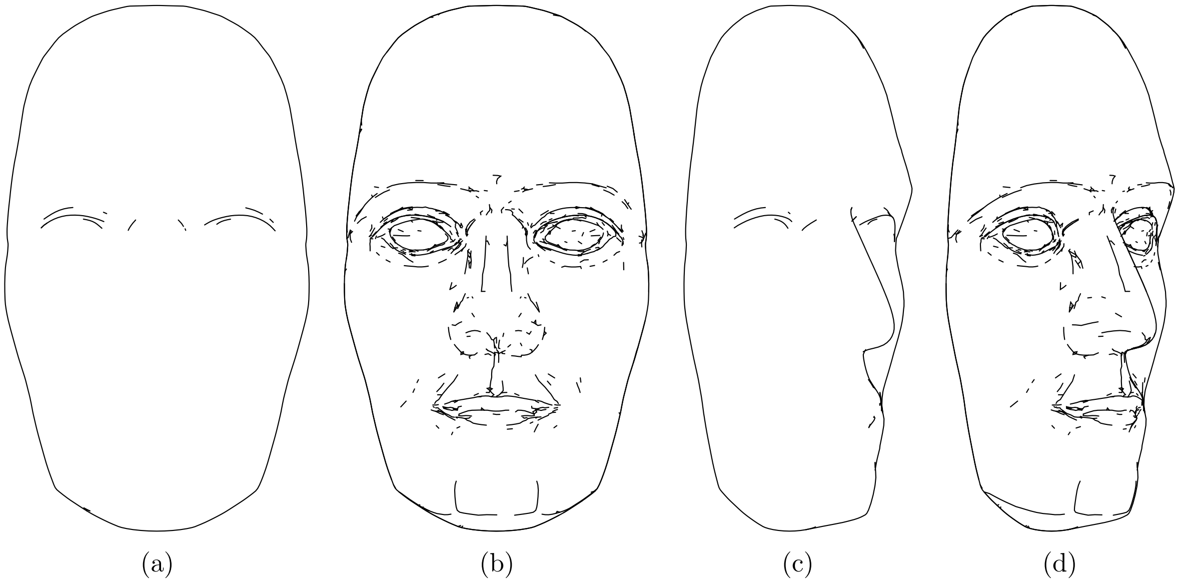


Figure 7.8: Comparison of using sharp angles between adjacent faces ((a) and (c)) and prototypically external skeletons ((b) and (d)) for computing feature lines of mostly smooth objects. The insignificant feature lines of the external skeleton have not been pruned yet in this example.

Improvement of Interactive Line Rendering Through Skeletons

One observation that can be made about silhouette line rendering is that it is not very efficient for objects that are far away from the viewer. Such objects typically do not contribute much to a rendition in terms of space covered. However, the computational cost is not different from objects close to the viewer. In addition, many very short strokes are produced that usually only results in an "ink blot" when line stylization is applied (see Figure 7.9 on the left).

In such cases it is possible to use a skeleton instead of the silhouette to represent distant objects [HIR$^+$03]. This not only has the advantage that the number of lines

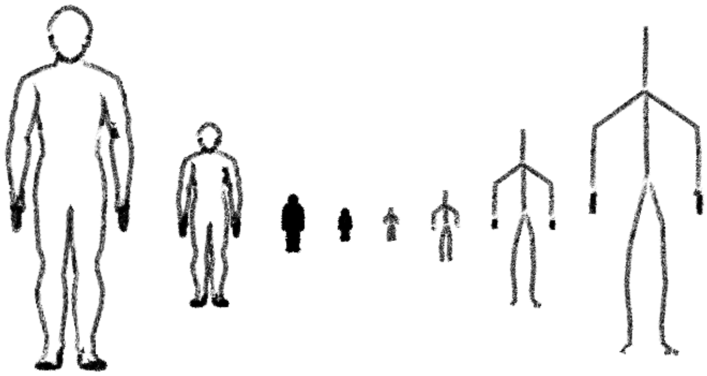


Figure 7.9: Comparison of rendering silhouette drawings with skeleton drawings for different distances of an object. Created in collaboration with Nick Halper.

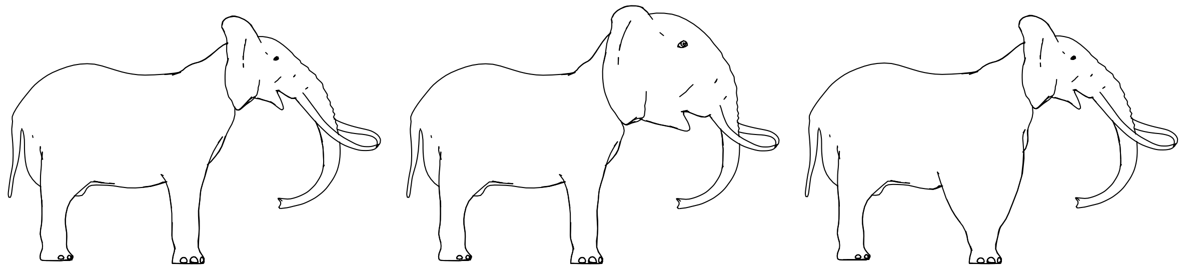
to be drawn is typically reduced. Also, the computational cost is much smaller since the skeleton only has to be computed once in contrast to a silhouette that has to be recomputed for every frame. Internal curve based skeletons can easily be employed for this purpose as demonstrated in Figure 7.9. It might also be possible to use external skeletons. This, however, has not yet been tested.

There are also other applications of skeletons within non-photorealistic rendering. For example, internal skeletons can be used to create hatching lines [DHR+99]. In a similar way, external skeletons could be used to guide the creation of hatching lines on the surface of polygonal meshes.

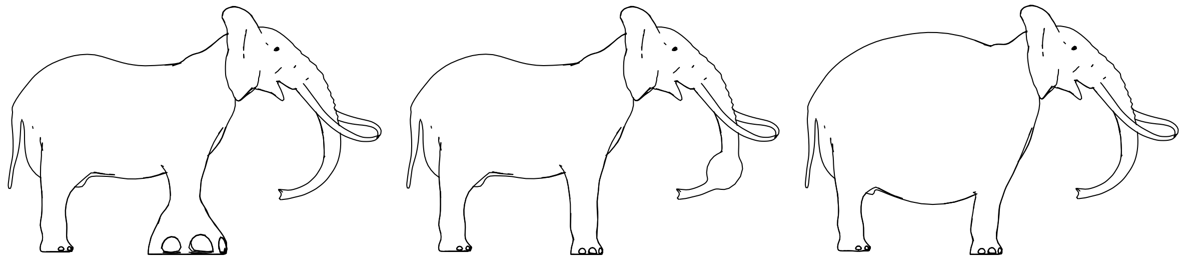
7.3.2 Stroke Manipulation Based Example Applications

One common property of silhouettes as well as skeletons is that both are lines or curves extracted from a mesh and can be represented as strokes. The major advantage of representing line data as strokes (as opposed to as accentuated pixels in a pixel image) is that it is still available as analytic data, i. e., as a vector graphic. Thus, a pipeline approach to stroke handling facilitates easy manipulation of these strokes. This means that a wide variety of applications that uses or manipulates stroke data is possible. Although some divert to some degree from the main topic of the dissertation, a few ideas will be presented below.

For example, two-dimensional distortion [Car99] can be applied to the strokes by manipulating the positions of the stroke's vertices. This allows, among other things, the easy creation of caricatures from 3D models. In Figure 7.10 an example for this application is shown. Starting with the same line rendition (shown in Figure 7.10(a)) various different line drawings were generated (Figures 7.10(b)–7.10(f)), each of them conveying a different meaning or impression. The advantage of this process is that an artist can interactively and in the same application select an appropriate view on the model, generate the line drawing (either using skeletons, silhouettes, or both), and then modify the rendition using distortion creating virtually endless possibilities.



(a) Un-distorted rendition. (b) Smart elephant. (c) Strong elephant.



(d) Bigfoot elephant. (e) Drinking elephant. (f) Elephant with big belly.

Figure 7.10: Examples for applying distortion to vector graphic line drawings in order to create caricatures. Created in collaboration with Petra Neumann and Sheelagh Carpendale.

As mentioned before, the analytic stroke data can easily be exported to a vector graphics file format (e. g., the PDF file format). Those files can now be used, for example, in illustrations. In particular, in the medical and technical domain silhouette renditions are used frequently. However, additional information about the geometric model, the image generation process, or the artist who created this model is often lost in the process of processing the image. Thus, such information should be stored along with the rendition. However, storing the data in separate files or data fields is not advisable since it will most certainly be lost when processing the graphic in vector graphic applications such as CorelDRAW.

Thus, algorithms for including the information directly within the vector graphic data can be employed—so-called *illustration watermarks* [SIDS03]. They change the vector graphic data in such a way that a bit sequence is embedded in the vertex sequence of the strokes. This can happen without changing the visual appearance of the rendition (for being able to use the graphic without noticing the additional data) or with changing it (to create a stylized appearance). Examples for both possibilities are shown in Figure 7.11.

Besides the creation of "pure" non-photorealistic renditions, a promising application of silhouette renditions lies in the combination with traditional photorealistic scenes—*hybrid rendering* [JI03]. However, the mere usage of silhouettes for parts of an otherwise shaded model leads to a *see through problem* as seen in Figure 7.12(a) and 7.12(b). Since some objects are represented only by their silhouette other objects can be seen through them. This can be avoided by adding additional non-photorealistic shading to those objects. In addition, this enhances the non-photorealistic character of the appearance

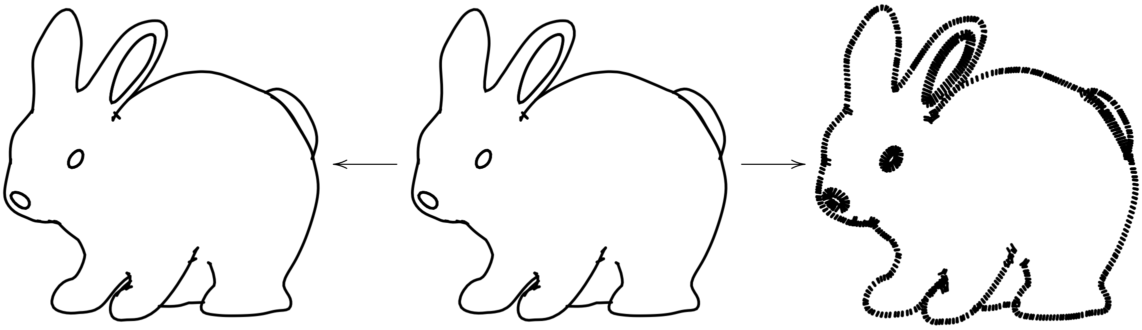


Figure 7.11: The silhouette is used to embed data with (on the right) or without (on the left) recognizable changes of the graphic. Other sources for 2D vector graphics are possible as well. Created in collaboration with Henry Sonnet.

of modified objects. Using dark silhouettes on a white or bright non-photorealistic shading, for example, conveys the impression and metaphor of drawing on a piece of paper (see Figures 7.12(c) and 7.12(d)).

The non-photorealistic component in these hybrid images can be used as an additional presentation variable. It is used to guide the viewer's attention to specific parts of interest. By adding animation in terms of continuously blending the silhouette component in and out this effect can be enhanced to make it even stronger (see examples in Figures 7.13 and 7.14 where some frames of such an animation are shown).

7.3.3 Combination with Alternative Stroke Generation

In addition to silhouettes, feature lines, and internal as well as external skeletons there are other techniques that are frequently used in non-photorealistic rendering. One of the most important of these is the usage of hatching. Hatching lines are strokes on the surface of an object that usually run almost parallel to each other (see examples in Figure 7.15). Sometimes, two or more sets of hatching lines are generated at a certain angle to each other (see example in Figure 7.16).

Since hatching lines are strokes most of the techniques described previously can be applied to hatching lines as well. For example, the hatching strokes have to be generated only once as a preprocessing step. For interaction with the model, using a newly developed hatching technique it could be demonstrated that the fast hybrid hidden line removal technique that was introduced in Section 6.4 can be applied to hatching strokes as well in order to remove the invisible subset [ZISS04] (this has been done in Figure 7.15). In addition to the possibility to generate new types of high quality illustrations, this primarily facilitates the creation of interactive illustrations. As other potential examples, illustration watermarks or distortion could easily be applied to hatching lines as well if a modular type of stroke handling is used.

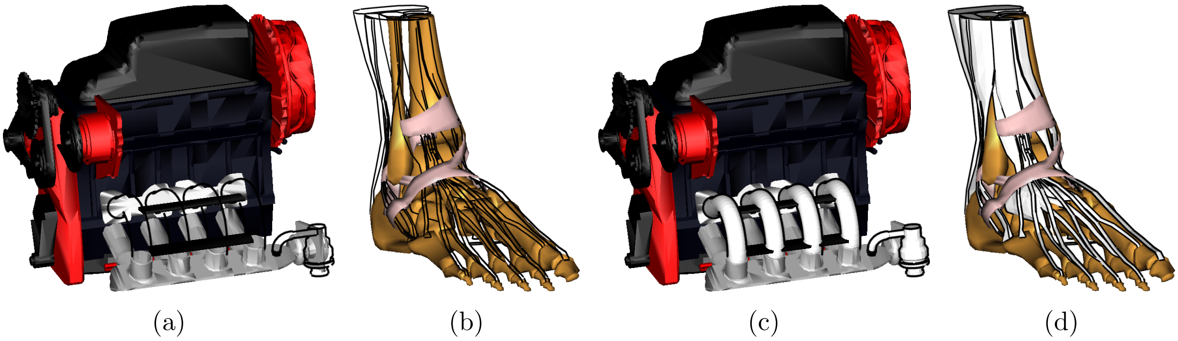

(a) (b) (c) (d)

Figure 7.12: When the shading of an object is exchanged with a line-based NPR technique, the photorealistically rendered remainder of the model is seen in the background (a) and (b), which is very distracting for many applications. Shading can be used to enhance the appearance of the line-based NPR style. This avoids that hidden surfaces are shining through the non-photorealistically rendered parts of the models. Created in collaboration with Roland JESSE and Bernd NETTELBECK.

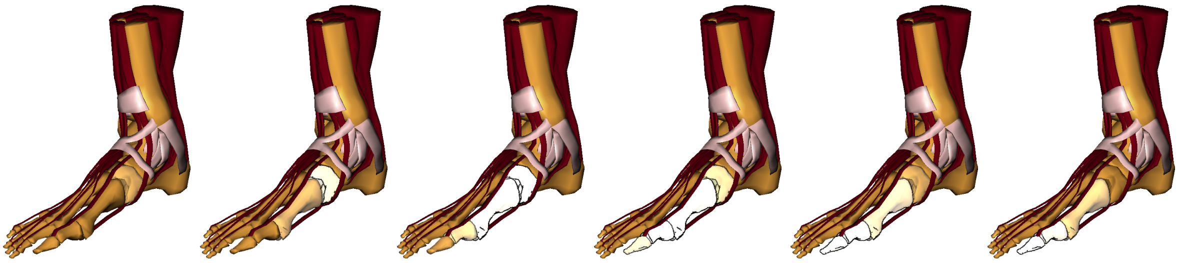


Figure 7.13: Snapshots of an animation pointing out specific parts of an anatomical example. The bones *Os cuneiformie I*, *Os metatarsale I*, *Phalanx proximalis I*, and *Phalanx distalis I* are emphasized in order to visualize their functionality. Created in collaboration with Roland JESSE and Bernd NETTELBECK.

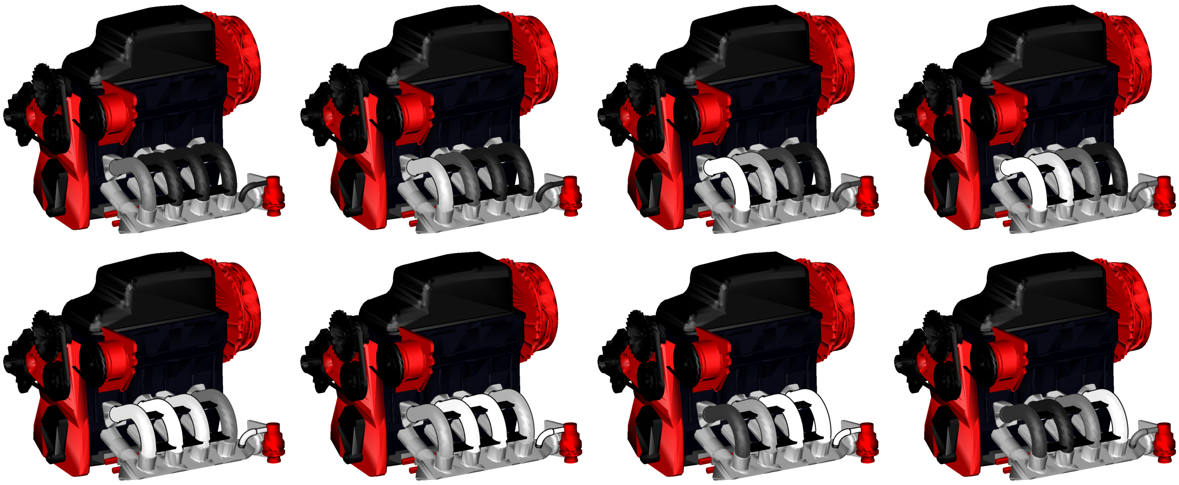


Figure 7.14: Snapshots of an engine model. The tubes of the engine's cooling system are gradually emphasized by being rendered in a line shaped silhouette style. The style changes reflect the possible utilization of the single tubes and their interplay with the cooling aggregate located ad the bottom right side. Created in collaboration with Roland JESSE and Bernd NETTELBECK.

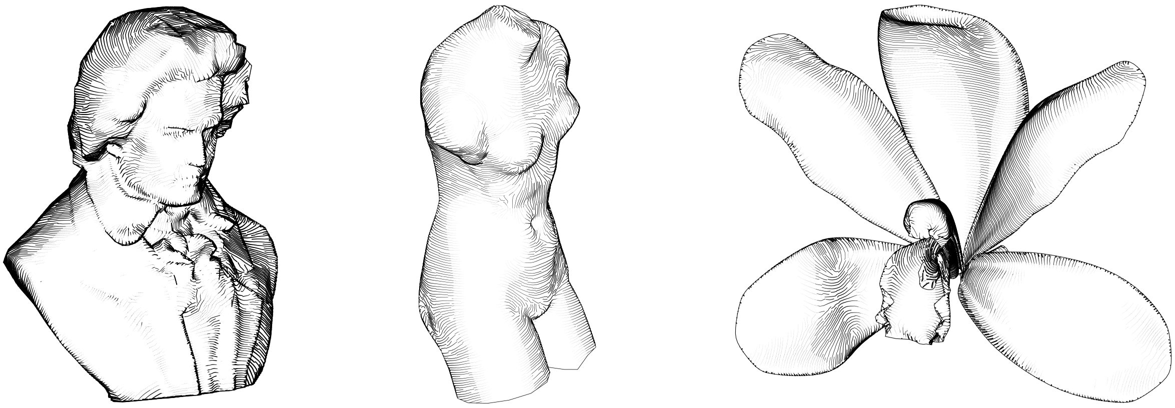


Figure 7.15: Hatching lines are another class of non-photorealistic rendering primitives which allow the generation of illustrations (combined with silhouettes in these examples). Created in collaboration with Johannes Zander and Stefan Schlechtweg.

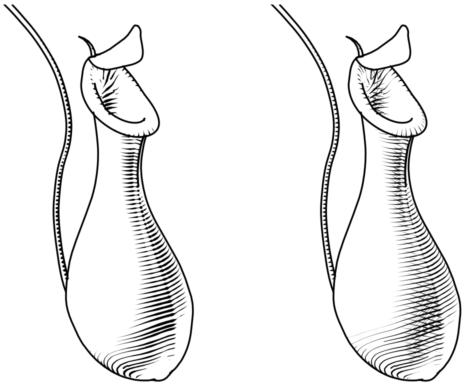


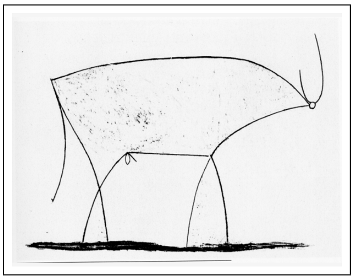
Figure 7.16: Rendition of the same model using single hatching (left) and triple hatching (right). Created in collaboration with Johannes Zander and Stefan Schlechtweg.

7.4 Summary

By the examples as well as exemplary tools and applications discussed in this chapter it could be demonstrated that it is possible to apply the concepts introduced in this dissertation in many areas. Potential application domains range from shape operations to non-photorealistic rendering.

It was discussed how external skeletons can be used for mesh decimation, a task very common in geometry processing. In addition, the application of skeletons and silhouettes in the domain of shape matching and model database queries was suggested.

It could also be demonstrated that both skeletons and silhouettes can be represented similarly as strokes. That way it is possible to process them all using the similar techniques. Important in this context is the modular character of the stroke based algorithms. This allows, for example, the easy combination of stroke modules. Stroke manipulation and visualization can be used without modifications with silhouettes, skeletons, and other stroke sources such as hatching lines. This facilitates the creation of many different non-photorealistic rendering techniques.



Chapter 8

Concluding Considerations

In this dissertation the topic of how to extract the essence of shape was examined. In particular, algorithms for the application to piecewise linear three-dimensional boundary representations (i. e., polygonal meshes) were explored.

Proposition 1:
Capturing the essence of shapes of three-dimensional models is very important in many applications in various domains. In most cases the essential shape information is used for distinguishing between significant and insignificant aspects of the geometry which in turn can be used in a number of different ways.

Among the example applications given in this dissertation are (progressive) shape storage, (networked) shape transmission, lossy and lossless shape compression, feature-driven shape resampling, as well as shape matching and shape database search. In addition, it has been demonstrated that essential shape information is relevant even in areas such as non-photorealistic rendering. The importance of detecting essential shape information is also demonstrated by the great variety of algorithms that try to solve this problem.

Proposition 2:
Previously presented methods for shape abstraction or feature extraction do capture the essence of shape. However, they are very specific in their potential application and, consequently, very numerous and seemingly disjoint.

The review of techniques in Chapter 3 demonstrates that there is indeed a great variety of algorithms for abstracting from shape or extracting features from objects. The techniques depend on and can be classified by the type of feature that is sought (e. g., curves or surfaces), the dimension of the object (i. e., mainly two-dimensional or three-dimensional objects), and the type of shape representation (i. e., discrete or analytic, volumetric data or boundary representations, etc.). However, even the features extracted by one group of algorithms can be very diverse. Therefore, it is difficult to use the same algorithm in different application domains. This was one of the main problems that is being solved by the techniques presented in this dissertation.

8.1 Summary

The discussion of the essence of shape started with the definition of the concept of essential shape representations in Chapter 2.

Proposition 3:
The concept of an essential shape representation (ESR) can be used for capturing the essence of shape in three-dimensional objects.

For ESRs, a number of requirements that were derived from the potential application areas were given. These requirements were used later on to evaluate ESRs reviewed in Chapter 3 and the newly developed external skeleton.

As one of the major properties of ESRs, it was argued that ESR scheme and ESR criterion should not be one unit. Otherwise the algorithm would be restricted to one specific problem in one application domain.

Proposition 4:
The specific ESR scheme has to be independent from the choice of ESR criterion in order to allow for maximum flexibility in applications.

The separation of scheme and criterion makes it possible to freely exchange the ESR criterion in order to adapt the algorithm to the application. This allows the use of a broader range of features and is not restricted to, e. g., angles between adjacent faces. In addition, different ESR schemes are possible and can be used with the same criterion. This is useful in cases where the criterion accurately captures the essence but an ESR scheme has to be found that efficiently operates on the specific shape representation.

It was discussed that it is not possible to exactly say what is important and what is not because this depends on the specific application.

Proposition 5:
It cannot be a priori defined what the essence of a shape is given just a geometric model because what is considered to be important varies between applications. ESR criteria can be specified in order to characterize essence of shape for a specific application.

The use of ESR criteria has the advantage in that it can now be specified what is important for a specific application. For polygonal objects, so-called degree of interest (DoI) functions that are used to compute the importance of a vertex, edge, or face for the shape of the mesh were developed. Arbitrary geometric and non-geometric DoI functions are possible. In Chapter 4 a variety of DoI functions was conceived and examined. The discussion of DoI functions focused on geometric functions. In particular, functions that are based on the local first and second principal curvature κ_1 and κ_2 that were approximated for the polygonal mesh were derived.

These DoI functions can now be used to determine curves on the surface of objects that characterize the essence of the their shape.

Proposition 6:
An ESR criterion determines a network of extreme ridges on the surface of an object. These extreme ridges characterize the location that carries the essential shape information.

These extreme ridges have to be identified. This requires a complete traversal of the mesh. In order to avoid visiting certain areas of the mesh twice, a wave propagation algorithm was proposed in Chapter 5. The wave propagation is initiated by local (unrestricted) maxima of the chosen DoI function.

Proposition 7:
A wave propagation algorithm can be used for identifying the network of extreme ridges in the surface of a mesh. The resulting structure is an essential shape representation and is called external skeleton of the mesh.

The wave propagation technique not only serves as a means for traversing the mesh but also yields wavefronts approximately perpendicular to the sought extreme ridges. Thus, the extreme ridges can easily be located by searching for restricted local maxima of the DoI function on the wavefronts. These are finally connected across wavefronts yielding the graph of extreme ridges—the external skeleton.

In Chapter 6 a topic was approached that appears not to be connected to external skeletons at first sight—the computation of silhouettes. However, it was demonstrated that in fact silhouettes are essential shape representations as well.

Proposition 8:
The essential shape representation concept also has other potential applications above and beyond feature detection. For example, both external skeletons (i. e., feature lines) and silhouettes can be considered to be essential shape representations.

However, it was argued that although the wave propagation based algorithm for external skeletons can also be applied to the computation of silhouettes this is not the ideal way. Instead, silhouettes are usually used in interactive or real-time rendering so that the algorithm has to allow for at least interactivity.

Proposition 9:
Algorithms for computing silhouettes will typically differ from those for external skeletons due to the different requirements of their respective main applications.

In fact, from the example of silhouettes and external skeletons it can be inferred that different algorithms are necessary for working with dynamic DoIs (e. g., the silhouette property) than are for static DoIs (the ones used for computing the external skeleton).

Proposition 10:
In general, dynamic criteria need different algorithms compared to those used with static criteria in order to facilitate interactive or real-time applications.

In order to allow for efficient silhouette computation, a comprehensive survey was conducted to identify the group of algorithms for silhouette detection that can be applied to interactive, stylized rendering. In addition, it was concluded that detecting the silhouettes is not enough for generating appropriate renditions. Instead, the visibility of silhouettes has to be determined as well.

Therefore, a method was developed for visibility culling of silhouettes. This algorithm takes the special properties of silhouette edges into account, i. e., that they are—visually—located at the border between object and background. The main advantage of this algorithm is that it usually does not require the computation of additional information because it makes use of the z-buffer. In addition, a set of artifact removal techniques that reduces the artifacts of silhouettes strokes that are caused by the polygonal mesh of the objects was conceived.

Chapter 7 gave an overview of potential application areas for the algorithms developed in the dissertation. Three major domains were discussed in this context: progressive coding of shape information, shape matching and database search, and non-photorealistic rendering. For each of these, a number of applications was discussed and some of them demonstrated with examples. This diverse selection of application examples also nicely supports Proposition 1.

8.2 Criticism

As has been shown not only in the case study chapter, the wave propagation based algorithm works well for feature detection when applied to meshes that have a fine and balanced tessellation. This is caused by the fact that the wave propagation ideally needs geodesic wave movement on the surface.

Proposition 11:
The wave propagation algorithm works well on balanced and finely tessellated meshes.

Unfortunately, this geodesic wave movement cannot fully be guaranteed by a polygonal surface. However, balanced tessellations provide a good basis for approximately geodesic wave movement even on polygonal meshes.

The algorithm works well with those models that inherently have such a balanced tesselation. For example, models derived from three-dimensional object scanning typically contain many small triangles of about the same size. In addition, such models often are very large and have to be pre-processed in order to reduce the amount of polygons they use. In this domain the wave propagation based external skeleton extraction would perform very well. Unfortunately, this could not be confirmed in practice due to the lack of appropriate model data.

A problem when working with the external skeleton algorithm consists in finding good values for the thresholds t_{2D} and t_{1D} in order to capture what can be considered to be the major aspects of shape is often tedious. Thresholds that work well for one model may not necessarily work well for a different one. Automatic ways to derive the threshold values would be desirable.

8.3 Future Work

In addition to the ideas for solving some of the problems discussed above there are also some major directions for continuing the work presented in this theses. These will be presented below.

The problem mentioned in Proposition 11 that a balanced tessellation is necessary for optimal performance of the algorithm can be overcome by applying a remeshing algorithm to the models prior to external skeleton extraction. For example, HORMANN, Labsik, and Greiner suggest an algorithm that can generate a balanced tessellation [HLG01]. Alternatively, the method presented by Alliez, Meyer, and Desbrun could be used [AMD02].

Moreover, remeshing could also be applied depending on the chosen DoI function. In contrast to the balanced tessellation created proposed to be used above, a remeshing algorithm could also be parameterized so that it generates higher tessellations at interesting portions of the surface an lower tessellations at dull parts. This in turn would cause the wave propagation to move slower in the interesting part creating a more detailed skeleton there. In uninteresting sections, on the other hand, the wave fronts would move slower and generate less detailed skeletons.

Even better than using remeshing for generating balanced tessellations would be to use a geodesic way of wave movement on the surface. The algorithm would then be independent from the actual tessellation of the mesh which would make the results more stable.

The polygonal nature of the meshes used for silhouette extraction also poses some problems for silhouette computation in that rendering artifacts are produced (e. g., zigzags). The artifact removal techniques introduced in Section 6.4.4 help to remove these artifacts but cannot eliminate them in all cases. Thus, using the silhouette detection algorithm that generates sub-polygon silhouettes may help avoiding the artifacts instead of removing them afterwards. Hence, it should be examined in the future how this affects the performance of the whole algorithm.

In addition, a case study to determine which of the developed DoI functions works best for computing an external skeleton should also be conducted in the future. This study could judge the DoI functions according to, among others, the type of model they are applied to (e. g., technical or medical models) or the application the external skeleton extraction is applied to.

References

[AAAG95a] Oswin Aichholzer, David Alberts, Franz Aurenhammer, and Bernd Gärtner. A Novel Type of Skeleton for Polygons. *Journal of Universal Computer Science*, 1(12):752–761, December 1995.

[AAAG95b] Oswin Aichholzer, David Alberts, Franz Aurenhammer, and Bernd Gärtner. Straight Skeletons of Simple Polygons. In J. Chen, X. Shi, and W. Gao, editors, *Proceedings of the 4th International Symposium at LIESMARS (LIESMARS'95, October 25–27, 1995, Wuhan, China)*, pages 114–124, 1995.

[AB99] Nina Amenta and Marshall Bern. Surface Reconstruction by Voronoi Filtering. *Discrete and Computational Geometry*, 22(4):481–504, September 1999.

[ABK98] Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A New Voronoi-Based Surface Reconstruction Algorithm. In Michael Cohen, editor, *Proceedings of SIGGRAPH 98 (Orlando, FL, July 19–24, 1998)*, *Computer Graphics* Proceedings, Annual Conference Series, pages 415–421, New York, 1998. ACM SIGGRAPH, ACM Press.

[ABOK94] Elena V. Anoshkina, Alexander G. Belyaev, Oleg G. Okunev, and Tosiyasu Laurence Kunii. Ridges and Ravines: A Singularity Approach. *International Journal of Shape Modeling*, 1(1):1–11, September 1994.

[ACD+01] Daniel I. Azuma, Brian Curless, Tom Duchamp, David H. Salesin, Werner Stuetzle, and Daniel N. Wood. View-Dependent Refinement of Multiresolution Meshes with Subdivision Connectivity. Technical Report UW-CSE-2001-10-02, University of Washington, October 2001.

[AFR03] Ashish Amresh, Gerald Farin, and Anshuman Razdan. Adaptive Subdivision Schemes for Triangular Meshes. In Gerald Farin, Bernd Hamann, and Hans Hagen, editors, *Hierarchical and Geometric Methods in Scientific Visualization*, pages 319–328. Springer-Verlag, Berlin · Heidelberg · New York, 2003.

[AMD02] Pierre Alliez, Mark Meyer, and Methieu Desbrun. Interactive Geometry Remeshing. *acm Transactions on Graphics, Proceedings of ACM SIGGRAPH 2002*, 21(3):347–354, July 2002.

[App67] Arthur Appel. The Notion of Quantitative Invisibility and the Machine Rendering of Solids. In *Proceedings of the* 22^{nd} *ACM National Conference*, pages 387–393, Washington, DC, 1967. Thompson Books.

[Aur91] Franz Aurenhammer. Voronoi Diagrams—A Survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys*, 23(3):345–405, September 1991.

[AWC⁺03] Daniel I. Azuma, Daniel N. Wood, Brian Curless, Tom Duchamp, David H. Salesin, and Werner Stuetzle. View-Dependent Refinement of Multiresolution Meshes with Subdivision Connectivity. In Stephen N. Spencer, editor, *Proceedings of AFRIGRAPH 2003*, pages 69–78, New York, 2003. ACM SIGGRAPH.

[BAK97] Alexander G. Belyaev, Elena V. Anoshkina, and Tosiyasu Laurence Kunii. Ridges, Ravines, and Singularities. In Anatolij T. Fomenko and Tosiyasu Laurence Kunii, editors, *Topological Modeling for Visualization*, chapter 18, pages 375–383. Springer-Verlag, Berlin · Heidelberg · New York, 1997.

[Bau75] Bruce G. Baumgart. A Polyhedral Representation for Computer Vision. In *Proceedings AFIPS National Computer Conference*, volume 44, pages 589–596, 1975.

[BBK96] Alexander G. Belyaev, Ilia A. Bogaevski, and Tosiyasu Laurence Kunii. Principal Direction Ridges. Technical Report 96-4-001, Center for Mathematical Sciences, The University of Aizu, Japan, 1996.

[BE99] Fabien Benichou and Gershon Elber. Output Sensitive Extraction of Silhouettes from Polygonal Geometry. In *Proceedings of the* 7^{th} *Pacific Graphics Conference (Seoul, Korea)*, pages 60–69, Los Alamitos, CA, 1999. IEEE Computer Society, IEEE Computer Society Press.

[BI04] Angela Brennecke and Tobias Isenberg. 3D Shape Matching Using Skeleton Graphs. In Thomas Schulze, Stefan Schlechtweg, and Volkmar Hinz, editors, *Simulation und Visualisierung 2004*, pages 299–310, Erlangen, San Diego, 2004. SCS European Publishing House.

[Bia01] Silvia Biasotti. Topological Techniques for Shape Understanding. In *Central European Seminar on Computer Graphics for Students*, 2001.

[BL99] Jules Bloomenthal and Chek Lim. Skeletal Methods of Shape Manipulation. In *Proceedings of International Conference on Shape Modeling and Applications*, pages 44–49, Los Alamitos, CA, 1999. IEEE Computer Society, IEEE Computer Society Press.

[Blu67] Harry Blum. A Transformation for Extracting New Descriptors of Shape. In W. Wathen-Dunn, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, Cambridge, MA, 1967.

[BMS00] Silvia Biasotti, Michela Mortara, and Michela Spagnuolo. Surface Compression and Reconstruction Using Reeb Graphs and Shape Analysis. In Bianca Falcidieno, editor, *Proceedings of Spring Conference on Computer Graphics*, pages 175–184. Comenius University, Bratislava, 2000.

[BO00] Alexander G. Belyaev and Yutaka Ohtake. An Image Processing Approach to Detection of Ridges and Ravines on Polygonal Surfaces. In *Proceedings of Eurographics 2000, Short Presentations*, pages 19–28, 2000.

[BR94] Roger A. Browse and James C. Rodger. Investigations of Three-Dimensional Shape Perception for Telepresence Using Superquadric Primitives. In Bernice E. Rogowitz and Jan P. Allebach, editors, *Proceedings of Human Vision, Visual Processing, and Digital Display V (San Jose, CA, February 1994)*, volume 2179 of *SPIE proceedings*, pages 235–246. SPIE—The International Society for Optical Engineering, 1994.

[Bre03] Angela Brennecke. Skelett-Graphen-Vergleich. Internship report, Otto-von-Guericke University of Magdeburg, 2003.

[BS00] John W. Buchanan and Mario C. Sousa. The Edge Buffer: A Data Structure for Easy Silhouette Rendering. In Stephen N. Spencer, editor, *Proceedings of First International Symposium on Non Photorealistic Animation and Rendering (Annecy, France, June 5–7, 2000)*, pages 39–42, New York, 2000. ACM SIGGRAPH / Eurographics, ACM Press.

[BSBK02] Mario Botsch, Stephan Steinberg, Stephan Bischoff, and Leif Kobbelt. OpenMesh – A Generic and Efficient Polygon Mesh Data Structure. In *OpenSG Symposium 2002*, 2002.

[Car99] M. Sheelagh T. Carpendale. *A Framework for Elastic Presentation Space.* PhD thesis, Simon Fraser University, Burnaby, British Columbia, 1999.

[CM02] Drew Card and Jason L. Mitchell. Non-Photorealistic Rendering with Pixel and Vertex Shaders. In Wolfgang F. Engel, editor, *Vertex and Pixel Shaders Tips and Tricks*, chapter 3, pages 319–333. Wordware Publishing, 2002.

[CO02] Ding-Yun Chen and Ming Ouhyoung. A 3D Object Retrieval System Based on Multi-Resolution Reeb Graph. In *Proceedings of Computer Graphics Workshop*, page 16, Tainan, Taiwan, 2002.

[dC76] Manfredo P. do Carmo. *Differential Geometry of Curves and Surfaces.* Prentice Hall, Englewood Cliffs, NJ, 1976.

[DHR$^+$99] Oliver Deussen, Jörg Hamel, Andreas Raab, Stefan Schlechtweg, and Thomas Strothotte. An Illustration Technique Using Hardware-Based Intersections and Skeletons. In I. S. MacKenzie and J. Stewart, editors, *Proceedings of Graphics Interface'99 (Kingston, Ontario, June, 1999)*, pages 175–182. Canadian Human-Computer Communications Society, Morgan Kaufmann Publishers Inc., 1999.

[DS00] Oliver Deussen and Thomas Strothotte. Computer-Generated Pen-and-Ink Illustration of Trees. In Kurt Akeley, editor, *Proceedings of SIGGRAPH 2000 (New Orleans, LA, July 23–28, 2000)*, *Computer Graphics* Proceedings, Annual Conference Series, pages 13–18, New York, 2000. ACM SIGGRAPH.

[Ebe96] David Eberly. *Ridges in Image and Data Analysis.* Series on Computational Imaging and Vision. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996.

[EC90] Gershon Elber and Elaine Cohen. Hidden Curve Removal for Free Form Surfaces. In Forest Baskett, editor, *Proceedings of SIGGRAPH 90 (Dallas, TX, August 6–10, 1990)*, *Computer Graphics* Proceedings, Annual Conference Series, pages 95–104, New York, 1990. ACM SIGGRAPH, ACM Press.

[EGHHZ00] Alon Efrat, Leonidas J. Guibas, Olaf A. Hall-Holt, and Li Zhang. On Incremental Rendering of Silhouette Maps of a Polyhedral Scene. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 910–917, San Francisco, CA, 2000.

[EM94] Herbert Edelsbrunner and Ernst P. Mücke. Three-Dimensional Alpha Shapes. *acm Transactions on Graphics*, 13(1):43–72, January 1994.

[ER02] Michal Etzion and Ari Rappoport. Computing Voronoi Skeletons of a 3-D Polyhedron by Space Subdivision. *Computational Geometry*, 21(3):87–120, March 2002.

[FEdFC02] Ricardo Fabbri, Leandro Farias Estrozi, and Luciano da Fontoura Costa. On Voronoi Diagrams and Medial Axes. *Journal of Mathematical Imaging and Vision*, 17(1):27–40, July 2002.

[Fle80] Rudolf Fleischmann. *Einführung in die Physik.* Physik Verlag, Weinheim, 2nd edition, 1980.

[FM03] Adam Finkelstein and Lee Markosian. Nonphotorealistic Rendering. *IEEE Computer Graphics and Applications*, 23(4):26–27, July/August 2003.

[FMK+03] Thomas Funkhouser, Patrick Min, Michael Kazhdan, Joyce Chen, Alex Halderman, David Dobkin, and David Jacobs. A Search Engine for 3D Models. *acm Transactions on Graphics*, 22(1):83–105, January 2003.

[GG01] Bruce Gooch and Amy Gooch. *Non-Photorealistic Rendering.* A K Peters, Ltd., Natick, 2001.

[GH97] Michael Garland and Paul S. Heckbert. Surface Simplification Using Quadric Error Metrics. In Turner Whitted, editor, *Proceedings of SIGGRAPH 97 (Los Angeles, CA, August 3–8, 1997)*, *Computer Graphics* Proceedings, Annual Conference Series, pages 209–216, Reading, MA, 1997. ACM SIGGRAPH, Addison Wesley Publishing Company.

[GH98] Michael Garland and Paul S. Heckbert. Simplifying Surfaces with Color and Texture using Quadric Error Metrics. In David Ebert, Holly Rushmeier, and Hans Hagen, editors, *IEEE Visualization '98*, pages 263–270, Los Alamitos, CA, 1998. IEEE Computer Society, IEEE Computer Society Press.

[Gie03] Johan Gielis. A Generic Geometric Transformation that Unifies a Wide Range of Natural and Abstract Shapes. *American Journal of Botany*, 90(3):333–338, March 2003.

[GKHS98] Nikhil Gagvani, Deepak R. Kenchammana-Hosekote, and Deborah Silver. Volume Animation Using the Skeleton Tree. In *Proceedings of the 1998 IEEE Symposium on Volume Visualization*, pages 47–53. ACM Press, 1998.

[GKV89] Christian Gerthsen, Hans O. Kneser, and Helmut Vogel. *Physik*. Springer-Verlag, Berlin · Heidelberg · New York, 16th edition, 1989.

[Gla91] Andrew S. Glassner. Maintaining Winged-Edge Models. In James Arvo, editor, *Graphic Gems II*, chapter IV.6, pages 191–201. Academic Press, Inc., 1991.

[GNIS04] Marcel Götze, Petra Neumann, Tobias Isenberg, and Thomas Strothotte. User Supported Interactive Illustration of Text. 2004. Submitted.

[Gol01] Jack Goldfeather. Understanding Errors in Approximating Principal Direction Vectors. Technical Report 01-006, University of Minnesota – Computer Science and Engineering, 2001.

[Gol02] E. Bruce Goldstein. *Sensation and Perception*. Wadsworth Group, Pacific Grove, CA, 6th edition, 2002.

[GSG$^+$99] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley, and Richard Riesenfeld. Interactive Technical Illustration. In *1999 ACM Symposium on Interactive 3D Graphics*, pages 31–38, New York, 1999. ACM.

[Hal03] Nick Halper. *Supportive Presentation for Computer Games*. PhD thesis, Otto-von-Guericke University of Magdeburg, Magdeburg, October 2003.

[HBK01] Masayuki Hisada, Alexander G. Belyaev, and Tosiyasu L. Kunii. A 3D Voronoi-based Skeleton and Associated Surface Features. In *Proceedings of the Ninth Pacific Conference on Computer Graphics and Applications (October 16–18, 2001, Tokyo)*, pages 89–96, Los Alamitos, CA, 2001. IEEE Computer Society, IEEE Computer Society Press.

[HBK02] Masayuki Hisada, Alexander G. Belyaev, and Tosiyasu L. Kunii. A Skeleton-based Approach for Detection of Perceptually Salient Features on Polygonal Surfaces. *Computer Graphics Forum*, 21(4):689–700, December 2002.

[HDD$^+$93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDondald, and Werner Stuetzle. Mesh Optimization. In James T. Kajiya, editor, *Proceedings of SIGGRAPH 93 (Anaheim, CA, August 1–6, 1993)*, *Computer Graphics* Proceedings, Annual Conference Series, pages 19–26, New York, 1993. ACM SIGGRAPH, ACM Press.

[Her99] Aaron Hertzmann. Introduction to 3D Non-Photorealistic Rendering: Silhouettes and Outlines. In Stuart Green, editor, *Non-Photorealistic Rendering*, volume 17 of *SIGGRAPH 99 Course Notes*, chapter 7. ACM Press, New York, 1999.

[HG01] Andreas G. P. Hubeli and Markus H. Gross. Multiresolution Feature Extraction from Unstructured Meshes. In Thomas Ertl, Kenneth I. Joy, and Amitabh Varshney, editors, *Proceedings of the IEEE Visualization 2001 (October 21–26, 2001, San Diego, California)*, pages 287–294, Los Alamitos, CA, 2001. IEEE Computer Society, IEEE Computer Society Press.

[HIR$^+$03] Nick Halper, Tobias Isenberg, Felix Ritter, Bert Freudenberg, Oscar Meruvia, Stefan Schlechtweg, and Thomas Strothotte. OpenNPAR: A System for Developing, Programming, and Designing Non-Photorealistic Animation and Rendering. In Jon Rokne, Reinhard Klein, and Wenping Wang, editors, *Proceedings of Pacific Graphics 2003*, pages 424–428, Los Alamitos, CA, 2003. IEEE Computer Society, IEEE. Short paper and poster.

[HLG01] Kai Hormann, Ulf Labsik, and Günther Greiner. Remeshing Triangulated Surfaces with Optimal Parameterizations. *Computer-Aided Design*, 33(11):779–788, September 2001.

[HMG00] Andreas G. P. Hubeli, Kuno Meyer, and Markus H. Gross. Mesh Edge Detection. In *Workshop Lake Tahoe (Lake Tahoe City, California, USA, October 15–17, 2000)*, 2000.

[HMH$^+$03] Nick Halper, Mara Mellin, Christoph S. Herrmann, Volker Linneweber, and Thomas Strothotte. Psychology and Non-Photorealistic Rendering: The Beginning of a Beautiful Relationship. In Jürgen Ziegler and Gerd Szwillus, editors, *Mensch & Computer 2003: Interaktion in Bewegung (September 8–10, 2003, Stuttgart)*, pages 277–286, Stuttgart, Leipzig, Wiesbaden, 2003. Teubner Verlag.

[Hod89] Elaine R. S. Hodges, editor. *The Guild Handbook of Scientific Illustration.* Van Nostrand Reinhold, New York, 1989.

[Hop96] Hugues Hoppe. Progressive Meshes. In Holly Rushmeier, editor, *Proceedings of SIGGRAPH 96 (New Orleans, LA, August 4–9, 1996)*, *Computer Graphics* Proceedings, Annual Conference Series, pages 99–108, Reading, MA, 1996. ACM SIGGRAPH, Addison Wesley Publishing Company.

[HS02] Knut Hartmann and Thomas Strothotte. A Spreading Activation Approach to Text Illustration. In Andreas Butz, Antonio Krüger, Patrick Olivier, Stefan Schlechtweg, and Michele Zhou, editors, *Proceedings of the 2nd International Symposium on Smart Graphics (June 11-13, 2002, Hawthorne, NY, USA)*, pages 39–46, New York, 2002. ACM Press.

[HSEP00] Horst K. Hahn, Dirk Selle, Carl J. G. Evertsz, and Heinz-Otto Peitgen. Interaktive Visualisierung von Gefäßsystemen auf der Basis von Oberflächenprimitiven. In Thomas Schulze, Peter Lorenz, and Volkmar Hinz, editors, *Simulation und Visualisierung 2000*, pages 105–118, Ghent, Belgium, 2000. SCS – The Society for Computer Simulation International, SCS-Europe BVBA.

[HSKK01] Masaki Hilaga, Yoshihisa Shinagawa, Taku Kohmura, and Tosiyasu L. Kunii. Topology Matching for Fully Automatic Similarity Estimation of 3D Shapes. In Eugene Fiume, editor, *Proceedings of SIGGRAPH 2001 (Los Angeles, CA, August 12–17, 2001)*, *Computer Graphics* Proceedings, Annual Conference Series, pages 203–212, New York, 2001. ACM SIGGRAPH, ACM Press.

[HSS02] Nick Halper, Stefan Schlechtweg, and Thomas Strothotte. Creating Non-Photorealistic Images the Designer's Way. In Stephen N. Spencer, editor, *Proceedings of Second International Symposium on Non Photorealistic Animation and Rendering (Annecy, France, June 3–5, 2002)*, pages 97–104, New York, 2002. ACM SIGGRAPH / Eurographics, ACM Press.

[Huy90] Christiaan Huygens. *Traité de la Lumière*. Leyden, 1690. Presented in Paris to the Académie Royale des Sciences in 1678.

[HZ00] Aaron Hertzmann and Denis Zorin. Illustrating Smooth Surfaces. In Kurt Akeley, editor, *Proceedings of SIGGRAPH 2000 (New Orleans, LA, July 23–28, 2000)*, *Computer Graphics* Proceedings, Annual Conference Series, pages 517–526, New York, 2000. ACM SIGGRAPH.

[IFH$^+$03] Tobias Isenberg, Bert Freudenberg, Nick Halper, Stefan Schlechtweg, and Thomas Strothotte. A Developer's Guide to Silhouette Algorithms for Polygonal Models. *IEEE Computer Graphics and Applications*, 23(4):28–37, July/August 2003.

[IFP95] Victoria Interrante, Henry Fuchs, and Stephen Pizer. Enhancing Transparent Skin Surfaces with Ridge and Valley Lines. In *Proceedings of 6th IEEE Visualization 1995 Conference (VIS'95, October 29–November 03, 1995, Atlanta, Georgia)*, pages 52–60, Los Alamitos, CA, 1995. IEEE Computer Society, IEEE Computer Society Press.

[IH03] Takeo Igarashi and John F. Hughes. Smooth Meshes for Sketch-based Freeform Modeling. In *ACM SIGGRAPH 2003 Symposium on Interactive 3D Graphics (ACM I3D'03, Monterey, California, April 27–30, 2003)*, pages 139–142, New York, 2003. ACM Press.

[IHK03] Tobias Isenberg, Knut Hartmann, and Henry König. Interest Value Driven Adaptive Subdivision. In Thomas Schulze, Stefan Schlechtweg, and Volkmar Hinz, editors, *Simulation und Visualisierung 2003*, pages 139–149, Erlangen, San Diego, 2003. Otto-von-Guericke University of Magdeburg, SCS European Publishing House.

[IHS02] Tobias Isenberg, Nick Halper, and Thomas Strothotte. Stylizing Silhouettes at Interactive Rates: From Silhouette Edges to Silhouette Strokes. *Computer Graphics Forum (Proceedings of Eurographics)*, 21(3):249–258, September 2002.

[IMT99] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A Sketching Interface for 3D Freeform Design. In Alyn Rockwood, editor, *Proceedings of SIGGRAPH 99 (Los Angeles, CA, August 8–13, 1999)*, *Computer Graphics* Proceedings, Annual Conference Series, pages 409–416, New York, 1999. ACM SIGGRAPH, Addison Wesley Publishing Company.

[Jen01] Peter Jenny. *Notizen zur Figuration.* Verlag Hermann Schmidt, Mainz, 2nd edition, 2001.

[JI03] Roland Jesse and Tobias Isenberg. Use of Hybrid Rendering Styles for Presentation. In *Proceedings of WSCG 2003*, 2003. Short Paper.

[JINS04] Roland Jesse, Tobias Isenberg, Bernd Nettelbeck, and Thomas Strothotte. Dynamics by Hybrid Combination of Photorealistic and Non-Photorealistic Rendering Styles. Technical Report 5/2004, Department of Computer Science, University of Magdeburg, Germany, 2004.

[Joh02] Scott F. Johnston. Lumo: Illumination for Cel Animation. In Stephen N. Spencer, editor, *Proceedings of Second International Symposium on Non Photorealistic Animation and Rendering (Annecy, France, June 3–5, 2002)*, pages 45–52, New York, 2002. ACM SIGGRAPH / Eurographics, ACM Press.

[KBB+00] Leif P. Kobbelt, Stephan Bischoff, Mario Botsch, Kolja Kähler, Christian Rössl, Robert Schneider, and Jens Vorsatz. Geometric Modeling Based on Polygonal Meshes. Research Report MPI-I-2000-4-002, Max-Planck-Institut für Informatik, Saarbrücken, July 2000.

[KDMF03] Robert D. Kalnins, Philip L. Davidson, Lee Markosian, and Adam Finkelstein. Coherent Stylized Silhouettes. *acm Transactions on Graphics, Proceedings of ACM SIGGRAPH 2003*, 22(3):856–861, July 2003.

[Koe84] Jan J. Koenderink. What Does the Occluding Contour Tell Us About Solid Shape. *Perception*, 13:321–330, 1984.

[Koe90] Jan J. Koenderink. *Solid Shape.* MIT Press, Cambridge, MA, 1990.

[KSK98] Reinhard Klette, Karsten Schlüns, and Andreas Koschan. *Computer Vision: Three-Dimensional Data from Images.* Springer-Verlag, Berlin · Heidelberg · New York, 1998.

[Kun99] Tosiyasu Laurence Kunii. Computational Shape Modeling: Valid vs. Invalid. In *International Conference on Shape Modeling and Applications*, pages 2–9, Los Alamitos, CA, 1999. IEEE Computer Society, IEEE Computer Society Press.

[KvD92] Jan J. Koenderink and Andrea J. van Doorn. Surface Shape and Curvature Scales. *Image and Vision Computing*, 10(8):557–565, October 1992.

[LC87] William E. Lorensen and Harvey E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In Maureen C. Stone, editor, *Proceedings of SIGGRAPH 87 (Anaheim, CA, July 27–31, 1987)*, *Computer Graphics* Proceedings, Annual Conference Series, pages 163–169, New York, 1987. ACM SIGGRAPH, ACM Press.

[Lév91] Lorraine Lévy. *Picasso.* Henry Holt & Co., New York, NY, 1991.

[LL02] Yunjin Lee and Seungyong Lee. Geometric Snakes for Triangular Meshes. *Computer Graphics Forum (Proceedings of Eurographics 2002)*, 21(3):229–238, September 2002.

[LLS92] Louisa Lam, Seong-Whan Lee, and Ching Y. Suen. Thinning Methodologies—A Comprehensive Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(9):869–885, September 1992.

[LLS93] Seong-Whan Lee, Louisa Lam, and Ching Y. Suen. A Systematic Evaluation of Skeletonization Algorithms. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(5):1203–1225, 1993.

[Loh98] Gabriele Lohmann. *Volumetric Image Analysis.* John Wiley & Sons and B. G. Teubner, Chichester, West Sussex, England and Stuttgart, Germany, 1998.

[Loo87] Charles T. Loop. Smooth Subdivision Surfaces Based on Triangles. Master's thesis, University of Utah, Department of Mathematics, 1987.

[Lov02] Jörn Loviscach. Rendering Artistic Line Drawings Using Off-the-Shelf 3-D Software. In Isabel Navazo Alvaro and Philipp Slusallek, editors, *Proceedings of Eurographics: Short Presentations*, pages 125–130, Oxford, UK, 2002. Eurographics Association, Blackwell Publishers.

[LPRM02] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least Squares Conformal Maps for Automatic Texture Atlas Generation. *acm Transactions on Graphics, Proceedings of ACM SIGGRAPH 2002*, 21(3):362–371, July 2002.

[LS95] Louisa Lam and Ching Y. Suen. An Evaluation of Parallel Thinning Algorithms for Character Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(9):914–919, September 1995.

[Mar00] Lee Markosian. *Art-based Modeling and Rendering for Computer Graphics.* PhD thesis, Brown University, May 2000.

[MBC02] Jason L. Mitchell, Chris Brennan, and Drew Card. Real-Time Image-Space Outlining for Non-Photorealistic Rendering. In *SIGGRAPH 2002 Conference Abstracts and Applications*, page 239, New York, 2002. ACM SIGGRAPH.

[MCF02] Patrick Min, Joyce Chen, and Thomas Funkhouser. A 2D Sketch Interface for a 3D Model Search Engine. In *SIGGRAPH 2002 Conference Abstracts and Applications*, page 138, New York, 2002. ACM SIGGRAPH.

[Mil63] John W. Milnor. *Morse Theory.* Princeton University Press, New Jersey, 1963.

[MKT$^+$97] Lee Markosian, Michael A. Kowalski, Samuel J. Trychin, Lubomir D. Bourdev, Daniel Goldstein, and John F. Hughes. Real-Time Nonphotorealistic Rendering. In Turner Whitted, editor, *Proceedings of SIGGRAPH 97 (Los Angeles, CA, August 3–8, 1997)*, *Computer Graphics* Proceedings, Annual Conference Series, pages 415–420, Reading, MA, 1997. ACM SIGGRAPH, Addison Wesley Publishing Company.

[MSS97] Maic Masuch, Stefan Schlechtweg, and Bert Schönwälder. daLi! – Drawing Animated Lines! In Oliver Deussen and Peter Lorenz, editors, *Simulation und Animation '97*, pages 87–95, Delft, Belgium, 1997. SCS–Society for Computer Simulation Int., SCS Europe.

[MSS98] Maic Masuch, Lars Schumann, and Stefan Schlechtweg. Animating Frame-to-Frame-Coherent Line Drawings for Illustrative Purposes. In Peter Lorenz and Bernhard Preim, editors, *Simulation und Visualisierung '98*, pages 101–112, Delft, Belgium, 1998. SCS–Society for Computer Simulation Int., SCS Europe.

[MWO03] Wan-Chun Ma, Fu-Che Wu, and Ming Ouhyoung. Skeleton Extraction of 3D Objects with Radial Basis Functions. In *Proceedings of Shape Modeling International 2003 (SMI'03, May 2003, Soul, Korea)*, pages 207–215, 295, Los Alamitos, CA, 2003. IEEE Computer Society Press.

[Neu96] Peter J. Neugebauer. Scanning and Reconstruction of Work Pieces from Range Images. In *2nd IFIP 5.10 Workshop on Virtual Prototyping (Arlington, Texas, May 6–8, 1996).* Automation and Robotics Research Institute, 1996.

[NM00] J. D. Northrup and Lee Markosian. Artistic Silhouettes: A Hybrid Approach. In Stephen N. Spencer, editor, *Proceedings of First International Symposium on Non Photorealistic Animation and Rendering (Annecy,* *France, June 5–7, 2000)*, pages 31–37, New York, 2000. ACM SIGGRAPH / Eurographics, ACM Press.

[OFCD02] Robert Osada, Thomas Funkhouser, Bernard Chazelle, and David Dobkin. Shape Distributions. *acm Transactions on Graphics*, 21(4):807–832, October 2002.

[PBD$^+$01] Mihai Pop, Gill Barequet, Christian A. Duncan, Michael T. Goodrich, W. Huang, and S. Kumar. Efficient Perspective-Accurate Silhouette Computation. In *Proceedings of the 17th Annual ACM Symposium on Computational Geometry*, pages 60–68, Medford, MA, 2001.

[PRS97] Bernhard Preim, Andreas Raab, and Thomas Strothotte. Coherent Zooming of Illustrations with 3D-Graphics and Text. In W. A. Davis, M. Mantei, and R. V. Klassen, editors, *Proceedings of Graphics Interface '97*, pages 105–113. Canadian Information Processing Society, 1997.

[QGRP01] William R. Quadros, B. Gurumoorthy, Krishnan Ramaswami, and Fritz B. Prinz. Skeletons for Representation and Reasoning in Engineering Applications. *Engineering with Computers*, 17(2):186–198, 2001.

[Raa98] Andreas Raab. *Techniken zur Exploration und Visualisierung geometrischer Modellen.* PhD thesis, Otto-von-Guericke University of Magdeburg, Magdeburg, 1998. Published at Shaker-Verlag, Aachen, 2001.

[Ras01] Ramesh Raskar. Hardware Support for Non-photorealistic Rendering. In *2001 SIGGRAPH / Eurographics Workshop on Graphics Hardware (August 2001)*, pages 41–47. ACM Press, 2001.

[RBR01] Chris Raymaekers, Koen Beets, and Frank Van Reeth. Fast Haptic Rendering of Complex Objects Using Subdivision Surfaces. In Karl Reinig, editor, *Proceedings of the Sixth PHANTOM Users Group Workshop (Aspen, Colorado, October 27–30, 2001)*, 2001.

[RC99] Ramesh Raskar and Michael Cohen. Image Precision Silhouette Edges. In Stephen N. Spencer, editor, *Proceedings of 1999 ACM Symposium on Interactive 3D Graphics*, pages 135–140, New York, 1999. ACM Press.

[Ree46] Georges Reeb. Sur le points singuliers d'une forme de Pfaff complètement intégrable ou d'une foncion numérique [On the Singular Points of a Completely Integrable Pfaff Form or of a Numerical Function]. *Comptes Randus Acad. Sciences Paris*, 222:847–849, 1946.

[RG03] Princeton Shape Retrieval and Analysis Group. Princeton 3D Model Search Engine. Web site, August 2003. `http://shape.cs.princeton.edu/search.html`.

[RK99] Christian Rössl and Leif Kobbelt. Approximation and Visualization of Discrete Curvature on Triangulated Surfaces. In *Vision, Modeling, and*

Visualization (VMV) '99 Proceedings, pages 339–346, Erlangen, Germany, 1999. SFB 603, Graduate Research Center, infix.

[RKS00] Christian Rössl, Leif Kobbelt, and Hans-Peter Seidel. Extraction of Feature Lines on Triangulated Surfaces using Morphological Operators. In Andreas Butz, Antonio Krüger, and Patrick Olivier, editors, *Smart Graphics 2000, AAAI Spring Symposium, Stanford University*, pages 71–75, Menlo Park, California, 2000. AAAI – American Association for Artificial Intelligence, AAAI Press. Technical Report SS-00-04.

[Röb03] Niklas Röber. Visualization of Fuel Cell Simulations. Diplom thesis at the Department of Simulation and Graphics, Otto-von-Guericke University of Magdeburg, Magdeburg, 2003.

[Rös99] Christian Rössl. Semi-Automatische Methoden für die Rekonstruktion von CAD-Modellen aus Punktdaten. Master's thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 1999.

[RR96] Rémi Ronfard and Jarek Rossignac. Full-range Approximation of Triangulated Polyhedra. *Computer Graphics Forum*, 15(3):67–76, 1996.

[Rub03] Jason Rubin. Great Game Graphics...Who Cares? Keynote speech at Games Developer Conference 2003 (GDC 2003, March 6–8, 2003, San Jose, CA), August 2003.

[Rus89] Pramod Rustagi. Silhouette Line Display From Shaded Models. *IRIS Universe*, 1989(9):42–44, Fall 1989.

[RvE92] Jarek R. Rossignac and Maarten van Emmerik. Hidden Contours on a Frame-Buffer. In *Proceedings of the 7th Eurographics Workshop on Computer Graphics Hardware (Cambridge, UK, September 1992)*, pages 188–204, 1992.

[Sel99] Dirk Selle. *Analyse von Gefäßstrukturen in medizinischen Schichtdatensätzen für die computergestützte Operationsplanung.* PhD thesis, Universität Bremen, September 1999.

[Sel00] Dirk Selle. *Analyse von Gefäßstrukturen in medizinischen Schichtdatensätzen für die computergestützte Operationsplanung.* Berichte aus der Medizinischen Informatik und Bioinformatik. Shaker-Verlag, Aachen, 2000.

[SG81] Stuart Sechrest and Donald P. Greenberg. A Visible Polygon Reconstruction Algorithm. In Henry Fuchs, editor, *Proceedings of SIGGRAPH 81 (Dallas, TX, August 3–7, 1981)*, *Computer Graphics* Proceedings, Annual Conference Series, pages 17–27, New York, 1981. ACM SIGGRAPH, ACM Press.

[SGG$^+$00] Pedro V. Sander, Xianfeng Gi, Steven J. Gortler, Huges Hoppe, and John Snyder. Silhouette Clipping. In Kurt Akeley, editor, *Proceedings of SIGGRAPH 2000 (New Orleans, LA, July 23–28, 2000)*, *Computer Graphics* Proceedings, Annual Conference Series, pages 327–334, New York, 2000. ACM SIGGRAPH, ACM Press.

[SGP93] Atul Sudhalkar, Levent Gürsöz, and Fritz Prinz. Continuous Skeletons of Discrete Objects. In *Proceedings on the Second ACM Symposium on Solid Modeling and Applications (Montreal, Quebec, Canada)*, pages 85–94, New York, NY, USA, 1993. ACM Press.

[She68] Donald Shepard. A Two-Dimensional Function for Irregularly Spaced Data. In *Proceedings of the 23rd ACM National Conference*, pages 517–524, 1968.

[SIDS03] Henry Sonnet, Tobias Isenberg, Jana Dittmann, and Thomas Strothotte. Illustration Watermarks for Vector Graphics. In Jon Rokne, Reinhard Klein, and Wenping Wang, editors, *Proceedings of Pacific Graphics 2003*, pages 73–82, Los Alamitos, CA, 2003. IEEE Computer Society, IEEE.

[SKK91] Yoshihisa Shinagawa, Tosiyasu L. Kunii, and Yannick L. Kergosien. Surface coding based on morse theory. *IEEE Computer Graphics and Applications*, 11(5):66–78, September/October 1991.

[SPB95] Evan C. Sherbrooke, Nicholas M. Patrikalakis, and Erik Brisson. Computation of the Medial Axis Transform of 3-D Polyhedra. In *Proceedings of the Third ACM Symposium on Solid Modeling and Applications (Salt Lake City, Utah)*, pages 187–200, Salt Lake City, Utah, USA, 1995. ACM Press.

[SPR$^+$94] Thomas Strothotte, Bernhard Preim, Andreas Raab, Jutta Schumann, and David R. Forsey. How to Render Frames and Influence People. *Computer Graphics Forum (Proceedings of Eurographics 1994)*, 13(3):455–466, September 1994.

[SPSP02] Dirk Selle, Bernhard Preim, Andrea Schenk, and Heinz-Otto Peitgen. Analysis of Vasculature for Liver Surgical Planning. *IEEE Transactions on Medical Imaging*, 21(11):1344–1357, November 2002.

[SS00] Stefan Schlechtweg and Thomas Strothotte. Generating Scientific Illustrations in Digital Books. In *Smart Graphics. Papers from the 2000 AAAI Spring Symposium (Stanford, March 20–22, 2000)*, pages 8–15, Menlo Park, 2000. AAAI Press.

[SS02] Thomas Strothotte and Stefan Schlechtweg. *Non-Photorealistic Computer Graphics: Modeling, Rendering, and Animation.* Morgan Kaufmann, San Francisco, 2002.

[SSRL96] Jutta Schumann, Thomas Strothotte, Andreas Raab, and Stefan Laser. Assessing the Effect of Non-photorealistic Rendered Images in CAD. In *Proceedings of CHI'96*, pages 35–42, New York, 1996. ACM SIGCHI, ACM Press.

[SSS74] Ivan E. Sutherland, Robert F. Sproull, and Robert A. Schumacker. A Characterization of Ten Hidden-Surface Algorithms. *Computing Surveys*, 6(1):1–55, March 1974.

[ST90] Takafumi Saito and Tokiichiro Takahashi. Comprehensible Rendering of 3-D Shapes. In Forest Baskett, editor, *Proceedings of SIGGRAPH 90 (Dallas, TX, August 6–10, 1990)*, *Computer Graphics* Proceedings, Annual Conference Series, pages 197–206, New York, 1990. ACM SIGGRAPH, ACM Press.

[Teu03] Christian Teutsch. Optimierte Rekonstruktion von 3D-Geometrien aus Laserscan-Daten unter Ausnutzung von Information über den Scan-Prozess. Diplom thesis at the Department of Simulation and Graphics, Otto-von-Guericke University of Magdeburg, Magdeburg, 2003.

[TG92] Jean-Philippe Thirion and Alexis Gourdon. The 3D Marching Lines Algorithm and its Application to Crest Lines Extraction. Technical Report 1672, INRIA, Rocquencourt, France, May 1992.

[TG93] Jean-Philippe Thirion and Alexis Gourdon. The Marching Lines Algorithm: New Results and Proofs. Technical Report 1881, INRIA, Rocquencourt, France, April 1993.

[TG96] Jean-Philippe Thirion and Alexis Gourdon. The 3D Marching Lines Algorithm. *Graphical Models and Image Processing*, 58(6):503–509, November 1996.

[TITW04] Christian Teutsch, Tobias Isenberg, Erik Trostmann, and Michael Weber. Evaluation and Optimization of Laser Scan Data. In Thomas Schulze, Stefan Schlechtweg, and Volkmar Hinz, editors, *Simulation und Visualisierung 2004*, pages 311–322, Erlangen, San Diego, 2004. SCS European Publishing House.

[Tra94] Wes Trager. A Practical Approach to Motion Capture: Acclaim's Optical Motion Capture System. In *Character Animation Systems*, volume 9 of *SIGGRAPH 94 Course Notes*. ACM SIGGRAPH, 1994.

[TT98] Marek Teichmann and Seth Teller. Assisted Articulation of Closed Polygonal Models. In *Proceedings of 1998 Eurographics Workshop on Animation and Simulation*, pages 87–101, 1998.

[Vel93] Remco C. Veltkamp. 3D Computational Morphology. *Computer Graphics Forum*, 12(3):115–127, 1993.

[Vel94] Remco C. Veltkamp. *Closed Object Boundaries from Scattered Points.* PhD thesis, Centum voor Wiskunde en Informatica, Department of Interactive Systems, 1994.

[Vie99] Viewpoint Premier Catalog, 2000 Edition. 3D Model Catalog on CD-ROM. Viewpoint Digital, Inc., 1999.

[VL97a] Anne Verroust and Francis Lazarus. Extracting Skeletal Curves from 3D Scattered Data. *The Visual Computer*, 16(1):15–25, 1997.

[VL97b] Anne Verroust and Francis Lazarus. Extracting Skeletal Curves from 3D Scattered Data. Technical Report 3250, INRIA, Rocquencourt, France, September 1997.

[vO95] Cornelius W. A. M. van Overveld. Pondering on Discrete Smoothing and Interpolation. *Computer-Aided Design*, 27(5):377–384, May 1995.

[Wad00] Lawson Wade. *Automated Generation of Control Skeletons for Use in Animation.* PhD thesis, The Ohio State University, 2000.

[WB01] Kouki Watanabe and Alexander G. Belyaev. Detection of Salient Curvature Features on Polygonal Surfaces. *Computer Graphics Forum*, 20(3):385–392, September 2001.

[WNV00] Onno Wink, Wiro J. Niessen, and Max A. Viergever. Fast Delineation and Visualization of Vessels in 3D Angiographic Images. *IEEE Transactions on Medical Imaging*, 19(4):337–346, April 2000.

[WP00] Lawson Wade and Richard E. Parent. Fast, Fully Automated Generation of Control Skeletons for Use in Animation. In *Proceedings of the Computer Animation 2000 (CA'00; May 03–05, 2000, Philadelphia, Pennsylvania)*, pages 164–169, Los Alamitos, CA, 2000. IEEE Computer Society, IEEE Computer Society Press.

[WP02] Lawson Wade and Richard E. Parent. Automated Generation of Control Skeletons for Use in Animation. *The Visual Computer*, 18(2):97–110, March 2002.

[ZISS04] Johannes Zander, Tobias Isenberg, Stefan Schlechtweg, and Thomas Strothotte. High Quality Hatching. *Computer Graphics Forum (Proceedings of Eurographics)*, 23(3), September 2004. To appear.

[ZSD$^+$00] Denis Zorin, Peter Schröder, Tony DeRose, Leif P. Kobbelt, Adi Levin, and Wim Sweldens, editors. *Subdivision for Modeling and Animation*, volume 23 of *SIGGRAPH 2000 Course Notes.* ACM SIGGRAPH, 2000.

[ZT99] Yong Zhou and Arthur W. Tonga. Efficient Skeletonization of Volumetric Objects. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):196–209, July–September 1999.

List of Figures

1.1 "The Bull." Part of the lithography series by PICASSO. From [Lév91]. . 1

3.1 Example Voronoi diagram in 2D for a small set of points. 18
3.2 Deriving the medial axis of a shape. The medial axis is denoted by the dotted line. From [Blu67]. 19
3.3 Examples for medial axes of shapes. The medial axes are denoted by the dotted lines. From [Blu67]. 19
3.4 Medial axes of polygonal shapes in 2D. Some of the maximal balls are shown together with their respective centers. 20
3.5 Comparison of medial axes of different polygonal approximations of an ellipse and the exact ellipse. 22
3.6 The straight skeleton of a concave polygon. Note that there are no curved parts in the skeleton. From [AAAG95b]. 23
3.7 Example for the medial axis of a three-dimensional shape. 23
3.8 Deriving a Reeb graph of a torus. 25
3.9 Deriving a Multi-resolution Reeb Graph. From [HSKK01]. 25
3.10 Color-coded geodesic distance for two similar objects used for multi-resolution Reeb graph shape matching. From [HSKK01]. 26
3.11 Using an extended Reeb graph for shape compression. From [BMS00]. . 27
3.12 Original shape and selected results (using ARCEL, HILDI, and PAVLI) from LEE, LAM, and SUEN's survey of skeletonization algorithms. From [LLS93]. 29
3.13 Example for computing the skeleton from the volumetric data set of a brain. From [ZT99]. 30
3.14 Influence of the threshold t on the generated internal skeleton using SELLE's algorithm. From [SPSP02]. 31
3.15 Using skeletonization to determine which parts of the liver are supplied by which branch of the portal vein. From [SPSP02]. 31
3.16 Removal of artifacts in the skeleton at bifurcations. From [HSEP00]. . . 32
3.17 Visualization of the skeleton using frusta. From [HSEP00]. 32
3.18 Shapes and their skeletons as computed by TEICHMANN and TELLER. From [TT98]. 34
3.19 Constructing the skeleton using WADE and PARENT's algorithm. From [Wad00]. 35
3.20 Edge collapse operation. 36

3.21 Example for successive edge collapse operations resulting in the skeletonization of the polygonal approximation of a cylinder. 37
3.22 Model of a bicycle and the skeleton that was extracted from it using successive edge collapse operations. 37
3.23 Example for the application of the skeleton extraction algorithm in non-photorealistic rendering. From [DHR+99]. 38
3.24 Computing internal skeletons using RBFs and snakes. From [MWO03]. 40
3.25 Examples for the approach based on radial basis functions and snakes for skeleton extraction. The red spheres denote the maxima clusters and the green spheres the location of the snakes after minimizing their energy. Courtesy of Wan-Chun Ma, from [MWO03]. 40
3.26 Computing a skeleton from a point cloud. From [VL97a]. 41
3.27 Examples for skeleton extraction from point sets. From [VL97b]. 41
3.28 Deriving an external skeleton as presented by Rössl, Kobbelt, and Seidel. From [RKS00]. 43
3.29 *Salient curvature features* extracted by Watanabe and Belyaev. From [WB01]. 44
3.30 Applying strokes to detected ridges to generate a pen-and-ink rendering. From [BO00]. 44
3.31 Extracting surface features using an approximated Voronoi diagram. From [HBK01]. 45
3.32 Examples for feature extraction with *geometric snakes*. From [LL02]. . 46

4.1 Schematic illustration of the network of extreme ridges. 50
4.2 Schematic illustration of the definition of extreme ridges. 51
4.3 Instead of being a set of smooth curves, the network of extreme ridges for polygonal meshes consists of edges of the mesh. 52
4.4 Estimating the principal curvatures using the method by Rössl and Kobbelt. 53
4.5 Deriving an heuristic measure for the curvature at a vertex of a polygonal mesh. 57
4.6 Shape index for certain shapes. From [KvD92]. 58
4.7 Color-coded shape index for some examples (green and blue represent negative while yellow and red represent positive values). From [KvD92]. 59
4.8 Color-coded shape index based DoI values (red shows high values and blue low values). The DoI values have been normalized in order to fill the whole color interval; the DoI values in each image may be normalized differently. 60
4.9 Plot of function $f_1(\mathcal{K})$ in Equation 4.25. 61
4.10 Plot of the function $f_2(\mathcal{K})$ in Equation 4.28. 62
4.11 Color-coded fold indices and the difference image. Although visually both fold indices cannot be distinguished, there are slight differences. However, the differences are less than 2.4% on each of the three 256 index RGB color scales. 62

4.12 Estimation of the principal curvatures at a point other than a mesh vertex by using SHEPARD's interpolation scheme [She68]. 64
4.13 Color-coded examples for the application of the singularity index (red shows high values and blue low values). 65
4.14 Dynamic adaptive subdivision (three steps) for better silhouette generation. In both cases the model is viewed directly from the front. 66
4.15 Static adaptive subdivision for better silhouette generation: adjacent faces with an angle higher than 30 degrees are subdivided. 67
4.16 Triangles that produce perceivable discontinuities in haptic rendering are being subdivided. 68
4.17 Olaf's bracelet is the object of focus. Thus, its triangles and all triangles of objects close to it are subdivided. The closer the triangles are to the focus point the more subdivision steps are used. 69
4.18 Proposed taxonomy for the DoI functions developed and discussed in this chapter. 70
4.19 Color-coded DoI values using some of the DoI functions introduced in this chapter (red shows high values and blue low values). The DoI values have been normalized in order to fill the whole color interval; the DoI values in each image may be normalized differently. 72
4.20 Color-coded DoI functions as used in Figure 4.19 applied to a technical model. 73

5.1 Deriving a line based skeleton from shapes which are tubular or consist of tubular parts. 76
5.2 Reconstructing shapes from an internal edge based skeleton using the quadratic error. As shown in (a) and (b), the reconstruction of tubular objects works well. However, the reconstruction of compact objects does not correspond to the original as well ((c) and (d)). From [Raa98]. . . . 77
5.3 Example for a skeleton created with the edge collapse method where parts are outside of the object. In particular, see the neck and the beak of the dinosaur. 78
5.4 Stabilization of the drift behavior which is caused by a certain succession of edge collapse steps by averaging the final positions of the skeleton vertices. 78
5.5 Sensitivity of the edge collapse based internal skeleton to slight changes in the triangulation of the shapes they are computed for. Being visually equivalent, the two shapes (a) and (c) only differ in the scales of their underlying geometric models. 79
5.6 Problem of edge collapse based skeleton extraction with the genus of the mesh. Since the hole of the disk has short edges compared to the outer edges they are being collapsed first. Thus, instead of the expected skeleton ring a single edge is the output of the algorithm. 80
5.7 Skeletons extracted from segmented models (a) are segmented as well (c). For obtaining a connected skeleton graph as shown in (b) the patches have to be welded together prior to starting the edge collapse process. . 81

5.8 Reducing the number of insignificant local maxima by employing a threshold. The local maxima are indicated by red dots; the fold index was used as the DoI function. 86

5.9 Schematic view of wavefront propagation on a triangular mesh. Wavefront edges are the thick gray edges whereas the inter-wavefront edges are thin gray. The black dots represent vertices with local maxima in the DoI function. The black edges are part of the extreme ridges. . . . 88

5.10 Propagation of wavefronts on the surface of a triangular model, without correction. Note the wrongly marked edges between meeting wavefronts and at merged wavefronts. 88

5.11 Schematic illustration of cases for which regularization of wavefront edges is needed. Regular mesh edges are thin black lines, wavefronts are thick gray lines, and erroneously marked edges are shown as thick black lines. See also examples in Figure 5.10. 89

5.12 Regularization edges for use with the edges in Figure 5.10(b) and resulting regularized wavefront edges. 90

5.13 Potential problem when connecting restricted local maxima and how to avoid them. Wavefronts are visualized by thick gray lines and restricted local maxima by black dots. The proposed connection of the local maxima is shown by thick black lines. 91

5.14 External skeleton consisting of a network of extreme ridges computed for the bunny model. The DoI function used in this example is the fold index with function f_2. 92

5.15 Schematic illustration of the Winged Edge data structure. 94

5.16 Schematic illustration of the neighborhood relations stored in the edge data for every unique edge within the Winged Edge data structure. . . 94

5.17 External skeleton curves computed for a face model. The local extrema are visualized as dots (red for maxima and blue for minima). The insets are the same models viewed from a different angle. 97

5.18 Diagram showing the different possibilities for introducing multi-scale capability to external skeletons. This is achieved by combinations of low-pass filtering the original mesh MS_0 or pruning the set of local maxima LM_i or the external skeleton ES_j. 98

5.19 Growing level of scale of an external skeleton of the same model achieved through pruning LM and ES. The skeleton was computed from the shape shown in Figure 5.17(a) using κ_2 as the DoI function and looking for minima. 99

5.20 Simulating the reconstruction of the external skeleton by employing white noise. 100

5.21 The effect of scale on the reconstruction. The reconstructed shapes were generated from the external skeletons shown in Figure 5.19. 100

5.22 Possible degenerate result when collapsing of edges is not restricted to those starting from the external skeleton. Although the skeleton is fairly densely tessellated, the simplified mesh is degenerated (in this example a different model was used to show the effect because in the face model the sharp edge around the face prevents the effect to some degree). . . 102
5.23 Mesh simplification for preservation of the mesh's topology using edge collapse operations for the external skeletons shown in Figure 5.19. . . 102

6.1 Computing the silhouette using the wave propagation based external skeleton extraction algorithm and the DoI function from Equation 6.2. 109
6.2 Image-space silhouette detection based on edge detection operators on the z-buffer (original image, z-buffer, detected edges, and composed image). From [ST90]. 112
6.3 Image-space silhouette detection using edge detection operators on the z-buffer and the normal buffer (z-buffer, edges extracted from the z-buffer, normal buffer, edges extracted from the normal buffer, and combination of both edge buffers). From [Her99]. 112
6.4 Enlargement of a back-facing polygon to achieve wide silhouette lines. From [RC99]. 115
6.5 Silhouette generated with a hybrid algorithm. From [GSG⁺99]. 115
6.6 Computation of a sub-polygon silhouette for a single triangle and a mesh. The dot product between normal vector and view direction is positive at vertices with plus signs and negative at vertices with minus signs. Between a positive and a negative vertex linear interpolation is used to find the silhouette. From [Her99]. 118
6.7 Two images generated with the sub-polygon method by HERTZMANN and ZORIN. The silhouettes are combined with a hatching technique. From [HZ00]. 118
6.8 Pre-processing using the projection of the normals of the two faces adjacent to an edge onto the Gaussian sphere (left) and its surrounding cube (right). From [BE99]. 119
6.9 Example scene rendered with the Gaussian sphere pre-processing algorithm. Note that no visibility test was used. Courtesy of Gershon ELBER and Fabien BENICHOU. 119
6.10 Pre-processing for faster silhouette detection through arrangement of face clusters in anchored cones (visualized in 2D) by SANDER et al. From [SGG$^+$00]. 121
6.11 Example for APPEL's notion of quantitative invisibility (QI) for a line passing behind the object. The dots denote the positions where the QI value changes. Adapted from [App67]. 125
6.12 Image generated using the hybrid visibility test by NORTHRUP and MARKOSIAN. Note the use of line styles for rendering silhouette strokes. From [NM00]. 126

6.13 Problem with faces almost parallel to the viewing direction (indicated by the arrow). Although the z-value of the silhouettes (indicated by dots in the left image) is 5 the value of the related pixels' z-buffer is 2. The left image shows a cut through the model as shown in the middle (with silhouette lines emphasized) and right part shows the resulting z-buffer image. 127

6.14 Testing the visibility by looking at the z-buffer values of the pixel and its 8-neighborhood. Also, not every pixel is tested (in this case every fifth pixel is examined). 128

6.15 Example for handling triangles with two silhouette edges before and after the update. The arrow indicates the viewing direction. 130

6.16 Example for treating silhouette edge clusters. In the screen-shots the upper part is visible and the lower part is not. The triangles with all edges marked alternate from not visible to visible. To remove the cluster, a path around it is detected, the inner edges are removed, and the longest segment of the path between the two leaving edges is removed. The left and right images show, respectively, the situation before and after the update. 130

6.17 Example for removing zig-zags. The left side represents the result before and right side after the update. These zig-zags might originate from cases similar to the one displayed on the right side of Figure 6.15 due to a viewing direction which was changed just slightly in the counter-clockwise direction and minimal numerical errors during the visibility test. These can occur very easily because the viewing direction is almost parallel to the triangles. 131

6.18 Example for splitting silhouette strokes at sharp angles. The left side shows the situation before the update where one stroke is twisted awkwardly. On the right side, after the update, the stroke has been split into two separate strokes yielding a better rendering. 132

6.19 Example for removing short segments, left side before and right side after the update. 132

6.20 Examples for artifact removal. Remaining gaps are due to the applied line style. 133

6.21 Examples for applying different line styles to the generated silhouette lines produced by the OpenNPAR system. 134

7.1 Example for feature driven mesh decimation using the fold index. As can be observed, the features important in the shape are preserved even at high decimation rates. 141

7.2 Technical example for feature driven mesh decimation using the fold index. See also the visualization of the fold index DoI function for this model in Figure 4.20(k). 141

7.3 Comparing two similar shapes using simplified internal skeletons. . . . 143

7.4 Comparison matrix for a number of example shapes. The gray value indicates the quality of the match, black indicates a perfect match and white denotes no match at all. 144
7.5 Searching for a model of Nefertiti's bust by sketching the silhouette using the PRINCETON 3D MODEL SEARCH ENGINE [RG03]. 146
7.6 Searching for a model of Nefertiti's bust by sketching the shape using the PRINCETON 3D MODEL SEARCH ENGINE [RG03]. 147
7.7 Example application created with the OPENNPAR library that demonstrates interactive line rendering. 148
7.8 Comparison of using sharp angles between adjacent faces ((a) and (c)) and prototypically external skeletons ((b) and (d)) for computing feature lines of mostly smooth objects. The insignificant feature lines of the external skeleton have not been pruned yet in this example. 149
7.9 Comparison of rendering silhouette drawings with skeleton drawings for different distances of an object. Created in collaboration with Nick HALPER. 150
7.10 Examples for applying distortion to vector graphic line drawings in order to create caricatures. Created in collaboration with Petra NEUMANN and Sheelagh CARPENDALE. 151
7.11 The silhouette is used to embed data with (on the right) or without (on the left) recognizable changes of the graphic. Other sources for 2D vector graphics are possible as well. Created in collaboration with Henry SONNET. 152
7.12 When the shading of an object is exchanged with a line-based NPR technique, the photorealistically rendered remainder of the model is seen in the background (a) and (b), which is very distracting for many applications. Shading can be used to enhance the appearance of the line-based NPR style. This avoids that hidden surfaces are shining through the non-photorealistically rendered parts of the models. Created in collaboration with Roland JESSE and Bernd NETTELBECK. 153
7.13 Snapshots of an animation pointing out specific parts of an anatomical example. The bones *Os cuneiformie I*, *Os metatarsale I*, *Phalanx proximalis I*, and *Phalanx distalis I* are emphasized in order to visualize their functionality. Created in collaboration with Roland JESSE and Bernd NETTELBECK. 153
7.14 Snapshots of an engine model. The tubes of the engine's cooling system are gradually emphasized by being rendered in a line shaped silhouette style. The style changes reflect the possible utilization of the single tubes and their interplay with the cooling aggregate located ad the bottom right side. Created in collaboration with Roland JESSE and Bernd NETTELBECK. 153
7.15 Hatching lines are another class of non-photorealistic rendering primitives which allow the generation of illustrations (combined with silhouettes in these examples). Created in collaboration with Johannes ZANDER and Stefan SCHLECHTWEG. 154

7.16 Rendition of the same model using single hatching (left) and triple hatching (right). Created in collaboration with Johannes ZANDER and Stefan SCHLECHTWEG. 154

B.1 Examples for textures assigned to strokes to simulate certain drawing tools such as pen, charcoal, oil paint, pencil, watercolor, and others as well as sine wave perturbations. 191

B.2 Examples for different line styles applied to the same model, with or without additional shading, with or without visibility culling of the silhouettes, and on white as well as black backgrounds. 192

List of Tables

3.1 Summary of the main groups of feature extraction algorithms. 48
3.2 Summary of feature and skeleton extraction algorithms for boundary representations. 48

4.1 Deriving a singularity property based on the principal curvature values and directions estimated in the vertices of a polygonal mesh. 63
4.2 Comparison of the properties of geometric DoI functions. 71

5.1 Summary of analysis of internal skeletons. 83
5.2 Numbers of vertices and edges of the external skeletons shown in Figure 5.19 compared with the original model shown in Figure 5.17(a). . . 99

6.1 Comparison of silhouette detection algorithms. 122
6.2 Computation times for the silhouette stroke generation (including stroke segment generation) depending on the skipping value for the bunny model with 3608 polygons and an approximate rendered size of 295 x 223 pixels. 135
6.3 Computation times in milliseconds tested on a 450 MHz PentiumIII machine with a GeForce2MX graphics board and 256 MB RAM (1P, left columns) and a 800 MHz Dual-PentiumIII machine with a GeForce1 graphics board and 384 MB RAM (2P, right columns), both running MS Windows 2000 (PN = polygon number, SED = silhouette edge detection, PrVAR = pre-visibility artifact removal, SSG = silhouette stroke generation (including stroke segment generation), PoVAR = post-visibility artifact removal, STY = stylization). The skipping value was set to six pixels and the approximate rendered size was 295 x 223 pixels. The frame-rates given are total tested rates and less than to be expected from the computation times because they also include other activity like the standard OpenGL rendering of the model. 135
6.4 Computation times in percent according to the data in Table 6.3 (abbreviations as in Table 6.3). Differences from a sum of 100% are due to rounding errors. 136
6.5 Computation times for the silhouette stroke generation (including stroke segment generation) depending on the (approximate maximal) size of the rendered image for the bunny model with 3608 polygons and a skipping value of six. 136

6.6 Comparison of visibility culling algorithms (l_{pe} is the projected edge length and n_e denotes the number of edges). 137

A.1 List of implementations created in connection with this dissertation. . . 187

Appendix A

Implementations

The following table contains a list of tools that were implemented in connection with this dissertation. In all implementations the programming language C++ was used.

Tool	Author	Description
OpenNPAR general framework	Nick Halper, Tobias Isenberg, Felix Ritter	General framework of the OpenNPAR system, based upon Open Inventor (Section B).
OpenNPAR silhouette rendering	Tobias Isenberg	Brute-force and preprocessing silhouette detection (Section 6.3.3) and visibility culling (Section 6.4).
OpenNPAR shape processing	Tobias Isenberg	E. g., Winged Edge, subdivision, and mesh decimation (e. g., Section 7.1.2).
OpenNPAR skeleton extraction	Tobias Isenberg	Internal (Section 3.3.2) and external (Chapter 5) skeleton extraction.
OpenNPAR skeleton comparison	Angela Brennecke, (Laborpraktikum)	Heuristic for the comparison of shapes using their internal skeleton graphs.
OpenNPAR stroke pipeline and rendering	Tobias Isenberg, Nick Halper, Angela Brennecke	Stroke manipulation and rendering tools for visualizing the output of many algorithms (Sections B.2 and 7.3.2).
OpenNPAR hybrid rendering	Bernd Nettelbeck (Diplomarbeit), Tobias Isenberg, Roland Jesse	Combination of photorealistic and non-photorealistic renditions in interactive visualizations (Section 7.3.2).
OpenNPAR line output and illustration watermarks	Henry Sonnet, Tobias Isenberg	Vector graphic output of strokes and embedding of additional information in data (Section 7.3.2).
OpenNPAR line distortion	Petra Neumann, Sheelagh Carpendale, Tobias Isenberg	Distortion for interactively creating caricatures of strokes (Section 7.3.2).
OpenNPAR high quality lines	Johannes Zander (Praktikum)	Generation of hatching lines based on a curvature flow field (Section 7.3.2).

Table A.1: List of implementations created in connection with this dissertation.

Appendix B

A Unifying Implementation: OpenNPAR

In order to integrate the two aspects of essence of shape algorithms—external skeleton extraction and visible silhouette stroke computation—into one system they have both been implemented within the OpenNPAR framework.[1] OpenNPAR has originally been designed to facilitate the easy creation of non-photorealistic animations and renditions [HIR+03, Hal03]. It is based on Open Inventor and adds modules for non-photorealistic rendering. These modules can be integrated into the Open Inventor scene graph which enables an easy combination of different modules to implement various non-photorealistic algorithms. These are provided as a library and can be used in various applications (see, e. g., Figure 7.7 on page 148).

The framework provides essential tools necessary not only for extracting silhouettes but also for finding external and internal skeletons. Most important among these tools are a data structure for geometric meshes that provides local information in constant time as well as a stroke handling for assembling, exporting, and displaying strokes. This, in turn, allows for stroke manipulation that can be used in additional application scenarios of silhouettes and skeletons. These aspects and some application scenarios will be discussed in this section.

B.1 Winged Edge Data Structure

The Winged Edge data structure for polygonal geometric models was originally introduced by Baumgart in 1975 [Bau75]. It is based on a list of all edges in the shape and local connectivity information for every edge. The details of the data structure have been outlined earlier in Section 5.3.2.[2]

However, the important property of the Winged Edge which makes it ideal for both external skeletons and silhouette extraction is that local information can be found in O(1) or O(k). For example, it is possible to find the next edge clockwise around a face by just locating this information in the edge's locally stored adjacency information (see Figure 5.16). As a second example, it is possible to identify all k edges attached to a vertex by just looking into the vertex' edge ring (see Figure 5.15).

---

[1] For additional images, videos, publications, and demo programs see the OpenNPAR Website at `http://www.opennpar.org/`.

[2] For a more detailed discussion on the efficient implementation of a Winged Edge data structure see [Gla91].

This fast access to local information is essential for external skeletons and silhouette extraction.[3] In the first case—the extraction of external skeletons—it is necessary to simulate the HUYGENS' principle in order to propagate wavefronts on the surface of the mesh. This requires, for example, finding edges attached to a vertex. Also the progressive meshes algorithm that is used in topology preservation for external skeletons or even for the extraction of internal skeletons requires the fast identification of edges, faces, and vertices attached to one edge. The algorithm for extraction of visible silhouette strokes needs to identify neighboring silhouette edges in order to concatenate them to silhouette strokes. This is essential for stroke based rendering because isolated silhouette edges of dense meshes are too short for a pleasing visualization, e. g., using textures and other stylization.

In general, any mesh data structure that can provide the required local connectivity information could be used for efficient skeleton extraction and silhouette detection. However, mesh data structures, such as the very common indexed vertex list, that are efficient for traditional photorealistic rendering should not be used since they cannot provide the connectivity information in constant time. Instead, mesh data structures such as the *OpenMesh* can be employed [BSBK02] that also provide fast access to local connectivity information.

The Winged Edge data structure can represent arbitrary meshes, i. e., it is not limited to triangle meshes. However, most polygonal meshes in contemporary computer graphics are based on triangles. Thus, data structures other than the Winged Edge may be more efficient, especially in terms of memory consumption [ZSD$^+$00, page 107].

B.2 Stylization and Visualizing Strokes

OPENNPAR provides means for displaying the strokes generated by either skeleton extraction or silhouette detection. This is accomplished by a stroke pipeline that stylizes, texturizes, and finally renders the strokes, mostly for non-photorealistic visualization. Modules are provided that can, for example,

- remove artifacts such as zig-zags, short stroke segments, and sharp angles (as part of the hybrid visibility culling discussed in Section 6.4.4),
- subdivide the stroke to allow for more detailed stylization,
- transform the stroke into a Bézier curve for achieving a smoother appearance,
- add regular or randomized perturbations (such as sine waves),
- change the thickness of strokes (using a constant a size or depth cueing),

---

[3] For other operations on geometric meshes such as (adaptive) subdivision and edge collapse based algorithms local connectivity information is essential, too.

- add textures for simulating certain drawing tools (e. g., pen, chalk, charcoal, pencil, watercolor, oil paint, etc.; see examples for a number of line styles in Figure B.1 and example renditions shown in Figure B.2; also see examples previously shown in Figure 6.21 on page 134), and
- output the strokes as vector graphics (for example, as PDF files).

These modules can be individually activated, parameterized, and arranged in the stroke pipeline in order to achieve the desired output.

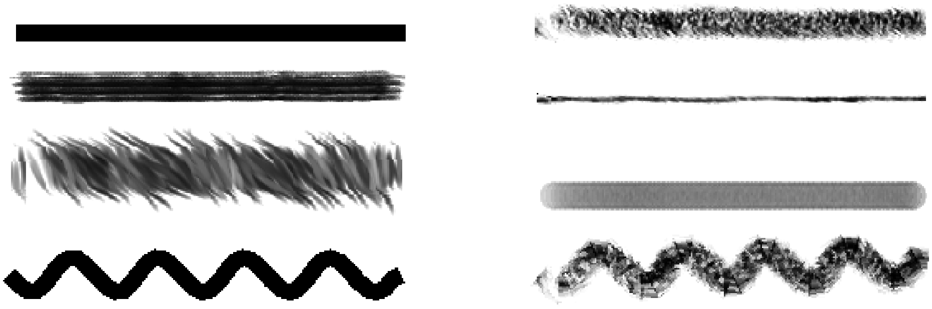
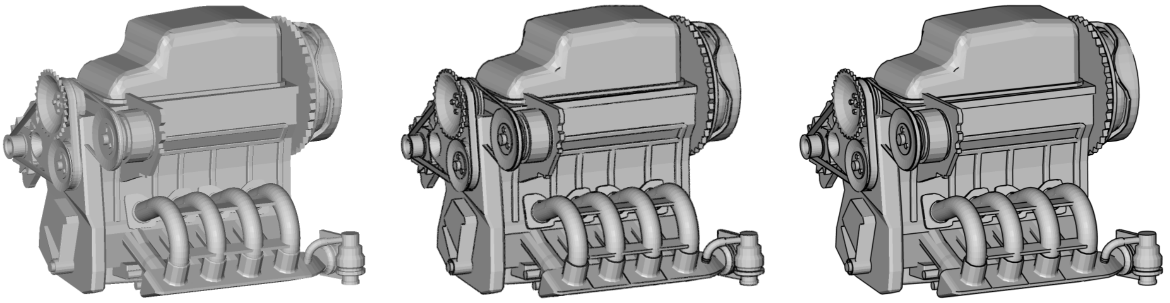


Figure B.1: Examples for textures assigned to strokes to simulate certain drawing tools such as pen, charcoal, oil paint, pencil, watercolor, and others as well as sine wave perturbations.

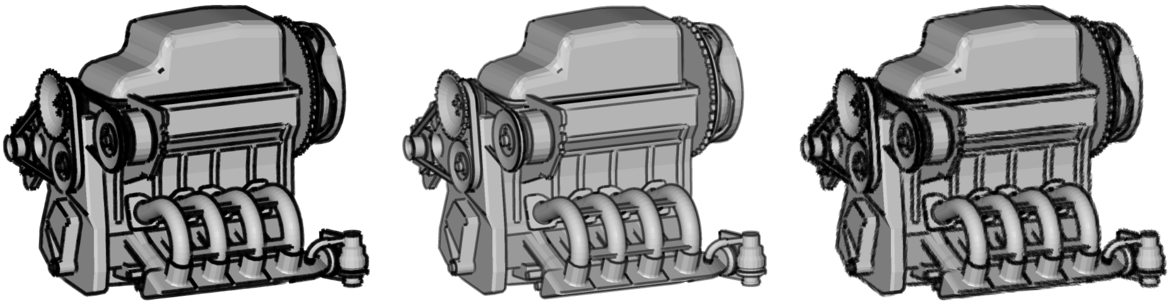
Figure B.2 illustrates the different appearances that can be achieved by changing the parameterization of the non-photorealistic rendition. The parameterization used in the example includes

- the line style (in particular the texture),
- whether or not a shaded model is displayed,
- whether or not visibility culling was applied to the lines,
- the color of the lines, and
- the color of the background.

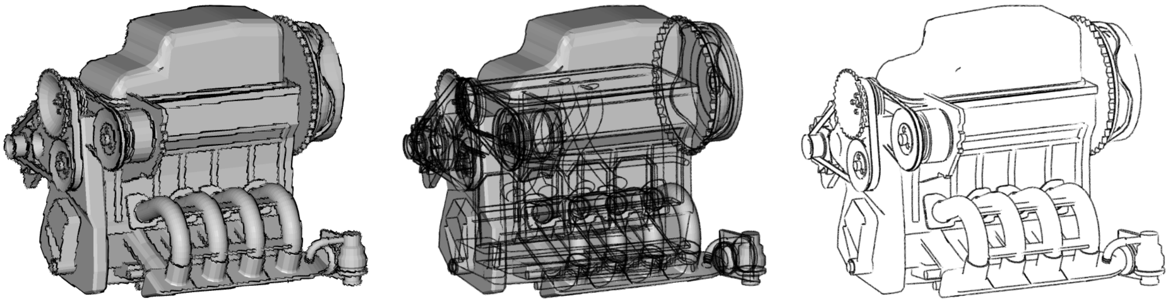
The use of different textures can simulate various drawing tools known from traditional art (e. g., Figures B.2(b), B.2(c), B.2(d), and B.2(e)). It can also create styles that are not possible with traditional drawing tools (for example, Figure B.2(f)). In addition, the texture can also simulate a path variation of a drawing tool, e. g., to simulate a sketchy style as shown in Figure B.2(g). By using silhouettes without visibility culling the appearance of a technical drawing can be achieved such as in Figures B.2(h) and B.2(j). Figures B.2(i) and B.2(j) demonstrate that by omitting the traditional shading of the object the non-photorealistic character of the rendition can be increased. Finally, using a white line style on a black background can create the appearance of a drawing on a black board as depicted in Figures B.2(k) and B.2(l). Interestingly enough, the same texture is used for creating both the charcoal drawing (e. g., in Figure B.2(b)) and the chalk rendition (e. g., in Figure B.2(l)).



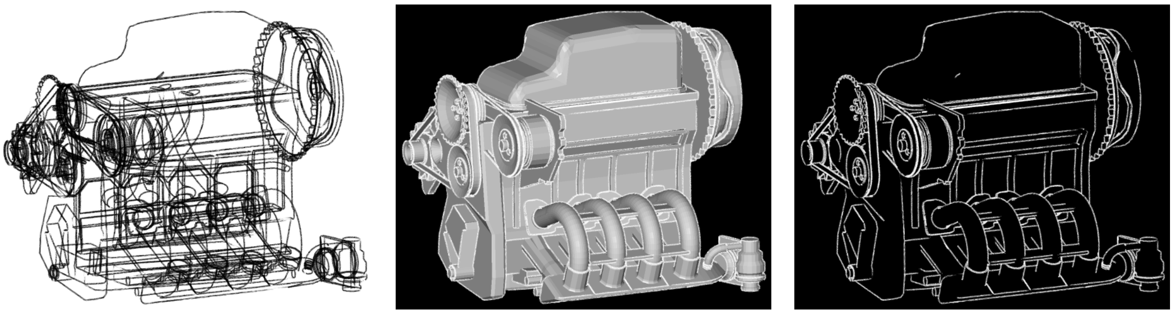
(a) Model without silhouettes. (b) Lines in charcoal style. (c) Lines in ink style.



(d) Lines in oilpaint style. (e) Lines in watercolor style. (f) Lines in van Gogh style.



(g) Lines in zigzag pen style. (h) Silhouettes without visibility culling. (i) Silhouettes without shaded model.



(j) Silhouettes without shaded model and without visibility culling. (k) Shaded model with lines in chalk style on black background. (l) Lines in chalk style on black background without shaded model.

Figure B.2: Examples for different line styles applied to the same model, with or without additional shading, with or without visibility culling of the silhouettes, and on white as well as black backgrounds.

Résumé

Personal Data

Name	Tobias Isenberg
Address	Papenburg-Privatweg 8 39106 Magdeburg Germany
Date of birth	December 14, 1974
Place of birth	Gardelegen, Germany
Nationality	German

Education

since 10/1999	Ph. D. studies at the Department of Simulation and Graphics of the Otto-von-Guericke University of Magdeburg
09/1999	Diplom degree (similar to a M. Sc.) in Computer Science from the Otto-von-Guericke University of Magdeburg, graduation with distinction, specialization: Simulation and Graphics, minor in English
05/1998	Bachelor of Science in Computer Information Systems from the University of Wisconsin-Stevens Point, USA
09/1997–05/1998	Studies and internship at the University of Wisconsin-Stevens Point, USA
10/1996–09/1999	Graduate studies at the Faculty of Computer Science of the Otto-von-Guericke University of Magdeburg
10/1994–07/1996	Undergraduate studies at the Faculty of Computer Science of the Otto-von-Guericke University of Magdeburg
08/1993–10/1994	Civil service at the Kreiskrankenhaus Gardelegen (hospital)
09/1989–06/1993	Abitur (high school diploma) at the “Gymnasium Am Weinberg” in Kleinmachnow (high school with special emphasis on mathematics, natural sciences, and technology)
09/1981–07/1989	Polytechnische Oberschule “Johann Wolfgang von Goethe” in Gardelegen

Wissenschaftlicher Lebenslauf

Persönliche Daten

Name	Tobias Isenberg
Adresse	Papenburg-Privatweg 8 39106 Magdeburg
Geburtsdatum	14. Dezember 1974
Geburtsort	Gardelegen
Nationalität	deutsch

Ausbildung

09/1981–07/1989	Polytechnische Oberschule „Johann Wolfgang von Goethe“ in Gardelegen
09/1989–06/1993	Abitur am Gymnasium „Am Weinberg“ in Kleinmachnow
08/1993–10/1994	Zivildienst am Kreiskrankenhaus Gardelegen
10/1994–09/1999	Informatikstudium an der Otto-von-Guericke-Universität Magdeburg
07/1996	Vordiplom in Informatik
09/1997–05/1998	Auslandsstudium und Praktikum an der University of Wisconsin-Stevens Point, USA
05/1998	Bachelor of Science in Computer Information Systems von der University of Wisconsin-Stevens Point, USA
09/1999	Diplom in Informatik, Abschluß mit Auszeichnung, Vertiefungsrichtung Simulation und Graphik, Nebenfach Anglistik
seit 10/1999	Doktorand am Institut für Simulation und Graphik an der Fakultät für Informatik der Otto-von-Guericke-Universität Magdeburg