Efficient Data Structures for Distributed and Mobile Geometry Processing

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der Rheinisch-Westfälischen Technischen Hochschule Aachen zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von
Master-of-Science Jianhua Wu
aus Zhejiang, China

Berichter:  Prof. Dr. Leif Kobbelt
Prof. Dr. Shimin Hu
Prof. Dr. Petri Mähönen

Tag der mündlichen Prüfung: 07. März 2006

Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek online verfügbar.
## Contents

**Preface**
- Preface: v

**Zusammenfassung**
- Zusammenfassung: vii

**Acknowledgements**
- Acknowledgements: ix

### 1 Introduction
1.1 3D Geometry ................................................. 1
1.2 Distributed and Mobile Processing .......................... 4
1.3 Contributions .................................................. 7
1.4 Outline ......................................................... 9

### 2 Geometric Data Structures
2.1 Parametric Representations .................................. 12
2.2 Explicit Representations ..................................... 16
2.3 Implicit Representations ..................................... 18
2.4 Special Notes On Conversion and Compression ............ 19
2.5 Summary ....................................................... 21

### Part I: Parametric

### 3 Multiple-Choice Decimation
3.1 Standard Mesh Decimation Algorithms ..................... 26
3.2 Multiple-Choice Techniques ................................ 27
3.3 Multiple-Choice Decimation ................................ 29
  3.3.1 Motivation ............................................... 29
# Contents

3.3.2 Mapping MCA to Mesh Decimation ........................................ 30  
3.3.3 Detailed Decimation Algorithm .......................................... 31  
3.4 Comparisons and Results ..................................................... 33  
3.5 Progressive Structures ......................................................... 36  
3.6 Summary ............................................................................. 38  

4 Geometric Stream Decimation .................................................... 41  
4.1 Related work ........................................................................ 42  
4.2 Ideal Stream Algorithm for Decimation .................................... 44  
4.3 Real Stream Algorithm for Decimation .................................... 46  
4.3.1 Overall Procedure ............................................................ 48  
4.3.2 Reading Input Streams ...................................................... 50  
4.3.3 Multiple-Choice Decimation ............................................. 51  
4.3.4 Writing Output Streams .................................................... 51  
4.4 Results and Discussions ........................................................ 52  
4.4.1 Discussions ...................................................................... 55  
4.5 Summary ............................................................................. 56  

5 3D Digital Watermarking ............................................................ 59  
5.1 Related Work ........................................................................ 61  
5.2 Overview of the Watermarking Algorithm ............................... 62  
5.3 New Orthogonal Basis Functions ............................................ 64  
5.4 Watermark Embedding ............................................................ 68  
5.5 Watermark Extraction .............................................................. 69  
5.6 Watermarking Results ............................................................. 70  
5.6.1 Overall Performance ......................................................... 71  
5.6.2 Robustness Against Attacks ............................................. 72  
5.6.3 Comparing to Laplace Basis Functions ............................... 74  
5.7 Summary ............................................................................. 75  

Part II: Explicit ................................................................. 79  

6 Optimized Splat Sub-Sampling ................................................... 81  
6.1 Background and Motivation .................................................... 81  
6.2 Previous Work on Explicit Geometry ..................................... 83
6.3 Sub-Sampling Overview ........................................... 85
6.4 Initial Splat Generation ........................................ 87
  6.4.1 Elliptical Splats ............................................. 88
  6.4.2 Hole-free Approximation ................................. 89
6.5 Greedy Selection ................................................ 90
6.6 Global Relaxation ............................................... 91
  6.6.1 Relaxation .................................................. 93
  6.6.2 Removing Redundant Splats ............................... 94
6.7 Results and Comparisons ...................................... 95
  6.7.1 Elliptical vs. Circular Splats .............................. 98
  6.7.2 Comparing to Point Cloud Simplification ............. 98
  6.7.3 Comparing to Mesh Simplification ..................... 99
6.8 Summary ......................................................... 100

7 Progressive Splatting ............................................. 103
  7.1 Related Work .................................................. 104
  7.2 Progressive Splat Decimation Algorithm .................. 106
  7.3 Initial Splat Creation ....................................... 106
  7.4 Error Metrics and Splat Merge Operators .................. 107
    7.4.1 $L^2$ Metric ............................................. 107
    7.4.2 $L^{2,1}$ Metric ........................................ 109
  7.5 Progressive Splat Structures ............................... 109
  7.6 Compressed Progressive Splat Transmission ............. 110
    7.6.1 Structure Transformation ............................... 111
    7.6.2 The Prediction Rule ................................... 111
    7.6.3 Quantization ............................................ 112
    7.6.4 Entropy coding ......................................... 113
    7.6.5 Experimental Results .................................. 114
  7.7 Results and Comparisons .................................... 115
    7.7.1 Comparison to Splat Decimation Methods ............ 116
    7.7.2 Comparison to Mesh Simplification .................... 118
  7.8 Summary ....................................................... 119

Part III: Implicit .................................................. 123
Contents

8 Piecewise Linear Distance Fields ........................................... 125
  8.1 Background and Motivation ........................................... 126
  8.2 Previous Implicit Data Structures .................................... 128
  8.3 Piecewise Linear Signed Distance Fields ........................... 129
  8.4 Generation Steps .................................................... 130
    8.4.1 Least Squares Linear Approximation .......................... 131
    8.4.2 BSP Splitting Plane Selection ................................ 132
    8.4.3 Final Linear Approximation Correction ...................... 136
    8.4.4 Discussions .................................................... 137
  8.5 Results and Applications ............................................ 138
  8.6 Summary ............................................................ 140

9 Robust Structure Recovery .................................................. 143
  9.1 Related Work ....................................................... 144
  9.2 Variational Surface Approximation ................................ 146
  9.3 Hybrid Variational Surface Approximation ......................... 147
    9.3.1 Sphere Fitting .................................................. 150
    9.3.2 Cylinder Fitting ............................................... 151
    9.3.3 Rolling-Ball Blend Patch Fitting .............................. 152
  9.4 Results ............................................................. 154
  9.5 Summary ............................................................. 156

10 Conclusion ........................................................................ 161

Bibliography ......................................................................... 169

Curriculum Vitae ................................................................. 189
Preface

This thesis addresses the data structure and geometry representation problems which arise in the emerging interdisciplinary research field of distributed and mobile geometry processing.

On the one hand, three-dimensional geometry is now establishing as a new digital multimedia data type after text and sound in 1980’s and images and video in 1990’s. The major advantage of 3D geometric data over previous multimedia data is that it enables users to actually interact with the displayed contents, which paves the way to more and enhanced interactive multimedia applications.

On the other hand, the rapid evolution of the network technology brings new potentials for communication between people and computers in heterogeneous environments. Nowadays computing hardware becomes more powerful and also turns out to be ubiquitous thanks to the ever increasing availability of distributed and mobile digital devices.

When we combine the strength from both above fields, we come to the research topic of this thesis, the distributed and mobile geometry processing. Like all other computer science subjects, efficient algorithms always have to be supported by efficient data structures, which reveals the importance of the our main target, i.e., to develop efficient data structures for our dedicated scenarios.

Within such distributed and mobile network-computing environment, the major challenges emerge from the quality requirements for the underlying geometric data structures, which have to meet the following four criteria:

- **Compactness**, to reduce the data size to save network bandwidth and to have a fast processing speed;

- **Progressiveness**, to allow users to access the important part of geometric data in the very beginning of the transmission;
• Robustness, to compete with network package loss;

• Security, to provide ownership assertion and copyright protection of digital geometry data when distributing them.

To fulfill these requirements, we present a wide range of efficient algorithms to process data structures adapted to distributed and mobile geometry processing. This wide range is necessary to cover all possible setups according to the natural classification of geometry data structures and the underlying mathematical surface representations: each type of geometric data structures has its own specific benefits and drawbacks and currently they seem to be irreplaceable with each other. Hence we have to develop various algorithms to provide effectiveness for all types of geometric data structures.

More specifically, for parametric representations, we have presented an in-core multiple-choice decimation algorithm as well as an out-of-core stream decimation method to produce compact and progressive representations for moderate-sized to massive triangle meshes. In addition, an efficient digital watermarking scheme which can provide secure representations for triangle meshes is also introduced and it can be easily extended to explicit geometric data like point sets.

For explicit representations, we have developed an optimized splat sub-sampling method for point sets to produce high-quality, compact, and robust splat-based representations. To provide the progressiveness property, a splat decimation algorithm is introduced, which can also reduce the data size of explicit splat models.

For implicit representations, we describe a compact and progressive linear approximation to the implicit signed distance fields of the input data. A robust structure recovery algorithm is also presented to extract highly compact shape information from the parametric surface geometry.

Based on the availability of these efficient geometric data structures, we expect to see more and more forthcoming practical applications for distributed and mobile geometry processing. While large-scale distributed scenarios mainly come from industrial applications like product design, computer simulation or movie production, small-scale mobile applications will gain relevance due to the vast number of mobile devices like game consoles, mobile phones, or PDAs for personal entertainment and mobile communications.
Zusammenfassung

Nach Text und Sound in den 80-er Jahren und nach Bild und Video in den 90-er Jahren hat sich die 3D-Geometrie als der neue digitale Multimedia-Datentyp etabliert. Einer der Hauptvorteile von 3D-Daten ist die Möglichkeit der Interaktion des Benutzers mit dem dargestellten Inhalt, was zu einer großen Anzahl neuer und verbesserter interaktiver Anwendungen geführt hat.


Wenn wir die Stärken aus beiden Bereichen kombinieren, so kommen wir zu dem neuentstandenen Forschungsbereich der verteilten und mobilen Geometrieverarbeitung. Diese Arbeit behandelt die grundlegenden Probleme der Datenstrukturen und der Geometrierepräsentationen in diesem Bereich. Genauer gesagt wird hier eine große Zahl an qualitativ hochwertigen Algorithmen für die Generierung effizienter geometrischer Datenstrukturen präsentiert, die speziell an die Anforderungen der verteilten und mobilen Geometrieverarbeitung angepasst werden.

Die resultierenden geometrischen Strukturen und Repräsentationen haben vier wichtige Eigenschaften, nämlich Kompaktheit, Progressivität, Robustheit und Sicherheit, und eignen sich gerade für verschiedene Anwendungen im Bereich der 3D-Multimedia Kommunikation wie z.B. CAD, CSCW, Film/Animationsproduktionen, 3D Mobilfunk, Online-Spiele, 3D online Dokumentation, etc.
Acknowledgements

First of all I would like to express my sincere thanks to my doctoral supervisor, Prof. Leif Kobbelt, for his continuous support during the whole period of my doctor’s study. Leif is the one who will always be there to give me advice whenever it is necessary. With his numberless motivating and inspiring comments, he has taught me not only how to do good research, but even how to write good papers and make good presentations.

I also thank the other two co-advisors of this thesis, to Prof. Petri Mähönen for our cooperation work on mobile communication systems to make the interdisciplinary idea possible and for his kind help on the testing environments, and to Prof. Shimin Hu who is also my master’s advisor and shows me the way to the computer graphics research field, for his enthusiasm to review this thesis.

Let me say “Thank you” to all my (previous and present) colleagues in the computer graphics group of RWTH Aachen University, for their support on my research work with insightful discussions, solving challenging problems, and even proofreading papers. A special thanks goes to Silke van Betteraey, who is our secretary and has helped me so many times in my (foreign) daily life in Germany.

This thesis work has received the financial support from DFG (German Science Foundation) Graduate College GRK 643 “Software for Mobile Communication Systems”. I like this kind of form for doctoral study and I think the Graduate College and its speaker, Prof. Otto Spaniol, have really done a great job to make it possible that professors and students from different area can have efficient cooperations for the same purpose to produce promising results.

Last, but not least, I thank my parents, Houmei Wu, and Meifang Wu, for giving me life to this world, for educating me with many aspects, and for their unconditional support and encouragement to pursue my interests, even when the interests go beyond boundaries of language and geography.
1 Introduction

1.1 3D Geometry

Three-dimensional (3D) geometry is the description and representation of real-world or virtual models and sceneries within a computer system. It naturally plays a central role in the computer graphics (CG) research field [FvDFH95] where digital images have to be synthesized from 3D geometry data and displayed on a two-dimensional (2D) screen.

In recent years, 3D geometry data has been quickly establishing as a new digital multimedia data type after text and sound in the 1980’ and image and video in the 1990’ to represent various kinds of real-world information (cf. Fig. 1.1). While this is obvious for typical geometry applications like computer-aided design [CAD05] for industrial product engineering (e.g., electronics, automobiles, and aeroplanes) where geometric objects are the core elements, different types of geometric models also have triggered breakthroughs in several other fields like education, medicine, media, arts, entertainment and so on (cf. Fig. 1.2).

![Figure 1.1: Evolution of digital multimedia data types.](image_url)
For computer simulation [RM98], e.g., output data typically needs to be visualized to allow the user a more intuitive and reliable inspection of the results. The respective algorithms are mostly geometry based. In medical imaging [Ban00], X-ray images and CT scans can either be considered to be 2D/3D images or they can be understood as special geometric shape representations of the respective organs which enables the derivation of physical properties like their volume or weights. 3D geometry has also been widely used in public entertainment such as animated feature movies (e.g. “Shrek”, “Finding Nemo” and “The Incredibles”) made with special effects and computer animation technology [Ker03]. In addition, diverse enhanced multimedia and network applications including virtual museums, Avatars, 3D-TV, online-encyclopedias, E-commerce, E-learning, and architectural planning, etc., have benefited a lot from the integration of 3D models into digital documents. Having more descriptive power than previous multimedia data types, 3D geometry data enables common users for the first time to truly interact with the displayed contents, beyond only running pre-recorded audio and video footage or following hypertext links.

To provide efficient solutions to the various requirements in the above application scenarios, 3D Geometry Processing [KT01, SS01, SGP] has been an active and productive
research area recently, where concepts from applied mathematics, computer science, and engineering are adopted to design efficient algorithms for the acquisition, manipulation, animation, and transmission of 3D geometric models. Like for all other computer science subjects, efficient algorithms always have to be supported by efficient data structures, which reveals the importance of the main research target of this thesis, i.e., to develop efficient data structures for geometry processing applications.

Typical data structures for geometry processing usually can be classified into three different categories [Kob03], parametric representations, explicit ones and implicit ones, corresponding to the respective mathematical descriptions of 3D geometric models. A more thorough discussion about various geometric data structures will be presented in the separate Chapter 2, where we will also explain that different structures have their specific advantages and drawbacks with respect to different geometry processing operations.

Due to this variety and diversity, in practice, the selection of geometric data structures has to be application-specific. For this reason, we will present in this thesis a wide range of algorithms to generate efficient geometric data structures of all three different categories. This will not only provide users more flexibility to select different structures, but moreover it will be even a requisite for an ubiquitous computing environment such as our designated distributed and mobile processing situation (cf. Sec. 1.2), where almost all types of 3D geometry data will be commonly employed.

In addition to the generation of geometric data structures, in order to facilitate the flexible design of application-specific data structures for the same geometric model, conversion methods between different geometric data structures [Kau87, YT02, LC87, KBSS01] exist which in most cases cause a certain information loss. In fact, some of our proposed algorithms also belong to this category with the advantage of providing more conversion accuracy. On the other side, to reduce the data size, standard compression techniques known for images, sound and video [Com, Say00], have also been introduced for the new multimedia type of geometric data in either a lossy or a lossless manner [AG05]. Due to the space limitation of this introduction chapter, we are going to present more special notes on these two issues in Session 2.4 of the next chapter.
1.2 Distributed and Mobile Processing

Extrapolating the current developments in Information Technology (IT) into the future, the greatest potential for extending more influences of computer systems in the human society seems to emerge from the rapid evolution of the network technology [Tan02]. Today, brought forth by the ubiquitous networks, new means of communications between computers and people have linked all into heterogeneous environments. In this thesis, we are specifically interested in the distributed and mobile processing applications made possible by the steadily improving networking capabilities.

First of all, networks have been utilized to increase the performance of computer systems when it comes to the idea of distributed processing [BEPT00] for large-scale, high performance computing. The typical “number cruncher” scientific computation applications (e.g. the simulation of physical, chemical, or biological processes [Fuj90, Fuj00]) already run on computers with several processors or even on several computers that are linked by fast LAN (local area network) network connections (e.g. PC clusters). On the
WAN (wide area network) scale, the fast-growing Internet and the new potentials for communication between people and computers have led to the development of highly effective computer-supported cooperative work systems (CSCW) [BS00], e.g., for industrial applications. Within such systems, all information is processed in a distributed way through network connections, which also implies that large amounts of data have to be transferred within the network frequently.

On the other hand, the ubiquitous networks, especially new ways of data exchange through wireless communications, also prevail the small-scale, low performance mobile processing [Sch03, TMC05]. Based on an ever increasing level of user acceptance, versatile mobile devices, e.g. laptop computers, personal digital assistants (PDAs), mobile phones or portable entertaining devices, are now permeating almost all aspects of our daily lives. The correspondingly improved standards and extended new services have brought forth many innovative mobile applications in domains like electronic commerce, electronic education, electronic government, new forms of working and so on. Indeed, witnessing the vast number (up to the order of billions) of available mobile devices, it seems timely and necessary that one conducts additional research work to further promote the development of mobile processing applications.

Given 3D geometry as a new multimedia data type (cf. Sec. 1.1), we naturally come to the interdisciplinary research field, the distributed and mobile geometry processing (cf. Fig. 1.3), where we combine the competence in both, networks and geometry data to provide basic tools for new services and application domains. The possible applications can be various, as we have just mentioned, like distributed CAD and CSCW, online clinical diagnosis, online virtual museums, and a large number of e-applications like e-commerce, e-learning, e-office, where geometry data can be embedded easily.

In the context of distributed and mobile geometry processing, the standard sequential geometry processing algorithms, e.g. rendering, decimation, fairing and optimization, have to be adapted to the new system architecture. Attention should be paid to the implementation issues for distributed and mobile processing which will include task definition, computational model, task management strategy, data management strategy and system communication and other aspects [CT96].

More specific situations need to be targeted for effective multimedia communications, since geometry data have to be transferred frequently through networks of different bandwidth and quality of service (QoS). While for distributed applications, the network
1 Introduction

connections can be fast and reliable, mobile devices usually only have low-bandwidth channels and possibly even with package loss, e.g., for wireless networks. Geometry data will also be accessed and processed by computer systems with greatly different computing power, ranging from high-performance PC clusters to extremely low-end single-chip mobile phones. Moreover, almost all types of geometric data structures are likely to be adopted which will comprise mesh surfaces, point/splat sets, and volumes. This variety is necessary since each representation provides special functionalities that are appropriate for different applications or different stages of a processing pipeline.

Within such distributed and mobile network-computing environment, especially for the designated multimedia communication systems, the core issue is always the efficiency of the underlying geometric data structures and corresponding geometry representations. With respect to the research target of this thesis, we summarize here that efficient data structures for distributed and mobile geometry processing have to meet the following four mandatory requirements:

- **Compactness.** As geometric models become ever larger \([LPC^+00]\), it is substantial to reduce the data size to save the network bandwidth and more importantly, to scale the model complexity, i.e., the amount of geometric details in a 3D model, to the available hardware resources in a heterogeneous network environment for an interactive processing speed.

- **Progressiveness.** An efficient transfer scheme has to be progressive, e.g., like the well-known progressive JPEG image format \([JPE]\) on the world wide web (WWW). Geometric models should also be transmitted in a progressive fashion that users can view and process the most important information of geometry data in the very beginning of the transmission phase.

- **Robustness.** When confronted with unstable network connections, considering the transmission efficiency, robust geometry representations and data structures turn out to be an obligation to provide resistance w.r.t. package loss. This is especially crucial for the low band mobile applications as they usually work with lossy wireless communications.

- **Security.** Anytime you distribute your data over the network which cannot be guaranteed to be secure, you will face the challenge of intellectual property right protection \([Bis04]\) threatened by possible attacks from hackers and crackers. Thus
for distributed and mobile geometry processing, we need efficient techniques to ensure the security, e.g., to provide ownership assertion and copyright protection of digital geometry data.

1.3 Contributions

Regarding the major target of this thesis, we have developed a large set of high-quality algorithms to produce efficient data structures for distributed and mobile geometry processing. Taking the variety of different geometric data structures into account, this thesis will cover all three types, i.e. parametric, explicit and implicit, of geometry representations. Our main contributions (cf. Tab. 1.1) are to make the respective geometric data structures satisfy the above requirements (cf. Sec. 1.2). They are listed below:

- For **parametric** representations, we have developed
  - An in-core multiple-choice decimation algorithm to produce compact and progressive representations for moderate-sized meshes. With the newly introduced multiple-choice optimization framework, our algorithm runs faster than most previous methods while consuming much less memory space.
  - An out-of-core stream decimation algorithm to produce compact and progressive representations for massive meshes. The stream processing fashion matches perfectly with the distributed and mobile processing pipelines, which also enables our scheme to deal with both input and output of arbitrary size with high speed and high quality.
  - An efficient digital watermarking scheme that can provide secure representations for meshes and be easily extended to explicit data like point sets. The watermarking is performed on the spectral domain spanned by a set of new orthogonal basis functions. Our computation speed is so fast that, for the first time to our knowledge, we can watermark geometric models with more than $10^6$ vertices.

- For **explicit** representations, we present
  - An optimized splat sub-sampling method for high-resolution point sets to produce high-quality and compact representations. We have generalized the
standard circular surface splats into elliptical ones and are trying to provide sub-sampled splat representations with best possible quality by global optimization.

- A splat decimation algorithm applied to point sets to produce compact splat representations for progressive splatting. Typical applications for triangle meshes like progressive rendering and progressive transmission now can be easily extended to the explicit splat setting.

- For implicit representations, we offer

  - A compact and progressive piecewise linear approximation to the signed distance field of the input mesh. The proposed representation is flexible enough that it can function in the basic level as the uniform surface representation and can support many geometry processing applications like global tolerance control, CSG, surface extraction, and so on.

  - A robust structure recovery algorithm to extract highly compact implicit information from the parametric input surface geometry. With a predefined set of geometric elements, our method can recover the underlying surface structures more accurately even with a much lower number of elements than previous work. A variational optimization framework leads to an automatic optimization procedure and makes our algorithm produce high quality reverse engineering results fully automatic.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Param.</th>
<th>Exp.</th>
<th>Imp.</th>
<th>Compact</th>
<th>Prog.</th>
<th>Robust</th>
<th>Secure</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC Decimation</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Stream Decimation</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>3D Watermarking</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Splat Sub-Sampling</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Progressive Splats</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Lin. Dist. Fields</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Structure Recovery</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 1.1: A wide range of geometry processing algorithms presented to generate efficient geometric data structures for distributed and mobile applications.
1.4 Outline

This thesis is structured as following (see also Tab. 1.1):

**Chapter 2** introduces three categories of geometric data structures with their advantages and drawbacks which explain the reason for the wide coverage of this thesis.

**Part I** focuses on parametric representations and contains the following three chapters.

**Chapter 3** introduces a fast mesh decimation algorithm based on multiple-choice optimization to scale the data complexity. A progressive mesh data structure that can be derived from decimation results is also discussed.

**Chapter 4** extends the stream processing idea to an out-of-core decimation algorithm for massive meshes that can not be fitted into the main memory.

**Chapter 5** gives a 3D digital spectral watermarking algorithm for large meshes to provide the copyright protection and ownership assertion of online geometry data.

**Part II** focuses on explicit representations and contains the following two chapters.

**Chapter 6** introduces an optimized sub-sampling method of point sets for high-quality surface splatting. The robustness property of explicit representations is also briefly discussed.

**Chapter 7** presents a splat decimation algorithm for progressive splatting. The resulting progressive splat data structures and their compression scheme are introduced too.

**Part III** focuses on implicit representations and contains the following two chapters.

**Chapter 8** introduces a compact linear approximation to the signed distance field of the input geometry, as well as its various evoking geometry applications.

**Chapter 9** presents a robust structure recovery algorithm via hybrid variational surface approximation, which can provide highly compact implicit information of the surface geometry and reveal its inherent characteristics.

**Chapter 10** finally concludes this thesis and presents some promising future directions as well as new forthcoming applications in both, industrial and end-user setups.
2 Geometric Data Structures

As for all computer science subjects, efficient algorithms always have to be supported by efficient data structures. The main content of this thesis is a wide range of high-quality algorithms to produce efficient data structures for distributed and mobile geometry processing applications. Before we go through all details of every algorithm, we first have to know how real-world 3D geometric objects are represented and stored in computers. This kind of data storage fashion can be referred to as surface representations in the specific field of geometry processing, or the more general term, geometric data structures.

In general, we are mainly concentrated on surface representations, although we know that, except surfaces, there are also points, curves, and solids being able to represent the geometric shape of a 3D object [FvDFH95]. While a surface representing the object boundary is a 2D element embedded in 3D space, points and curves can be considered as 0D and 1D elements respectively. Solids are real 3D volumes occupied by the 3D objects. The reasons why we focus on surfaces are two-fold. First of all, as the initial target of computer graphics research, drawing a geometric object visually on the display screen (the procedure called rendering) can be most easily done with surface representations. Second, up to now, most geometric objects and relevant geometry processing applications (say, more than 90% roughly) are based on surfaces too.

According to [Kob03], typical data structures for geometry processing usually can be classified into three different categories, the parametric representation, the explicit one and the implicit one, corresponding to the respective surface representations of 3D geometric models. Note that this classification is mainly distinguished by the underlying mathematical differences of diverse geometric data structures. More details will be given in the following sections. Typical model representations of each category will also be introduced as well as their efficient implementations for practical applications.

Besides, the above diversity of geometric data structures are also necessary for the current level of technologies which are still evolving. All representations have their
specific advantages and drawbacks since each provides special functionalities that are appropriate for different applications or different stages of a processing pipeline. Recently, [Kob03, KB03, BBK04, BPK05] are trying to use them simultaneously as hybrid representations for the same geometry where advantages of each type can be combined to support more efficient geometry processing algorithms. Generally speaking, these different categories of geometric data structures are so far irreplaceable with each other, which also somehow explains why we have worked on different geometric data structures in different parts with various efficient algorithms in this thesis.

More precisely, for geometry processing algorithms, we consider the situation where different geometric data structures have to be specifically chosen to provide efficient data access for the the three most frequently used geometric operators as defined in [Kob03]. The three operators are as following respectively,

- **Evaluation.** Access to the surface attributes of geometric data like surface positions and normal directions. A typical usage is for rendering of 3D geometric models on 2D display screens.

- **Query.** Determine the status, i.e. inside or outside, of any point in 3D space with respect to a solid object bounded by the geometric surface. Also query the distance from a point to the surface as well as its closest point on the surface. The query operator is widely used for Constructive Solid Geometry (CSG) operations.

- **Modification.** 3D geometric models can be modified with either geometry by changing surface positions or topology by cutting or joining operators for surface design and modeling.

We will find in the subsequent sections that the parametric representations will have domain structures and are easy for evaluation, the explicit representations do not have topological consistency constraints and are easy to be modified, and the implicit representations with domain structures are good for query.

### 2.1 Parametric Representations

Given a geometric surface $S$, its parametric surface representation is usually mathematically defined by a function $f : \Omega \subset \mathbb{R}^2 \rightarrow S \subset \mathbb{R}^3$ that maps a two-dimensional
2.1 Parametric Representations

Figure 2.1: Parametric geometry data: a triangle mesh (middle) with smooth-shaded rendering (left) and its decimated mesh (right).

parameter domain Ω to the three-dimensional surface $S$. It can be considered as the range of the function $f$.

With this domain structure, the evaluation operators to compute 3D point positions on the geometric surface can be straightforward for parametric data representations, by simply evaluating the function values of $f$ on the 2D parametric domain $Ω$. Many other geometric operations on the complex 3D surface $S$, like neighborhood search and texture mapping, can also be deduced onto the simpler 2D domain $Ω$.

On the contrary, the modification operators cannot be easily applied to the parametric surfaces, as when changing the geometrical structures of a surface $S$, topological inconsistency will hardly be avoided thus resulting in surfaces of bad quality. After modification, extra efforts are usually necessary to update the distorted domain structure $Ω → S$. Moreover, the query operators for inside/outside or distance computations are usually expensive for parametric surface representations as the surface $S$ only captures the range of a function, not its kernel.

Typical parametric surface representations include the traditional free-form spline surfaces (e.g. the NURBS surfaces [PT97]) and the general triangle meshes [KB+00]. Although the free-form surfaces can have higher approximation and smoothness order and be used in many CAD applications, they are not as flexible as triangle meshes, e.g., surface trimming and smoothness control between patch boundaries for free-form
surfaces are still challenging problems nowadays. So in this thesis, we are mainly focused on triangle meshes as parametric data structures and will present various algorithms to improve their efficiency in Part I of the thesis.

Triangle meshes, built by tightly connected triangles (cf. Fig. 2.1), are becoming the de facto standard surface representation for geometric applications and also have been supported by the modern graphics hardware for efficient processing [ATI, NVD]. An usual triangle mesh \( \mathcal{M} \) with \( m \) vertices and \( n \) triangles is composed with two parts: its geometry and its topology. The geometry is represented by a set of vertices \( \mathbf{P} \) sampling on the surface \( \mathcal{S} \),

\[
\mathbf{P} = \{ \mathbf{p}_1, \ldots, \mathbf{p}_m \mid \mathbf{p}_i \in \mathbb{R}^3 \},
\]

and the topology is a set of triangles \( \mathbf{T} \),

\[
\mathbf{T} = \{ \mathbf{t}_1, \ldots, \mathbf{t}_n \},
\]

where each triangle \( \mathbf{t}_i \) is a vertex index triple \((i, j, k)\) referring to its three vertices in \( \mathbf{P} \).

The standard implementation to this kind of triangle mesh is to use arrays to keep both sets of the vertices and triangles. Extra arrays might also be necessary to speedup the most frequent neighborhood information access for a certain vertex or triangle, e.g., an array of vertex index lists where each entry corresponds to the neighboring vertices of a vertex, or a triangle index array indicating the neighbors of a triangle.

More efficient structures for triangle meshes are based on halfedge arrays [CKS98, BSBK02] which allows for both efficient access and modification of the neighborhood information. Each edge of a mesh will be splitted into two halfedges pointing to the two endpoints of that edge in opposite directions. Main connectivity information can be stored on each halfedge with pointers to one vertex, one face, the next halfedge, and the opposite one. The one-ring neighborhood query of a vertex can be efficiently performed by traversing the outgoing halfedges of the vertex. This generic and efficient mesh data structure has been implemented as an open-source mesh library called OpenMesh and available for download [OM].

Recall the definition of parametric surface representations, a triangle mesh \( \mathcal{M} \) representing a geometric surface \( \mathcal{S} \) will have natural local domains \( \Omega \) corresponding to each single triangle and the function mapping can be the barycentric parameterization based on its three vertices. Finding the larger global domain \( \Omega \) that can map all triangle into a single plane (or sphere) is a more tough problem called mesh parameterization which
2.1 Parametric Representations

has been an active research area these days. As the global parameterization requires the
mesh to be topologically equivalent to a disk, and as this is usually not the case, most
current work has to cut the mesh into separated patches aiming at stretch-minimizing
parameterization [DMA02, GGH02, GGS03, PH03, THCM04].

From an abstract point of view, triangle meshes are $C^0$ piecewise linear surfaces with
connectivity graphs embedded in the 3D space. The approximation power of triangle
meshes is quadratic, e.g. when a smooth surface is approximated with piecewise linear
elements of uniform edge length $h$, the approximation error is of the order $O(h^2)$. When
the edge length is halved by increasing the sampling density, the approximation error
decreases by a factor of $1/4$ [Dav75].

Although this quadratic approximation power of triangle meshes is lower than the
other type of parametric surface representations, the free-form spline surfaces, triangle
meshes are much more flexible since the conceptually simple triangles can be freely posi-
tioned and connected to represent geometric models of arbitrary geometry and topology.
In addition, as parametric surface representations, triangles meshes also inherit both
advantages and drawbacks of them, i.e. they can be easy for evaluation operators and
difficult for query and modification.

Also due to the above flexibility, when we deal with triangle meshes, we always have
to take care of the mesh topological consistency, i.e., to check if a mesh is manifold or
not. A mesh is a manifold if neighborhood of each vertex can be locally homeomorphic
to a disk (or half-disk for boundaries). Since non-manifold meshes (cf. Fig. 2.2) are
hard to process in most geometry applications like decimation, compression, smoothing,
parameterization, and modeling, each geometric operator has to be validated in order
not to produce non-manifold meshes. To restore the topological consistency, there also
exists efficient mesh repair algorithm to fix non-manifold meshes [GW01, Ju04, BK05].

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{non-manifold meshes}
\caption{Non-manifold meshes with a gray triangle connected to a non-manifold
dge (left) and a non-manifold vertex (right).}
\end{figure}
2 Geometric Data Structures

Figure 2.3: Explicit geometry data: point-based rendering (left) of a dense point set (middle) and its coarse approximative splat set (right).

2.2 Explicit Representations

The explicit representation of a geometric surface $\mathcal{S}$ is given by a function $f : \mathbb{N} \rightarrow \mathcal{S} \subset \mathbb{R}^3$ that explicitly describes the geometric positions of all surface primitives with respect to their indices.

The most common surface primitive for explicit representations is the infinitesimal point, i.e. the geometric surface $\mathcal{S}$ is sampled and described by a point set $P$ [LW85, GD98] (cf. Fig. 2.3, middle),

$$ P = \{p_1, ..., p_m \mid p_i \in \mathbb{R}^3\}, $$

which is exactly the geometry part of a triangle mesh. The initial motivation to use explicit point sets rather than parametric triangle meshes comes from the rendering of complex geometric models. I.e., as the complexity of triangle meshes grows much faster than the evolution of the resolution of display screens, the projected area of triangles (especially distant ones to the viewers) will often be smaller than a single pixel, which makes the standard complex scan-line conversion operations unsuitable. Rather, by simply replacing triangles with points can drastically improve the rendering performance at this situation.
Comparing to triangle meshes, the point sets are $C^{-1}$ piecewise constant approximations of the original geometric surfaces. The $C^{-1}$ continuity means that point sets do not maintain any topological consistency, which makes them even more flexible than triangle meshes. On the other hand, point sets only have a linear approximation power, comparing to the quadratic one of triangle meshes. When the uniform distances $h$ between point samples (cf. edge length for triangle meshes) is halved, the approximation error of order $O(h)$ will also be decreased to one half [Dav75]. This insufficient object-space approximation power of point sets will require much denser sampling and more surface primitives than the triangle meshes, which can bring more burden to geometry processing systems. Moreover, when the point set surface is zoom-in viewed, visual holes will also appear during the rendering since points are infinitesimal while the sampling density can not go up to infinite.

To deal with the above two problems, the purely point sets are usually generalized to the splat sets [ZPBG01] (cf. Fig. 2.3, right), where the surface primitive is switched from infinitesimal points to overlapped splats with spatial extent to provide as high approximation power as triangle meshes and to visually bridge the gap between points as well. The splat sets $T$ to represent a geometric surface $S$ is a set of splats $t_i$,

$$T = \{t_1, ..., t_m\}.$$ 

Each splat $t_i$ is an oriented object-space ellipse,

$$t_i = (c_i, u_i, v_i),$$

where $c_i$ is its center, $u_i$ and $v_i$ are two non-normalized vectors defining its major and minor axis, and then its normal direction $n_i$ is fixed as a normalized vector of $u_i \times v_i$. The splat sets are $C^{-1}$ piecewise linear approximations of the geometric surfaces, which have the same quadratic approximation order as triangle meshes, while the flexibility of point sets are largely preserved for splat sets as both are explicit surface representations. From this point of view, splat sets are a good compromise between simplicity/flexibility and geometric approximation power [KB04].

Explicit surface representations are simple and we can directly use lists or arrays as basic data structures for both point sets and splat sets in the implementation. Hierarchical explicit data can also be realized with spatial partitioning structures like Octrees [BWK02]. Concerning the three geometric operators, because explicit representations are very flexible and do not impose any restrictions on the topological structures,
they are much easier for modification than parametric surfaces. This flexibility also brings another important characteristic, robustness, for the distributed and mobile geometry processing because even with package loss during the network transmission, the surface information still can be approximated with the received data. On the contrary, as points or splats are not tightly connected with each other, more extra efforts are needed for evaluation and query operators.

### 2.3 Implicit Representations

An implicit representation is *implicitly* defined to be the zero-set of a scalar function $F : \mathbb{R}^3 \to \mathbb{R}$, with the geometric surface $S = \{(x, y, z) \in \mathbb{R}^3 \mid F(x, y, z) = 0\}$. These representations are also referred to as volumetric ones in volume graphics [CKY00]. Comparing to parametric representations, they are considered to be the *kernel* of the function $F$.

The implicit function $F$ usually can be derived from the mathematical implicit functions [BW97], e.g., the typical one

$$F(x, y, z) = (x - a)^2 + (y - b)^2 + (z - c)^2 - r^2 = 0$$

for a spherical surface with center $(a, b, c)$ and radius $r$. As we can see, the domain structures of the implicit geometric representations, $\mathbb{R}^3 \to \mathbb{R}$, makes them very easy
to perform the query operators by simply computing the value of $F(p)$ at a certain 3D position $p$. The inside or outside status is determined from the sign of $F(p)$, e.g. positive for outside and negative for inside. In practice, the value $F(p)$ evaluation will also reflect a relative measure for the distance from $p$ to the implicit surface geometry. On the other side, as the implicit domain structures are mapped to only one dimension, it is really difficult to evaluate arbitrary 3D point positions on the surface $S$.

Although there are many different choices of function $F$ to capture the implicit information of a given surface $S$, the most common implicit representation is the signed distance field, which assigns to every point in 3D space its signed Euclidean distance to the surface $S$ (cf. Fig. 2.4). As the distance sampling density can not be infinite, there are different data structures to store the signed distance information. The most basic one till now is a uniform 3D grid $[s_{ijk}]$ that stores a scalar value at each grid node $(i, j, k)$. This discrete data is interpolated into the voxel cells by tri-linear functions [LC87].

Since for signed distance fields the accuracy in the vicinity of the zero-set surface is usually more important than further away from the surface, the uniform grid is not memory efficient because the spatial scalar field is sampled with the same rate everywhere and this makes it difficult to deal with high-resolution/precision objects. This redundancy can be avoided by adapting the sampling density to the distance from the surface with adaptively refined octrees [Sam90] where only cells having intersections with surface are subdivided to the highest resolution.

Furthermore, we can restrict the local refinement to those cells for which the tri-linear interpolation deviates more than the prescribed tolerance from the actual distance field. This restriction leads to even better adaptivity since extreme refinement is only necessary in the vicinity of the highly curved surface regions [FPRJ00]. Similarly, linear (not tri-linear) interpolation deviation can also be used to adjust the sampling density with even more compact binary space partitioning (BSP) trees [WK03a].

### 2.4 Special Notes On Conversion and Compression

Given different types of geometric data structures and surface representations, conversion methods between them are necessary in order to facilitate the free choice of application-specific data structures for the same geometric model. For example, 3D
scan-conversion techniques [Kau87, YT02] have been used to convert parametric structures to implicit ones by computing the signed distance fields of the input geometry. Inversely, marching cubes techniques and their extensions [LC87, KBSS01] are utilized to extract parametric surface representations from implicit data.

There are many other conversion methods and most of them can be looked as resampling procedures with certain amount of information loss. In fact, some of our proposed algorithms in this thesis also belong to this class of conversion methods with the advantage that we can provide higher conversion accuracy or better tradeoffs between information loss and data size/processing speed. Examples can be found in the coming chapters, like the optimal splat sub-sampling scheme to convert parametric triangle meshes to high quality explicit splat sets (cf. Chap. 6), or the piecewise linear distance fields to convert parametric triangle meshes to compact implicit structures (cf. Chap. 8).

Another important issue we have to specially notice is about data compression. As we have explained in the introduction (cf. Sect. 1.2), the compactness property is very critical for geometric data structures in distributed and mobile environments. The standard way to reduce the data size are the well developed data compression techniques, like quantization, encoding and decoding [Com, Say00], which have been widely used for images, sound and video in the multimedia community.

In general, data compression techniques are more or less like dealing with the binary bit stream to encode the data. While in this thesis, for the new multimedia data type of 3D geometry, we will not go into the details of the corresponding data compression methods. Rather, we are focusing on more profound approaches to reduce the data size of geometric data by mathematically reordering the original data. As we will show later, our proposed techniques like decimation (cf. Chap. 3 and 4), sub-sampling (cf. Chap. 6), and piecewise approximation (cf. Chap. 8) all belong to this category. An obvious advantage to do this is that even further data reduction can be easily accomplished by post-applying geometry compression techniques [AG05] to our mathematically pre-reordered data.
2.5 Summary

In this chapter, we have introduced the three different categories of data structures for geometry processing and presented their specific advantages and weaknesses in supporting various geometric operators. We have found that until now, it is still impossible to completely replace one type of geometric representation with the other one. Different categories of geometric data structures will continue to co-exist for the future due to their basic mathematical differences and may have to work at same time for practical applications.

This diversity will be even more evident for the research topics of this thesis, the distributed and mobile geometry processing, where almost all types of geometric data structures have to be delivered by service providers to enable geometry data to be adapted to greatly varied computing hardware.

To address this problem, in this thesis, we have developed a wide range of high-quality algorithms to produce efficient data structures for distributed and mobile geometry processing, which can cover all three categories in the classification. We are going to introduce them respectively in the following three parts.
Part I

Parametric Representations
3 Multiple-Choice Decimation

The compactness property of geometric data structures is always of most importance for distributed and mobile applications. Within such kind of remote applications, geometric data have to be transferred over the network very frequently, while on the contrary, the network connection speed currently available for distributed and mobile systems is usually far slower than local harddisk access. In order to have a similar performance of distributed and mobile applications as those running locally, the complexity of geometric data structures is necessary to be carefully tuned. Compact representations can save the network bandwidth and thus lead to less network latency.

On the other side, due to the technical advances of 3D laser scanners, iso-surface extraction methods or other physical measurements, the complexity of geometric datasets is increasing very fast. Mesh models containing millions of triangles are common nowadays and a few extraordinary massive models may have billions of triangles and require some Gigabytes’ storage space [LPC+00]. These large models are already difficult to process on large-scale and high-performance distributed systems and they seem to be impossible to deal with on current individual mobile devices with very limited computing capability. In this sense, compact geometry data structures are essential to adapt to diverse hardware resources to ensure a fast processing speed.

In the research field of 3D geometry processing, mesh decimation techniques have proven to be an efficient way to reduce the data complexity of geometric models while trying to keep their overall shapes as much as possible. While most high-quality mesh decimation methods today have adopted a greedy optimization framework, we have introduced a much faster and more memory efficient mesh decimation algorithm in [WK02] based on multiple-choice techniques. The result quality of our approach is almost same as previous ones, but as it consumes less memory and runs faster, our method is more suitable for distributed and mobile systems with restricted computing power.
In addition, we also explain in this chapter a progressive mesh data format [Hop96] that can be derived from the mesh decimation sequence, and can freely adjust the data complexity. This progressive property also can benefit applications like progressive transmission and rendering in the distributed and mobile environment. Comparing to the typical sequential data access, where users have to wait until all data have been received, the progressive structures make it possible that the most important part of the data can be delivered and viewed first in the beginning (cf. Fig. 3.7).

3.1 Standard Mesh Decimation Algorithms

Mesh decimation techniques have been an active research area over at least the last decade [Gar99, GGK02, LRC+03]. After many different approaches have been investigated and compared, today two basic concepts can be considered as the standard solutions: One concept is vertex clustering and the other is incremental decimation.

While vertex clustering [RB93, Lin00] is very fast and effective, the quality of the resulting meshes is often not satisfying. The major drawbacks of this approach are that it usually leads to a vertex distribution which does not adapt to the local curvature of the surface and that it cannot guarantee a proper manifold topology of the resulting mesh. On the other hand, incremental decimation [Hop96, GH97, WTHS01], where one atomic decimation step is executed after the other, typically leads to superior mesh quality in terms of approximation error for a prescribed triangle count as well as triangle count for a prescribed approximation error. In addition, incremental decimation can guarantee the preservation of the initial topology.

Although there are many different incremental decimation schemes in the literature, they all follow the same basic principle that each possible atomic decimation operation (candidate) is rated according to some quality criterion. Then in every step the “best” candidate is decimated which triggers new evaluations of the quality criterion in its vicinity. As observed in [KCS98], the decimation problem can be understood as an instance of the knapsack-problem with the number of decimation operations being the objective function and the geometric approximation error being the capacity function.

Obviously, finding the optimal decimation sequence is a very complex problem [AS94] and consequently one has to find solutions with approximate optimality. The above best-
Multiple-Choice Techniques

Multiple-Choice algorithms (MCA) are a probabilistic optimization technique that has been investigated thoroughly in the fields of tele-communication, distributed systems, and theoretical computer science. The fundamental idea behind MCA is quite simple and intuitive and can be explained best using the well-established bins-and-balls model [ABKU99, Vöc01] (cf. Fig. 3.1).

![Figure 3.1: Bins-and-balls model: random ball insertion (top), greedy ball insertion (middle) and multiple-choice ball insertion (bottom).](image)

first strategy is, in fact, a greedy strategy to find a decimation sequence that is close to optimal.
Consider we have $n$ balls to be uniformly distributed over $n$ bins, i.e., each ball is put into a random bin and for each ball, the destination bin is selected independently from earlier choices (cf. Fig. 3.1 top). Obviously, the expected number of balls in each bin would be one. Moreover, a more detailed probability analysis of this random allocation procedure shows that the maximum load, i.e., the expected number of balls in the fullest bin, will be

$$(1 + o(1)) \frac{\ln(n)}{\ln(\ln(n))},$$

with high probability [KSC78].

Instead of putting each ball into an independently selected bin, the idea of MCA is to choose a small random subset of $d$ bins and then put the ball into that bin with the least number of balls already in it [Vöc01] (cf. Fig. 3.1 bottom). By this MCA strategy, we can guarantee that the maximum load is

$$\frac{\ln(\ln(n))}{\ln(d)} + O(1),$$

(see [ABKU99] for a detailed proof) which is an exponential improvement compared to the pure random approach. Of course, an extreme case will appear when the random number $d$ equals to the bin number $n$ and this is actually the well-know greedy choice solutions (cf. Fig. 3.1 middle).

After their discovery, MCA have been used as an optimization tool in many different applications. For example, Karp et al. [KLM92] used them for efficient hashing in the context of shared memory computer simulation. In this application, the balls are just the hash-keys and the bins stand for table entries. The maximum load is the maximum length of the collision chains in the hash table.

Another application example is data allocation and data management [MSS96]. Here the balls represent data objects and the bins are memory blocks or disk sectors. The maximum number of requests to the same storage location corresponds to the maximum load.

In parallel and distributed systems, MCA have been adopted to online load balancing [ABKU99]: the balls represent jobs, the bins are online computers, and the maximum load is the maximum number of jobs per machine. Other applications include routing in networks [CMM+98], queueing processes [VDK96], etc. A thorough survey of Multiple-Choice optimization techniques can be found in [MRS02].
So far we are not aware of any application of MCA to optimization problems in the
field of computer graphics. In this chapter we are using MCA to find approximately
optimal solutions in the context of mesh decimation.

3.3 Multiple-Choice Decimation

Rather than the standard greedy solution, we are using a different optimization strategy
based on the above multiple-choice techniques to address the mesh decimation problem.
Instead of doing greedy optimization (which requires to find the best choice among all
candidates) we are using a Multiple-Choice paradigm (which requires to find the best
choice only among a small subset of the candidates).

3.3.1 Motivation

The motivation for using this probabilistic optimization strategy is the fact that when
decimating high resolution meshes, most of the vertices will be removed anyway – usually
90% to 99% of the original data. Hence it is not necessary to look at all possible
candidates in every decimation step.

Let us e.g. assume that we decimate a given 3D model down to 5% of the original
complexity. The basic idea of Multiple-Choice techniques is to pick a small random
subset of candidates, say 8 possible standard (half)edge collapse operators (cf. Fig. 3.2),
and then perform the best of them. This strategy leads to a wrong decision only in the
rare case when all 8 collapses do not belong to the 95% majority of candidates that are
supposed to be removed. The probability for a wrong decision is hence

\[
\left(\frac{5}{100}\right)^8 \approx 10^{-11},
\]

which implies that a reliable decision if a certain atomic decimation step is part of
the optimal decimation sequence or not can be based on a small subset of candidates.
Notice that for the above estimate we exploit the fact the most atomic operations are
independent from each other and the exact order of the decimation steps matters only
for direct neighbors.

The major benefit of the Multiple-Choice optimization compared to the greedy op-
timization is that the algorithmic structure is much simpler. An implementation of
the greedy strategy usually requires a priority queue data structure for the candidates that has to be initialized and updated during the decimation (whenever the priorities/qualities of the candidates are re-evaluated). For the Multiple-Choice optimization we do not need a priority queue and consequently we save memory space and computation time. Our experiments show that the Multiple-Choice decimation is more than a factor of 2.5 times faster than a highly optimized greedy implementation (both using the QEM quality criterion [GH97]). While producing the same quality of the output meshes as the QEM-based greedy optimization, our multiple choice approach can run at decimation rates of more than 70K triangles per second on a standard PC.

3.3.2 Mapping MCA to Mesh Decimation

In order to apply MCA to the mesh decimation problem we have to map balls, bins, and maximum load to the corresponding mesh entities. Since the balls are enumerated in the outer loop (“for each ball make a MC decision”) they correspond to the decimation steps. The bins represent the possible choices in each step and hence they correspond to the possible candidates. The maximum load finally is the value that is to be optimized and consequently we associate it with the quality criterion that is used to rate the candidates (cf. Tab. 3.1).

<table>
<thead>
<tr>
<th>Bins-and-balls Model</th>
<th>Multiple-Choice Decimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>balls</td>
<td>atomic decimation steps</td>
</tr>
<tr>
<td>bins</td>
<td>candidate operators</td>
</tr>
<tr>
<td>maximum load</td>
<td>maximum approximation error</td>
</tr>
</tbody>
</table>

Table 3.1: Multiple-Choice decimation correspondence to the bins-and-balls model.

In this setup, the MCA approach to mesh decimation consists of testing a small set of \(d\) randomly selected candidates (e.g. edge collapses) in each step and performing that decimation operation among this small set that has the best quality value.

The parameter \(d\) in the above description controls how good the MCA approximates the standard greedy approach. In fact, for \(d = n\) both algorithms are identical and the priority queue mechanism used in the greedy approach has only the purpose of speeding up the algorithm since it allows to reuse the ordering of the unmodified quality values from the previous decimation step.
3.3 Multiple-Choice Decimation

The statistical analysis of MCA (Sec. 3.2) implies that even for relatively small values of $d$ the approximation of MCA is very good. As will be demonstrated in Section 5, the mesh quality that we obtain with MCA is already indistinguishable from the greedy approach if we choose $d = 8$. With such a small number of candidates to be tested in every step, we do not need a global data structure like a priority queue to speed up the computation because the re-computation of the ordering is actually less expensive than updating the global queue. This reduces the memory consumption and makes the algorithm much easier to implement.

3.3.3 Detailed Decimation Algorithm

Using the above idea of Multiple-Choice decimation, the overall framework of our incremental decimation algorithm is as follows:

1. Read (part of) the mesh model into the main memory.
2. Randomly choose $d$ atomic decimation operators from all candidates, compute their respective decimation costs using a specific error/quality metric.
3. Select that operator with smallest cost/error and perform this operator.
4. Repeat step 2 and 3, until the user-defined terminating condition is met.

This generic framework is somehow similar with those previous ones (except the Multiple-Choice optimization strategy), but it is much easier to implement and it does not put any limitations on the choice of the atomic decimation operator or on the error or quality metric.

In our implementation we use the error quadrics (QEM) introduced in [GH97] to rate the quality of each candidate. The atomic decimation operator is the halfedge collapse that does not introduce new vertex positions but rather sub-samples the original mesh [KCS98] (see Fig. 3.2). We prefer half-edge collapses since they make progressive transmission more efficient (no intermediate vertex coordinates) and enable the construction of nested hierarchies on unstructured meshes [KCVS98] that can facilitate further applications.
Figure 3.2: The halfedge collapse operator (EC) for mesh decimation where the dashed halfedge (left) is collapsed to one of its endpoints; And its straightforward inverse, the vertex split operator (VS) for mesh refinement.

<table>
<thead>
<tr>
<th></th>
<th>Greedy Simplification</th>
<th>MC Decimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize</td>
<td>initialize quadrics, evaluate quality for all candidates, perform global queue sorting</td>
<td>initialize quadrics</td>
</tr>
<tr>
<td>Select Candidate</td>
<td>top of the queue</td>
<td>best out of $d$ random choices</td>
</tr>
<tr>
<td>Decimate</td>
<td>perform operator, locally recompute qualities, update global queue</td>
<td>perform operator</td>
</tr>
</tbody>
</table>

Table 3.2: Multiple-Choice decimation algorithm compared with greedy simplification.

Our implementation is based on the OpenMesh data structure [BSBK02] which is a generic edge-based polygon mesh data structure. We could even increase the performance of our implementation by switching to a more specialized data structure but we decided to use OpenMesh for software design reasons.

In Table 3.2, we summarize and compare the greedy and the MCA implementation of the QEM-based decimation. It is obvious to see that the MCA decimation does much less work than the greedy version and it can be expected to be much faster. Moreover, since both algorithms are using the same quality criterion, they are producing very similar decimation results.
3.4 Comparisons and Results

We compare our MCA decimation scheme to a highly optimized implementation of the greedy decimation scheme. Both implementations are using the same underlying mesh data structure, the same routines for the QEM evaluation and the same half-edge collapse procedure. Hence the differences in running time and memory allocation are only due to the different optimization principles.

All experiments were done on a commodity PC with AMD 1700+ CPU and 1 GB RAM. Since the MCA is a randomized algorithm (and since the system performance usually varies slightly due to background processes competing for CPU time) we let each experiment run five times and then took the median of the respective timings to eliminate outliers.

The parameter $d$ in the MCA is set to 8 in all experiments. With this value we obtain a very similar output quality for both approaches and the variance for the MCA is sufficiently low.

The approximation error is measured by the Hausdorff-distance between the original mesh and the decimation result. We chose the maximum error instead of some average since this value is more relevant in most technical applications (maximum tolerance).

The timings (excluding reading from and writing to disk) and errors for various models are summarized in Tab. 3.3. The MCA algorithm is about a factor of 2.5 times faster than the greedy version. It reaches a maximum performance of up to 70K decimated

<table>
<thead>
<tr>
<th>models</th>
<th># triangles</th>
<th>maximum error</th>
<th>running time(s)</th>
<th>speed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>input</td>
<td>output</td>
<td>greedy</td>
<td>MCA</td>
</tr>
<tr>
<td>Bunny</td>
<td>69666</td>
<td>696</td>
<td>0.0036</td>
<td>0.0040</td>
</tr>
<tr>
<td>Sucker</td>
<td>230141</td>
<td>10138</td>
<td>2.5</td>
<td>2.62</td>
</tr>
<tr>
<td>Max</td>
<td>398043</td>
<td>5918</td>
<td>7.60</td>
<td>7.25</td>
</tr>
<tr>
<td>Dragon</td>
<td>871414</td>
<td>10520</td>
<td>0.0012</td>
<td>0.0011</td>
</tr>
<tr>
<td>Buddha</td>
<td>1087469</td>
<td>16412</td>
<td>0.0015</td>
<td>0.0014</td>
</tr>
</tbody>
</table>

**Table 3.3:** Time and error performance comparisons of the Multiple-Choice decimation algorithm (MCA) and the greedy decimation method (greedy) both based on the quadric error metric (QEM) and half-edge collapses. Speed is measured as triangles/second.
triangles per second. We also depict the absolute maximum geometric errors for the Bunny model and the Max model (cf. Fig. 3.3) when decimating them to various levels of details by the two different algorithms. Whether the models are drastically decimated or not, the accuracy of the MCA algorithm is almost identical to that of the greedy version.

Figure 3.3: Absolute maximum geometric errors for the Bunny model (left) and the Max model (right) by measuring the respective Hausdorff distances.

Figure 3.4: The Bunny model (70K triangles) decimated to 696 triangles by the greedy decimation (left two) and the MCA version (right two). Models are shown with hidden-lines and corresponding error plots.

To obtain a better understanding of which sub-tasks in the decimation algorithms are using the most CPU time, we run a detailed profiling on the greedy and the MCA version when processing the same model (the bunny). The results are given in Table 3.4. It turns out that in the MCA version the evaluation procedure for the QEM as well as
3.4 Comparisons and Results

<table>
<thead>
<tr>
<th>Stages</th>
<th>MCA</th>
<th>greedy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>t(s)</td>
<td>%</td>
</tr>
<tr>
<td>init. quadrics</td>
<td>0.143</td>
<td>12.0</td>
</tr>
<tr>
<td>eval. quadrics</td>
<td>0.625</td>
<td>52.6</td>
</tr>
<tr>
<td>chk. collapse</td>
<td>0.246</td>
<td>20.7</td>
</tr>
<tr>
<td>update mesh</td>
<td>0.048</td>
<td>4.0</td>
</tr>
<tr>
<td>best out of $d$</td>
<td>0.127</td>
<td>10.7</td>
</tr>
<tr>
<td>update queue</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>total</td>
<td>1.189</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 3.4: Running time profiles of Multiple-Choice decimation and greedy decimation. Notice that the total time is higher than in Table 3.3 because of the frequent time measurements during runtime.

the procedure that checks if a given candidate collapse is legal (topological consistency, normal flipping) are executed less frequently than in the greedy version. This indicates that a certain portion of these evaluations and tests during the update of the priority queue are redundant, i.e., the quality rating of a candidate is updated several times (triggered by collapses in the vicinity) before it is actually considered for decimation. Of course, this redundancy cannot be avoided completely in the MCA version but it is obviously reduced.

Another observation is that the best-of-$d$ selection ($d=8$) takes much less time than the updating of the priority queue which justifies the claim that recomputing the ordering in the MCA version is less expensive than reusing the ordering of the unmodified part from the previous step as it is done in the greedy version.

Analyzing the memory usage, we find that for QEM-based decimation algorithms and a mesh model with $n$ vertices, we need $10\times4\times n$ Bytes for the quadrics (double precision) and $3\times8\times n$ Bytes for the edge-based priority queue ($3n$ edges, each has $4$ Bytes priority value and $4$ Bytes edge pointer) without considering the storage for the mesh itself. In total, the greedy implementation requires at least $64n$ Bytes memory overhead. Our MCA version instead only uses $40n$ Bytes for the quadrics which reduces the memory overhead by $37.5\%$. Even compared with the well-known memoryless simplification method [LT98], which only needs $24n$ Bytes for a priority queue, our approach is still
acceptable with a much faster speed. In addition, since the current memory overhead
of MCA is just used for candidate ordering and the MCA strategy is independent of
that, we can remove this memory overhead completely by adopting the cost functions
in [LT98] if needed.

The Figures 3.4 to 3.6 demonstrate that the visual quality of the decimated meshes
generated by the MCA version are not distinguishable from the result of the greedy
version. In Figure 3.5, the dragon model is drastically simplified (99%) but the difference
between both versions is still hardly perceivable. Since both versions are based on the
same quality criterion, they preserve the same amount of geometric detail (cf. Fig. 3.6).

3.5 Progressive Structures

The above decimation sequence can be used to produce a progress mesh data for-
mat [Hop96]. More specifically, given the original complex mesh \( \mathcal{M}_0 \), the decima-
tion procedure will continuously perform a sequence of halfedge collapse operators,
\( \{EC_1, EC_2, ..., EC_k\} \), and finally generate a coarser mesh \( \mathcal{M}_k \). As a halfedge collapse
operator \( EC \) can easily have its inverse, the vertex split operator \( VS \) (cf. Fig. 3.2), we
then compose the progressive mesh structure \( \mathcal{P}\mathcal{M}_0 \) of the original mesh \( \mathcal{M}_0 \) as following,

\[
\mathcal{P}\mathcal{M}_0 = \mathcal{M}_k + \{VS_k, VS_{k-1}, ..., VS_2, VS_1\}.
\]
3.5 Progressive Structures

Figure 3.6: The Buddha model (1088K triangles, left) and simplified ones with 32402 triangles by greedy (middle) and MCA (right) methods. The lower row shows zoom-in views of the corresponding models.

With this progressive mesh $PM_0$, any resolution (higher than the base mesh $M_k$) of level of details (LOD) of the original mesh $PM_0$ can be acquired by applying certain sequence of vertex split operators to the base mesh $M_k$ (cf. Fig. 3.7).

In general, the above progressive mesh structure behaves in the similar way as the standard progressive JPEG image transmission [JPE] over the Internet. While for progressive JPEG images, small-sized images are refined with detail coefficients, for progressive meshes, the small base mesh can be transferred first and followed by lengthy bits of vertex split refinement information.

This progressive fashion is good not only for transmission, but also for rendering and processing in the networking environment. On the clients’ side, only a small amount of data have to be received for users to see the most important part (like overall shapes)
of the whole geometric model. Then users can decide in the very beginning whether to continue the transmission or not to save both time and network resources.

In the specific settings of this thesis, the progressive structures also enable easy model interactions for low performance low bandwidth devices like PDA or mobile phones and low-cost, fast data preview for large-scale distributed applications.

### 3.6 Summary

Geometric models are getting larger and larger and even worse, these large models have to be transmitted very frequently within distributed and mobile geometry processing systems. All these request us to reduce the data size of geometry data for efficient appli-
3.6 Summary

cations. In this chapter, we have proposed a novel decimation algorithm to reduce the size of parametric triangle meshes. Unlike the most previously used greedy optimization framework, we introduce the multiple-choice random optimization idea for the first time from theoretical computer science to the field of computer graphics for mesh decimation. Comparing to related work, the resulting algorithm runs much faster, needs far less memory, but has almost same result quality.

Moreover, the output of a mesh decimation algorithm, the decimation sequence, can be used to generate the progressive mesh structures for dense meshes, which allow for efficient progressive transmission and rendering of geometric models in the network computing environments.
4 Geometric Stream Decimation

We have addressed the complexity reduction issue in the previous chapter (Chap. 3) for complex geometric data structures. This is certainly important for the distributed and mobile application scenarios especially on those low-performance computing devices like PDAs, mobile phones, or portable game consoles.

However, more and more large-scale distributed applications might need extremely high resolution data to provide enough precision which require huge amount of space for data storage and processing. Virtual museums and heritage protection projects, e.g., can produce massive models with some billions of triangles by precise 3D laser scanners [LPC+00]. For industrial applications like CAD design, medical visualization or biological simulation, the extracted models can also be very large.

It is obvious that these massive models are even more necessary to be decimated. But in fact, their data sizes are so large that they are often not fitted to the main memory size, and thus can not be processed by most standard mesh decimation algorithms which run in-core and have to load the whole model once into the memory.

To face this challenge, we have presented in [WK03b] an efficient stream algorithm for mesh decimation which runs out-of-core and does not require to permanently store the data at all (not even on disk). With this flexible stream algorithm, both the input and output size of the geometric data can be arbitrary without any limitation.

The concept of stream processing is that input data is taken in sequentially without backtracking. The output data is written sequentially as well and no feed-back to the input stream is possible. The amount of in-core memory that is allocated by a stream algorithm does not depend on the amount of data to be processed. Since stream algorithms process the data in one single pass, we can implement a software system where the geometry generating pre-process feeds its output directly into the decimation post-process without storing it to disk. In addition, as the stream processing fashion
matches the network transmission well, the stream decimation algorithm can also be easily incorporated with distributed and mobile applications.

Besides the independence from the sizes of the input and output meshes respectively, our stream algorithm has several important additional features: Since the decimation technique itself is based on incremental edge-collapsing with quadric error metrics (QEM) [GH97], we obtain a mesh quality which is indistinguishable from state-of-the-art in-core decimation techniques. On the other hand, we obtain decimation rates of 30K to 40K triangles/sec on commodity PC hardware including I/O times which is competitive to the fastest out-of-core algorithms with similar properties (which, however, produce meshes of far inferior quality).

4.1 Related work

From the previous chapter we know that, in general, mesh decimation is a very complex optimization problem. For most applications, the computation of the exact global optimum is far too complex [AS94]. Hence, one usually tries to find solutions with approximate optimality where the computation costs can be traded for geometric sub-optimality. Over the last years the greedy optimization paradigm has established the de facto standard for mesh decimation algorithms. In the greedy approach, the decimation is performed by a sequence of atomic decimation steps which typically remove a single vertex from the mesh. The greedy paradigm then states that in each step the best decision is made without any look-ahead or back-tracking. In order to efficiently identify the best choice in each step, all candidates have to be organized in a global priority queue [Hop96, GH97].

We have proposed another approximate optimization technique for mesh decimation in [WK02] and Chapter 3. It has been shown that randomized multiple choice optimization [MRS02] produces results that have almost the same quality as the results of greedy optimization but with significantly reduced computation costs. Since no global data structure such as a priority queue needs to be maintained, the algorithmic structure of multiple choice decimation is extremely simple. In each step, a random set of candidates is picked and the best among these candidates is chosen. Usually a small number of 10 to 15 random candidates is sufficient to produce meshes which are indistinguishable from the meshes produced by greedy algorithms.
4.1 Related work

To solve the problem of ever increasing input model sizes, out-of-core decimation techniques have been introduced. These techniques are designed to perform the decimation while reading the data such that only the output model has to be stored. In [ESC00] this is achieved by pre-sorting all edges according to their length and using this ordering for the decimation sequence. A more efficient algorithm appeared in [Lin00] where a vertex clustering technique [RB93] is applied. Incoming geometry data is accumulated on the fly in a voxel grid which guarantees complete independence from the input mesh complexity. Since the (random access) voxel grid has to be stored in-core, the memory requirements of this algorithm are on the order of the output mesh complexity.

To enable application scenarios where neither the input mesh nor the output mesh of a decimation algorithm fit into main memory, the out-of-core vertex clustering has been extended in [LS01] such that no internal data structure is required anymore. Instead, the decimation is performed in several passes. Although the algorithm requires several out-of-core sorting steps it still achieves a high decimation performance of some 30K triangles/sec on a commodity PC.

The major drawback of vertex clustering techniques for mesh decimation is the relatively poor geometric and topological quality of the resulting meshes. The output is usually no longer manifold and the vertex density does not adapt to the local curvature. A non-uniform clustering technique has been proposed in [SG01] but the required space partition data structure requires memory space proportional to the output mesh complexity. The RSIMP approach [BW00] in its out-of-core implementation [Wat02] is another adaptive vertex clustering technique which can be tuned to exploit caching effects by sorting the data according to the corresponding voxel-bins. However, it still loses performance compared to [LS01]. The multi-phase algorithm in [GS02] can produce high quality meshes, since after an initial uniform clustering phase, the second phase applies a standard greedy decimation procedure [GH97]. However, again, the maximum size of the output mesh is limited by the memory resources.

Another way to decimate massive meshes is to split the data into smaller blocks and then stitch the decimated pieces together. This approach can easily be combined with incremental edge collapsing [GH97] and thus usually leads to meshes with far superior quality compared to vertex clustering. However, the splitting and stitching can be computationally expensive and special care has to be taken at the seams between the blocks to avoid mesh artifacts.
In [Hop98] this approach is applied to terrain models and in [Pri00, CRMS03] to arbitrary meshes. In both cases the advantage of improved mesh quality is compromised by a significantly reduced decimation rate. In [CRMS03] a rate of only 6K-10K triangles/sec is achieved for large datasets on a PC comparable to the one used in [LS01]. This rate is even halved if the spatial splitting pre-process is counted in as well.

4.2 Ideal Stream Algorithm for Decimation

When decimating a stream of geometry data, the mesh is logically split into three parts: the unread postfix of the input stream representing the unprocessed part of the input mesh, the in-core portion of the mesh that is currently processed by the decimation algorithm, and the written prefix of the output stream representing the part of the output mesh that has already been decimated to the target resolution (cf. Fig. 4.1).

Figure 4.1: This snapshot of the stream decimation to a cylinder mesh shows the yet unprocessed part of the input data (left), the current in-core portion (middle) and the already decimated output (right). The data in the original file happened to be pre-sorted from right to left.

As we already defined in the introduction, a stream algorithm has to transform a sequential input stream of arbitrary size into an output stream of also arbitrary size by using an in-core buffer of only fixed size. In our mesh decimation setup, these streams consist of geometry data in STL format. In this format (dubbed “triangle soup” in [Lin00]) every triangle is given by its three vertices with three coordinates each. Although this format redundantly stores every mesh vertex several times (six
times in average) it is still preferred in many applications such as rapid prototyping since no global indexing is required.

Since the stream algorithm uses an in-core buffer of limited size, we have to assume that the geometry stream is *approximately* pre-sorted (e.g. by one coordinate). This is a very natural assumption since most massive mesh models are generated by some marching cubes type algorithm which builds up the geometry layer by layer [LC87, CL96]. Obviously the pre-sorting does not have to be perfect. We only have to guarantee that the portion of the input stream that lies between the actual occurrence of a particular triangle and its position in a perfectly sorted stream, fits into the in-core buffer of the stream algorithm. In the rare cases where this mild sorting requirement is not satisfied, we have to apply an out-of-core pre-sorting step like in [LS01].

Let $N_{\text{max}}$ be the maximum number of triangles that fit into the in-core triangle buffer. The stream algorithm is performing three different operations that affect the filling level $N_{\text{current}} \leq N_{\text{max}}$ of that buffer.

- **READ$(k)$** takes the next $k$ triangles from the input stream and inserts them into the current in-core portion (which maintains both geometry and topology) of the mesh. $N_{\text{current}} \leftarrow N_{\text{current}} + k$

- **DECIMATE$(k)$** performs $k$ edge collapse operation on the in-core portion of the mesh according to the multiple choice optimization strategy. Each edge collapse removes two triangles from the mesh. $N_{\text{current}} \leftarrow N_{\text{current}} - 2k$

- **WRITE$(k)$** removes $k$ triangles from the in-core portion and writes them into the output stream. $N_{\text{current}} \leftarrow N_{\text{current}} - k$

While the algorithm is running, these three operations are applied in arbitrary order. The only hard restriction is that the buffer must not overflow which implies that if $N_{\text{current}} = N_{\text{max}}$, we have to apply DECIMATE or WRITE before reading the next input triangle. Some additional weak restrictions are that the filling level should be kept as high as possible to provide a sufficiently large set of candidates for the multiple choice optimization and that the number of DECIMATE and WRITE operations should be balanced such that the output mesh has the prescribed target resolution.
Let $p$ be the percentage to which the input mesh should be decimated and $x$ be the unknown number of input triangles. Obviously, the decimation algorithm has to perform $px$ WRITE operations and hence $(1 - p)x/2$ DECIMATE operations to process the complete input data. Even though the stream algorithm does not know the number of input triangles, we can still derive that the ratio between DECIMATE and WRITE operations is

$$\frac{\text{DECIMATE}}{\text{WRITE}} = \frac{1 - p}{2p}$$

which implies that the ideal randomized multiple choice algorithm is given by the below pseudo-code. This algorithm consists of three stages: the initial filling of the buffer (line 1), the actual stream processing (lines 2 to 5), and the concluding clearing of the buffer (lines 6 to 9).

\begin{verbatim}
READ (N_{max})

while input not empty
    random choice with probabilities $1 - p : 2p$
    A: DECIMATE (1) & READ (2)
    B: WRITE (1) & READ (1)

while buffer not empty
    random choice with probabilities $1 - p : 2p$
    A: DECIMATE (1)
    B: WRITE (1)
\end{verbatim}

### 4.3 Real Stream Algorithm for Decimation

For a real implementation of this ideal algorithm, we have to make a few modifications in order to make the algorithm run effectively. In the introduction we explained that while a stream algorithm is running, the data is split into three disjoint parts ($\mathcal{A}$: postfix of the input stream, $\mathcal{B}$: in-core portion, and $\mathcal{C}$: prefix of the output stream). In the case of mesh decimation these three parts are sub-meshes of the original data at different resolution levels (cf. Fig. 4.1).
Figure 4.2: During stream processing the mesh data is split into three parts. Part $\mathcal{A}$ (top) is the unread part of the input stream and part $\mathcal{C}$ (bottom) has already been written to the output stream. Part $\mathcal{B}$ (middle) is currently in-core. The two boundary polygons $P_{ab}$ and $P_{bc}$ must not be modified. This restriction disqualifies the shaded in-core mesh parts from the set of candidates for the decimation.

In order to maintain the global mesh consistency, the two boundary polygons $P_{ab}$ and $P_{bc}$ which represent the interfaces between the parts $\mathcal{A}/\mathcal{B}$ and $\mathcal{B}/\mathcal{C}$ respectively, must not be modified by the DECIMATE operations. For $P_{ab}$ this is necessary since no vertex should be removed before all its neighbors are read from the input stream. For $P_{bc}$ this restriction guarantees that the output mesh has a consistent triangle neighborhood relation, i.e. the neighborhood of a triangle is not further decimated after it has been written to the output stream (cf. Fig. 4.2). In our implementation the operators DECIMATE and WRITE take care of this consistency checking (cf. Section 4.3.3 and 4.3.4). Notice that, depending on the geometric shape of the input mesh, the boundary polygons $P_{ab}$ and $P_{bc}$ usually consist of many connected components.

Another issue is that with a pure random decision whether a DECIMATE or a WRITE operation is performed next, it can happen that a very small (high resolution) triangle is accidentally written to the output stream. This typically leads to strong variations in the vertex densities of the output mesh. In a real implementation we therefore have to make sure that the WRITE operation is only executed when a sufficient number of coarse
Finally in a real operating system the file and stream access is usually much faster if we read and write the data in larger blocks. Hence we do not want to apply the \texttt{READ} and \texttt{WRITE} operation one by one but rather in larger chunks.

\subsection*{4.3.1 Overall Procedure}

The major difference between the ideal and the real stream decimation algorithm is that the decision whether to \texttt{DECIMATE} or to \texttt{WRITE} is no longer random. Only the decision which candidate to \texttt{DECIMATE} or to \texttt{WRITE} still remains a random process. The blockwise \texttt{READ} and \texttt{WRITE} is achieved by introducing a "hysteresis" which means that we are not reading any data before the filling level $N_{\text{current}}$ is down to 50\%. However, whenever we actually perform a \texttt{READ} operation we read in the maximum number of triangles $N_{\text{max}} - N_{\text{current}}$ to raise the filling level to 100\%.

In order to rate the \textit{resolution level} $R$ of the current in-core portion of the mesh, we count the total number of triangles $N_{\text{in}}$ that have been read from the input stream until now, the number $N_{\text{current}}$ of triangles currently in the triangle buffer, and the number $N_{\text{out}}$ of triangles that have been written to the output stream until now. Since the target resolution is given by the percentage $p$, we know that each triangle in the output stream statistically (not geometrically) represents $1/p$ triangles from the input stream. Consequently, the $N_{\text{current}}$ triangles in the buffer represent $N_{\text{in}} - N_{\text{out}}/p$ many input triangles and hence we find the current in-core resolution level to be

$$R = \frac{N_{\text{current}}}{N_{\text{in}} - N_{\text{out}}/p}.$$ 

The stream algorithm has to make sure that \texttt{WRITE} operations are only executed when the resolution level of the in-core portion equals $p$. Notice that the above resolution estimate is based on a statistical argument. Since the resolution level of the in-core triangles usually varies between the two boundaries $P_{ab}$ and $P_{bc}$, we cannot make any statement about individual triangles. Nevertheless, since our candidate selection is based on a random (multiple) choice, we have to rely on the \textit{expected} (average) resolution level.

In the initialization phase of the algorithm, we have to fill the triangle buffer and then apply \texttt{DECIMATE} operations. Before the first triangle is written to the output
stream, the boundary polygon $P_{bc}$ is empty but as long as only a small portion of the mesh has been read, the boundary polygon $P_{ab}$ covers a relatively large part of the in-core mesh. Hence we have to be careful about the right filling level to avoid extreme over-decimation of the non-boundary parts of the in-core mesh. To achieve this, we use a special initialization loop that alternates reading and decimation until the in-core resolution level $R$ has reached the target value $p$.

\[
\text{set } n \text{ such that } \frac{1}{n} \geq p > \frac{1}{n+1} \\
\text{READ } (N_{\text{max}}/2) \\
\text{repeat } (n - 1) \text{ times} \\
\quad \text{READ } (N_{\text{max}}/2) \\
\quad \text{DECIMATE } (N_{\text{max}}/4)
\]

At the end of the loop we invariantly have a filling level of $N_{\text{current}} = N_{\text{max}}/2$. After the loop has been repeated $(n - 1)$ times we have read $\frac{n}{2} N_{\text{max}}$ triangles in total and no triangle yet written to the output stream. Hence the current in-core resolution level is $R = \frac{1}{n} \approx p$.

After the initialization, the main loop is processing the stream according to the following pseudo-code:

\[
\text{while } \text{input not empty} \\
\quad \text{READ } (N_{\text{max}}/2) \\
\quad \text{DECIMATE } ((1 - p) N_{\text{max}}/4) \\
\quad \text{WRITE } (\frac{p}{2} N_{\text{max}})
\]

Again, at the end of the loop we have $N_{\text{current}} = N_{\text{max}}/2$ and the in-core resolution level is always kept close to $R \approx p$. Notice that in line 3 of the above pseudo-code, we are performing $(1 - p) N_{\text{max}}/4$ edge collapses to remove $(1 - p) N_{\text{max}}/2$ triangles from the mesh such that the READ and DECIMATE operations together increase the filling level by $\frac{p}{2} N_{\text{max}}$ which is exactly the number of triangles that are then written to the output stream in line 4.
After all triangles have been read from the input stream, we have to clean the in-core buffer. If the in-core resolution level would be exactly \( R = \frac{1}{n} = p \) after the main loop, we could simply write out all remaining triangles. However, in order to account for the slight inaccuracy \( \frac{1}{n} - p = \varepsilon \geq 0 \) that has been made in the initialization loop, we have to decimate a few more triangles to get the exact number of output triangles \( N_{out} = p N_{in} \).

\[
\text{DECIMATE} \ ((1 - n p) N_{max}/4) \\
\text{WRITE} \ (n p N_{max}/2)
\]

The number of \( n p N_{max}/2 \) output triangles corresponds to the number of \( n N_{max}/2 \) input triangles that have been read in the initialization loop.

Notice that if the input mesh is not large enough or if the decimation percentage \( p \) is too small then the complete output mesh might fit into the in-core buffer. In this case the stream decimation reaches the end of the input stream while still in the initialization loop. Hence, the main loop is skipped and the buffer contents is written to the output stream. Also, the length of the input stream is most likely not an exact multiple of \( N_{max}/2 \) such that the stopping criteria for the initialization and the main loop have to check the actual number of read triangles.

### 4.3.2 Reading Input Streams

The \( \text{READ}(k) \) operation reads a sequence of \( k \) triangles from the input stream and inserts them into a half-edge mesh data structure [CKS98] representing the in-core part of the mesh. The computationally most expensive part of this procedure is to find the edge(s) in the boundary polygon \( P_{ab} \) where this new triangle has to be connected. If such an edge does not exist then the new triangle is inserted as an isolated triangle with all its three edges forming a new connected component of \( P_{ab} \). Otherwise the new triangle is connected to its neighbors and the boundary \( P_{ab} \) is updated. To speed up the neighbor search, the algorithm maintains a lookup table for all mesh vertices currently belonging to \( P_{ab} \). The error quadrics associated with each vertex of the new triangle are initialized or updated by adding the squared plane equation.
4.3.3 Multiple-Choice Decimation

As explained in Section 4.1, the concept of multiple choice optimization consists of sampling a random set of candidates in every step and choosing the best among them. No global sorting of candidates is needed. In the case of mesh decimation the overall set of candidates is the set of half-edges in the mesh. The quality of a candidate collapse operation is rated by using the quadric error metric introduced in [GH97]. Hence, in every decimation step we test for a small number (usually 5 to 15) of half-edges the potential quadric error and then execute that collapse with the smallest error. If $Q_p$ and $Q_q$ are the error quadrics of two vertices $p$ and $q$ connected by a half-edge pointing from $p$ to $q$ then after the collapse only the vertex $q$ survives and its associated error quadric is set to $Q_p + Q_q$.

As mentioned earlier, the decimation procedure must not modify the two boundary polygons $P_{ab}$ and $P_{bc}$. Hence all half-edges emanating from a boundary vertex have to be excluded from the candidate set. Also the half-edges pointing to vertices on the polygon $P_{ab}$ cannot be collapsed because the error quadric of its boundary end-vertex is not completely known yet. Half-edges pointing to boundary vertices on $P_{bc}$ however, can be collapsed because their error quadrics are known and they do not modify $P_{bc}$.

4.3.4 Writing Output Streams

The selection of the triangles to be written to the output stream is also implemented as a randomized multiple choice selection. For a random set of candidate triangles we evaluate their error quadrics and then choose the one with the largest error. In analogy to the last section we define the error quadric of a candidate triangle $T = [p, q, r]$ by simply adding the three quadrics $Q_p$, $Q_q$, and $Q_r$ associated with its three vertices. The actual quadric error value is then obtained by evaluating this quadric at the center of the triangle.

In principle, all triangles not touching the boundary polygon $P_{ab}$ could be candidates for output. However, in practice it turns out that with this large set of candidates, the randomized multiple choice selection will generate many small holes scattered all over the in-core mesh. As a consequence the components of the boundary $P_{bc}$ will cover large portions of the mesh and thus reduce the candidate set for decimation significantly (cf. Fig. 4.3). We therefore restrict the candidate set to those triangles adjacent to $P_{bc}$ (and
not adjacent to $P_{ab}$) since this guarantees that the number of connected components of $P_{bc}$ does not increase. Only in exceptional cases where no suitable triangle adjacent to $P_{bc}$ can be found, we allow all interior triangles to be candidates for output.

When we select a triangle and write it to the output stream, we change the positions of its three vertices to the optimal position indicated by the associated error quadric (minimum of the quadric). This improves the feature alignment of the vertices in the output mesh. Notice that by this final re-positioning we obtain precisely the output quality of those decimation schemes that apply full edge collapses in every step (while we are only using half edge collapses). The reason for this is that the intermediate vertex positions computed for the full edge collapses have no influence on the error quadric accumulation.

Since we have to preserve the mesh consistency along the boundary polygon $P_{bc}$, we can re-position each output vertex only once and then freeze its position for the neighboring triangles.

### 4.4 Results and Discussions

To make our results comparable to others we use a setup for our experiments in which the stream algorithm reads its input data from a file and writes it back to another file.
4.4 Results and Discussions

by using the UNIX pipe mechanism. All CPU-timings we give include these I/O times. Notice that in an integrated software system, the decimation could directly process the output of the mesh generation process and hence would run much faster since no disk access would be necessary at all.

We use a PC with 866 MHz P3 processor since a similar type of computer has been used by OOCSx [LS01] and OEMM [CRMS03] as well. For the timings in Table 4.1 we set the in-core buffer capacity to 400K triangles for the Buddha and David models (1200K triangles for the St. Matthew model) and the multiple choice decisions are always based on 8 random candidates. The resulting in-core memory consumption of the corresponding UNIX-process is 100MB (315MB for St. Matthew), which includes memory space for a half-edge based mesh representation of the in-core portion and various hash tables to accelerate random access and searching. This memory consumption is also comparable to OEMM.

<table>
<thead>
<tr>
<th>models</th>
<th>$T_{input}$</th>
<th>$T_{output}$</th>
<th>$p$ (%)</th>
<th>time (hh:mm:ss)</th>
<th>rate (t/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buddha</td>
<td>1,087,470</td>
<td>21,748</td>
<td>2</td>
<td>0:32</td>
<td>33.3K</td>
</tr>
<tr>
<td></td>
<td>217,038</td>
<td>20</td>
<td>0:29</td>
<td>30.0K</td>
<td></td>
</tr>
<tr>
<td>David head</td>
<td>4,000,885</td>
<td>80,016</td>
<td>2</td>
<td>2:04</td>
<td>31.7K</td>
</tr>
<tr>
<td></td>
<td>409,048</td>
<td>10</td>
<td>2:09</td>
<td>27.8K</td>
<td></td>
</tr>
<tr>
<td>David 2mm</td>
<td>8,254,150</td>
<td>82,541</td>
<td>1</td>
<td>4:19</td>
<td>31.6K</td>
</tr>
<tr>
<td></td>
<td>829,048</td>
<td>10</td>
<td>4:30</td>
<td>27.5K</td>
<td></td>
</tr>
<tr>
<td>David 1mm</td>
<td>56,230,343</td>
<td>562,219</td>
<td>1</td>
<td>32:40</td>
<td>28.4K</td>
</tr>
<tr>
<td></td>
<td>2,815,413</td>
<td>5</td>
<td>33:21</td>
<td>26.7K</td>
<td></td>
</tr>
<tr>
<td>St. Matthew</td>
<td>372,767,445</td>
<td>559,087</td>
<td>0.15</td>
<td>3:54:06</td>
<td>26.5K</td>
</tr>
<tr>
<td></td>
<td>1,863,837</td>
<td>0.5</td>
<td>4:01:27</td>
<td>25.6K</td>
<td></td>
</tr>
</tbody>
</table>

**Table 4.1:** Run-time performance of the stream algorithm including I/O times on a PC with 866 MHz P3 processor.

The observed decimation rate is about 28K triangles/sec which matches the average rates obtained by OOCSx on PCs for massive datasets. The rate is slightly sensitive to the input model size since a larger boundary polygon $P_{ab}$ increases the searching time when inserting new triangles into the in-core mesh. However, the sensitivity to the output model size is not as strong as it is for OOCSx where large intermediate
Figure 4.4: Decimation results for the Happy Buddha model (from left to right): stream decimation (18,486 tri.), OEMM (18,338 tri.), QEM (18,338 tri.) and OOCS (20,950 tri.). Their corresponding relative Hausdorff maximum errors (over the bounding box diagonal) are 0.89%, 0.89%, 0.87% and 1.12% respectively.

file sizes slow down the algorithm significantly. Compared to OEMM, our timings are about a factor 3 to 5 times faster and this factor even increases if we count the OEMM construction as part of the decimation.

On the 866 MHz PC the time for I/O access is responsible for approximately 50% of the total running time. If we run the same algorithm on a faster PC with 2.8 GHz P4 processor, the average decimation rates go up to 40K – 59K triangles/sec in the above experiments.

As we show in Fig. 4.4, 4.5 and 4.6, the quality of our output meshes is the same as obtained by OEMM and superior to the quality obtained by OOCSx with respects to visual appearances, error measurements and topology consistencies. Fig. 4.4 also reveals that our out-of-core stream decimation can generate comparable results with in-core QEM [GH97] simplification. To produce these figures we used the example data provided by P. Cignoni for the OEMM and QEM results and re-implemented OOCS [Lin00] which produces the same results as OOCSx.
4.4 Results and Discussions

Figure 4.5: Detail view of the decimation results for the David 1mm model with 56M triangles in the original data: stream decimation (562,219 faces, left), OEMM (500,000 faces, middle), and OOCS (578,503 faces, right). Top and bottom rows show the same models with different shading.

4.4.1 Discussions

One difficulty of our stream decimation algorithm is that triangles adjacent to the boundary polygon $P_{ab}$ have to be excluded from the decimation since the neighborhoods of the respective vertices are not yet complete. This however implies that boundaries in the original mesh cannot be decimated at all since the stream algorithm cannot distinguish true mesh boundaries and temporary boundaries in $P_{ab}$ before the input stream is read completely. Nevertheless since triangles adjacent to $P_{ab}$ are also no candidates for being written to the output, they stay in the in-core buffer until the input stream is empty and can then be decimated before the buffer is cleared.

The more serious limitation is that the in-core buffer has to store all the boundary triangles of the input mesh. In the case of the St. Matthew model this was the reason why we had to increase the buffer size: Due to imperfect 3D reconstruction this model has a large number of small holes and hence many boundary triangles. One possible
way to address this issue is to introduce a preprocessing step to tag such boundaries and this step might even be embedded in the model generation procedure.

4.5 Summary

To reduce the data size of massive geometric models which cannot be loaded into the main memory at once for distributed and mobile processing, in this chapter, we have presented a new out-of-core mesh decimation algorithm that works on arbitrarily large streams of triangle data. It combines high quality decimation results with high decimation performance by using incremental decimation based on the quadric error metric and by processing the input data in one single pass.

More specifically, our novel method overcomes many drawbacks of earlier techniques. First, the decimation is based on edge collapsing and quadric error metrics which guarantees a high quality of the output meshes. However in contrast to [CRMS03] our scheme uses the multiple choice optimization strategy and no spatial splitting of the input mesh is necessary. On the other hand, our algorithm is as fast as the out-of-core vertex clustering technique in [LS01]. This is achieved because our stream algorithm processes the data in one single pass. The memory requirements of our algorithm are independent from the size of the input and the output.

We have used the quadric error metric in our implementation for efficiency reasons. However, the algorithmic structure is flexible enough to include any other more sophisticated distance metric as well [KCS98]. Another convenient feature is that our algorithm can prescribe exactly the number of output triangles while vertex clustering techniques can only prescribe approximate values. In our experiments, all massive datasets could be processed properly without any pre-sorting. It has turned out in practice that all above flexibility and convenience are important especially for distributed and mobile geometry processing applications since for these applications, we always have to face almost all kinds of computing and quality requirements.
Figure 4.6: Results shown with topology errors (red/dark dots) for St. Matthew model by stream decimation (left, 1,863,837 tri.) and OOCS (right, 1,910,970 tri.). Note OOCS will generate much more topology inconsistencies than our method.
5 3D Digital Watermarking

Security is another important theme for distributed and mobile processing systems, when 3D geometry data are transmitted and distributed over the network, especially on the Internet. Like all digital multimedia data types as audio, images or video, the 3D digital geometric models can be easily copied and disseminated: digital copies can be made identical to the original and later easily reused or even manipulated. Thus there is an increasing need for efficient methods allowing for copyright protection and ownership assertion of digital multimedia information. And then it is in this context where digital watermarking techniques come to our hand.

In principle watermarks is a distinguishing piece of information adhered to the data to protect them, difficult to extract or remove once watermarked [PGH99]. Traditional watermarks have long been used in our daily life like stamps, seals, or watermarks on the cash notes.

Nowadays, digital watermarking is an established way to provide copyright protection and ownership assertion in the area of steganography. Most available digital watermarking techniques have been focusing mainly on classical media data types like audio, images and videos because of the dominancy of these data on the Internet [KP00, CMB01]. Due to their regularly parametrized functional representations, most watermarking schemes are based on spread-spectrum methods with signal processing, i.e., the media data have to be transformed into a spectral domain, then the coefficients corresponding to the most perceptually salient basis functions will be modulated with watermarks to achieve robustness against possible attacks.

Extending the above spectral watermarking methods to the newly emerged multimedia data type of 3D geometry models is difficult mainly because of the lack of basic 3D signal processing tools like filtering, regular parametrization and frequency analysis. On the other hand, with a drastically increasing availability of 3D datasets and practical geometric applications in recent years, the need for efficient watermarking schemes for 3D
models becomes more eminent. Although previous adapted 3D spectral watermarking approaches already exhibit good resistance and robustness to many types of real-world attacks, they are often far too slow to cope with nowadays large meshes due to the involved complicated numerical computations.

![Image of Iphigenie model](image_url)

**Figure 5.1:** The Iphigenie model (left, 1.01M vertices, 2.02M triangles) is watermarked in 86 seconds (middle) using only 100 orthogonal basis functions. Although without perceptible visual differences, these two meshes have a maximum Hausdorff distance of 1.64% of the major bounding box diagonal length (right, distances with color coding - blue minimum and red maximum).

In this chapter, we will present a novel imperceptible spectral watermarking scheme to support ownership claims on parametric triangle meshes of given 3D shapes [WK05a] (cf. Fig. 5.1). In fact, our method can be easily extended to explicit point-based geometry as well. Extensive tests have proved that our watermarking scheme exhibits almost the
same quality and robustness against various attacks as other recent spectral approaches. On the other side, by utilizing a new set of orthogonal basis functions derived from \textit{radial basis functions} (RBFs), our scheme runs much faster by two orders of magnitude, hence can watermark very large models more efficiently.

\section{5.1 Related Work}

Most previous works to watermark 3D models have been trying to mimic the common spectral approaches in some alternative ways, though early works on 3D watermarking even did not utilize the spectral idea. Watermarks were embedded into 3D meshes by directly modifying either the geometry, the vertex positions, the topology, the vertex connectivity [OMA98, HB02], or the surface normals [Ben99]. Simple enough, these kind of methods usually can not provide enough robustness with respect to many different types of ordinary attacks.

Kanai et al. [KDK98] decomposed the target mesh into a spectral domain by applying the lazy \textit{wavelet transform} proposed by Lounsbery et al. [LDW97]. Wavelet coefficients were then modified to embed watermarks. Extending this work, a blind watermarking algorithm was presented more recently [UCB04] that can ignore the original mesh information on the detector side. One constraint to these methods is that the input mesh is limited to have a prerequisite semi-regular subdivision connectivity.

\textit{Multiresolution analysis} is another way to construct the spectral-like domains. While Praun et al. [PHF99] used standard mesh simplification [GGK02] to construct multiresolution hierarchies, Yin et al. [YPSZ01] adopted the scheme in [GSS99] to perform the multiresolution decomposition. Watermark information can be embedded into some spatial kernels of the low-frequency component of the shape corresponding to the low-resolution representation in the geometry hierarchy.

Recent spectral domain watermarking algorithms employed the \textit{spectral mesh analysis} first proposed by Karni and Gotsman [KG00]. Eigenvectors of the Laplace matrix to the input mesh, the Laplace basis functions, can span an ideal spectral space for robust watermarking, i.e., leading coefficients corresponding to smallest eigenvalues can be modulated with watermarks [OMT02]. Later this idea was generalized to watermark
the topology-free point-based geometry in [CWPG04] where $k$-nearest neighbors have to be constructed to compose substitute Laplace matrices.

Despite the variety of available mesh watermarking algorithms, to our knowledge, none of them has reported to be able to watermark meshes with more than $10^4$ vertices mainly because of their involved complicated numerical computations. We will present a fast yet robust spectral watermarking algorithm based on the orthogonalization of a small set of radial basis functions that can efficiently handle large meshes even with more than $10^6$ vertices (cf. Fig. 5.1).

Except for our geometry dependant basis functions derived from RBFs, we note that other bases can have similar functionality as ours such as the harmonic functions computed with mesh Laplacian which were recently used for surface deformation and shape approximation [SCOIT05, ZRKS05]. Other than the necessary post orthogonalization step, they still have to solve sparse linear systems of Laplace equations, which makes them less efficient as ours. In addition, other than in digital watermarking, the fact that altering the low frequency components of a shape remains nearly invisible to the human eyes has also been observed in mesh compression [SCOT03].

### 5.2 Overview of the Watermarking Algorithm

Following the most successful spread-spectrum watermarking idea, we also watermark 3D meshes in the spectral domain. The whole watermarking scenario which is typically composed of the watermark embedding and the watermark extraction steps, is illustrated in Figure 5.2 similar to [OMT02, CWPG04].

The major difference of our scheme compared to previous work is that we use a new set of geometry dependant orthogonal basis functions derived from radial basis functions (cf. Section 5.3) to span the spectral space rather than using Laplace basis functions which emerge from the Laplace matrix that depends only on the mesh connectivity. We will present a fast algorithm to generate these orthogonal basis functions. Compared with the time-consuming eigensolvers for Laplace matrices, our method runs faster by two orders of magnitude and thus can efficiently watermark very large meshes as they are common today. Having the new orthogonal basis functions in hand, the remaining watermarking procedures are quite similar to other spectral watermarking approaches.
5.2 Overview of the Watermarking Algorithm

The watermark embedding phase (cf. Section 5.4) first computes a small set of our new orthogonal basis functions. Then the geometry of the original mesh is projected to these basis functions spanning the spectral domain to acquire a set of corresponding spectral coefficients. Watermarks will be encoded into the leading coefficients which can be used later to reconstruct the watermarked mesh together with the unmodified coefficients. Finally this watermarked mesh will be published with licenses if necessary and possibly receive some attacks.

To assert ownership, the watermark extraction step (cf. Section 5.5) has to be performed. The possibly attacked test mesh is first aligned by a registration routine and resampled by projections according to the original watermarked mesh. Then this modified attacked mesh will be transformed to the same spectral domain as in the embedding phase. Watermarks can be extracted by comparing the current and original coefficients. The correlation between the extracted and original watermarks will also be found to draw the final ownership assertion.

Figure 5.2: A typical watermarking scenario.
5.3 New Orthogonal Basis Functions

The basic idea of spectral watermarking is to represent the geometric information of a mesh with respect to a special basis such that most of the information is captured in just a few coefficients. These coefficients are then used to embed the watermark.

More precisely, let $\mathcal{M}$ be a given mesh with vertex positions $p_1 \ldots p_n \in \mathbb{R}^3$. Then we want to find a basis $B_1 \ldots B_n \in \mathbb{R}^n$ such that

$$(p_1 \ldots p_n)^T = \sum_{j=1}^n B_j c_j^T$$

with coefficients $c_j \in \mathbb{R}^3$. The particular basis should have the property that the approximation error

$$E_k = \| (p_1 \ldots p_n)^T - \sum_{j=1}^k B_j c_j^T \|$$

decreases as quickly as possible. A very bad example is the canonical basis $B_j = (0 \ldots 1 \ldots 0)^T$ for which the error $E_k$ decreases only linearly in $k$. A much better example is the set of basis functions which emerge as eigenvectors from the topological Laplace matrix defined through the connectivity of the given mesh. This basis has been used in [KG00] where it has been shown empirically that the approximation error $E_k$ decreases so fast that only a few hundreds to thousands of basis coefficients are sufficient to encode even fairly complex shapes [BCG05].

Another important property of the basis is orthogonality since modifying the coefficients of orthogonal basis is always stable. In this case the coefficients $c_j$ can simply be computed by a dot product

$$c_j = (p_1 \ldots p_n) B_j.$$ 

Both, the canonical as well as the Laplace basis are orthogonal. However, both bases have the important drawback that their basis function definition completely ignores the geometry of the input shape. Although the Laplace basis does depend on the mesh connectivity, the geometric shape of the input mesh only plays a minor role under the assumption that the mesh quality (i.e., the shape of the triangles) establishes a loose correlation between geometric and topological distance.

In this paper we are constructing a particular orthogonal basis which, on the one hand is optimized for the shape of the input geometry and on the other hand concentrates as
5.3 New Orthogonal Basis Functions

**Figure 5.3:** The Rocker arm mesh (left-most, 40K vertices) approximated with 100 orthogonal basis functions based on RBFs using decimated centers (red dots, middle left) and the random uniform centers (middle right). The right-most image shows the vertex-to-vertex deviation vectors from the random center RBF approximation to the original model where vectors pointing inside the object are flipped and rendered as green lines. Notice the good approximation quality of only 100 basis functions and the small differences between two different center placement strategies as well.

much geometric information as possible in as few coefficients as possible. We start by defining a pre-basis \( \{B_j\} \) and then augment it later.

We are using radial basis functions (RBFs) [TO99, CBC+01, MYR+01, TO02, OBS03, OBS04, TRS04] to capture the mesh geometry information. Let \( q_1 \ldots q_k \) be a set of 3D positions scattered in the vicinity of the mesh surface. There are many possibilities to define these positions, e.g., by selecting the remaining points in \( p_i \) after the mesh decimation with halfedge collapses, or even randomly selecting a uniform subset of the vertices \( p_i \). (See Figure 5.3 for a comparison, if not otherwise stated, we will always use random uniform selection for efficiency reasons). The \( q_j \) are used as the centers of a set of radial basis functions

\[
\phi_j(p) = \phi(\|p - q_j\|)
\]

where we choose \( \phi(\cdot) \) to be a monotonically decreasing function with compact support, e.g. \( \phi(r) = (1 - r)^4(4r + 1) \) like in [Wen95, OBS04] with \( r = \|p - q_i\|/\sigma \) and \( \sigma \) its support size (usually half the length of main bounding box diagonal to prevent singularity of the later composed matrix \( B \)).

If we evaluate these radial basis functions at all vertex positions \( p_i \), we obtain our set of discrete pre-basis functions

\[
B_j = [\phi_j(p_1) \ldots \phi_j(p_n)]^T.
\]
Due to the compact support of $\phi(\cdot)$, the discrete pre-basis functions are very likely to be linearly independent in $\mathbb{R}^n$. Moreover if the radial basis functions $\phi_j(\cdot)$ have a sufficient overlap then it turns out that a relatively small number $k$ of pre-basis functions are already sufficient to represent the geometric information of the input mesh fairly accurately (cf. Fig. 5.3), i.e.,

$$\begin{equation}
(p_1 \ldots p_n)^T \approx \sum_{j=1}^{k} B_j c_j^T.
\end{equation}
$$

Moreover, if we choose the distribution of the centers $q_j$ fairly uniform (and we assume that the vertices $p_i$ are also distributed uniformly over the mesh surface) then it turns out that the coefficients $c_j$’s magnitudes do not differ too much.

Notice that the choice of the actual number $k$ of pre-basis functions has influence only on the computation time of the subsequent orthogonalization step and not on the applicability of the approach in general.

Remember that our goal is to construct an orthogonal basis which concentrates most of the geometric information in just a few coefficients. In order to obtain this basis we
compute a singular value decomposition (SVD [PTVF95]) of the matrix $B \in \mathbb{R}^{n \times k}$ which has the pre-basis functions $B_j$ as its columns. With this decomposition we find

$$
(p_1 \ldots p_n)^T \approx B (c_1 \ldots c_k)^T = U W V^T (c_1 \ldots c_k)^T = U (c'_1 \ldots c'_k)^T.
$$

Hence the orthogonal columns of $U$ form a new set of discrete basis functions. The corresponding coefficients $c'_j$ are obtained by first multiplying the initial coefficients $c_j$ by the orthogonal matrix $V^T$ and then multiplying the $j$-th coefficient with the $j$-th singular value of $B$.

With increasing overlap, i.e., with increasing radius of the radial basis function $\phi_j(\cdot)$, we observe a stronger and stronger decay of the singular values of $B$ and hence a more and more pronounced concentration of the geometric information to just a few leading coefficients (cf. Fig. 5.4). In fact, suppressing the later coefficients $c'_l \ldots c'_k$ (which have been multiplied by the smaller singular values) causes only an additional squared approximation error of

$$
E_l'^2 = \sum_{j=l}^k \|c'_j\|^2
$$

due to the orthogonality of the columns of $U$.

While the concentration effect observed with our orthogonalized basis is similar to the effect observed with the eigenbasis of the Laplace operator, the advantage of our approach is that its computation is significantly faster. In fact, computing the eigenbasis of a very large sparse matrix is a numerically challenging task and usually large meshes are split into smaller patches and processed separately in order to cope with this problem [KG00, OMT02].

In our case, however, we do not have to analyze the large $(n \times k)$ matrix $B$ directly. Instead it is sufficient to decompose the much smaller symmetric $(k \times k)$ matrix $B^T B$ into

$$
B^T B = V W U^T U W V^T = V W^2 V^T.
$$

With this decomposition we can easily find the orthogonalized basis by

$$
U = B V W^{-1}.
$$

Since $k$ is usually much smaller than $n$, the time spent to compute the SVD is negligible and the computation time is dominated by the multiplication of $B$ and $V$. 

67
5.4 Watermark Embedding

Similar to other spectral watermarking approaches, our watermarking scheme embeds the digital watermarks by modifying the low-frequency components of a given shape in the spectral domain. As we have discussed in Section 5.3, the new orthogonal basis functions \( \{ B_i \} \) derived via SVD will be used to decompose the input mesh into a spectral representation and the coefficients of the leading part of the spectrum which are more robust against attacks, then can be modulated. The watermarked mesh is later produced with an inverse transform using the same basis functions and is ready to be distributed.

More specifically, given the original (large) input mesh \( \mathcal{M}_o \) with \( n \) vertices, a set of \( k \) orthonormal basis functions \( \{ B_i \} \) is first computed to span a spectral domain. Then spectral analysis is performed by projecting the mesh geometry (all three \( x, y, \) and \( z \) components) onto each basis function \( B_i \) to produce \( 3k \) mesh spectral coefficients \( \{ \alpha_i^{(x)} \}, \{ \alpha_i^{(y)} \} \) and \( \{ \alpha_i^{(z)} \} \), i.e.,

\[
\alpha_i^{(d)} = \sum_{j=1}^{n} p_j^{(d)} B_{i,j}, \quad i = 1...k, \quad d \in \{ x, y, z \}.
\]

(5.1)

where \( B_{i,j} \) denotes the \( j \)-th entry of the \( i \)-th basis function. Based on these, a spectral mesh representation \( \mathcal{M}_o' \) (cf. Fig. 5.3 middle) can in turn be assembled to approximate the original mesh \( \mathcal{M}_o \) with an inverse transform:

\[
\mathcal{M}_o' = \sum_{i=1}^{k} \alpha_i B_i.
\]

The approximation quality of \( \mathcal{M}_o' \) is already quite good. But as we only use a small number of basis functions \( (k \ll n) \), we still observe small vertex-to-vertex differences between the approximation \( \mathcal{M}_o' \) and original \( \mathcal{M}_o \) (cf. Fig. 5.3 right). These differences \( \Delta \) will be recorded separately to help the later reconstruction, i.e.,

\[
\Delta = \mathcal{M}_o - \mathcal{M}_o' = \mathcal{M}_o - \sum_{i=1}^{k} \alpha_i B_i.
\]

The watermarks are a binary bit string \( \{ b_i \} \) with length \( 3m \) and \( m < k \). To embed watermarks into the spectral representation, we first convert \( \{ b_i \} \) to a sign string \( \{ b'_i \} \) with \( b'_i = -1 \) for \( b_i = 0 \) and \( b'_i = 1 \) for \( b_i = 1 \). Then the first \( 3m \) spectral coefficients are modulated as

\[
\beta_i^{(d)} = \alpha_i^{(d)} \cdot (1 + b'_{si+d} \cdot \rho), \quad i = 1...m, \quad d \in \{ x, y, z \},
\]

where \( \rho \) is a small positive constant.
where $\rho$ is the *watermarking amplitude*.

Finally, these modulated coefficients $\{\beta_i\}$, together with the unchanged ones $\{\alpha_i\}$ and the differences $\Delta$, are composed to *reconstruct* the output watermarked mesh $M_w$,

$$M_w = \sum_{i=1}^{m} \beta_i B_i + \sum_{i=m+1}^{k} \alpha_i B_i + \Delta.$$  

### 5.5 Watermark Extraction

The watermarked mesh $M_w$ will be distributed with licenses if necessary and will possibly receive some types of real world attacks. To assert ownership of this attacked test mesh $M_t$, previous embedded watermarks have to be extracted as we will discuss in the following.

The watermarked mesh $M_w$ will be distributed with licenses if necessary and will possibly receive some types of real world attacks. To assert ownership of this attacked test mesh $M_t$, previous embedded watermarks have to be extracted as we will discuss in the following.

To undo a possible similarity transform or translation attacks, the attacked mesh $M_t$ first has to be aligned with the watermarked mesh $M_w$. We have used a typical *registration* toolbox developed in [Ale03] to compute an affine map for the final alignment. In principle it needs user intervention to define three matching point pairs to compute initial absolute orientations followed by an automatic iterative closest point (ICP) algorithm [BM92] to improve the initial registration till a local error minimum is reached.

After registration, a *resampling* phase is usually necessary to deal with those attacks that may modify the mesh topology like simplification or remeshing. The goal is to map the original topology of $M_w$ to $M_t$ as our basis functions $\{B_i\}$ are defined in the vertex indices order. Only when the two meshes have the same vertex order, the comparisons between their spectral coefficients to extract the watermarks will be reasonable. Because the registration has already minimized the distances between $M_t$ and $M_w$, we can use a simple nearest point search strategy to obtain the resampled test mesh $M'_t$: the topology of $M'_t$ is the same as $M_w$ and each vertex $v_i'$ of $M'_t$ is fixed as the nearest point of the corresponding vertex $p_i$ of $M_w$ on the mesh surface $M_t$. Note that like [PHF99] this
resampling step also marks vertices as cropped if the nearest distances are larger than a user-defined threshold to account for possible cropping attacks.

Having the registered and resampled test mesh $M'_t$, we perform a similar spectral analysis to it as Equ. (5.1) to get another set of $3m$ spectral coefficients $\{\gamma_i^{(x)}\}, \{\gamma_i^{(y)}\}$ and $\{\gamma_i^{(z)}\}$. Then for each $x, y$ or $z$ component, watermarks are extracted as follows,

$$c_i^{(d)} = \begin{cases} 
1, & \text{if } \gamma_i^{(d)} > \alpha_i^{(d)} + \varepsilon, \\
0, & \text{if } \gamma_i^{(d)} < \alpha_i^{(d)} - \varepsilon, \\
N/A, & \text{otherwise,}
\end{cases}$$

where $\varepsilon$ is the detection sensitivity and is set to $0.1\rho$.

Since various attacks might disturb some of the embedded watermarks, bitwise comparisons have to be performed to create the correlation between embedded watermarks $\{b_i\}$ and extracted watermarks $\{c_i^{(d)}\}$, i.e., the correlation will be

$$R = \frac{1}{3m} \cdot \sum_{i=1}^{m} \sum_{d=x}^{z} (c_i^{(d)} == b_{(3i+d)}).$$

The final ownership assertion can then be made as in [CWPG04]. If the correlation $R$ is larger than a specified threshold (e.g. 0.75), we affirm that the attacked test mesh $M_t$ contains the originally embedded watermarks.

### 5.6 Watermarking Results

We have tested our watermarking algorithm on various typical 3D mesh datasets. The overall performance including the timing measurement and memory estimation is first to be discussed. The watermarking robustness of our scheme will be verified with a lot of diverse real world attacks. A comparison between our orthogonal basis functions based on RBFs and the Laplace basis functions will also be conducted. All experiments shown here are run on a commodity Linux PC with a 3.2GHz P-IV CPU and 2GB main memory. Some important parameters are usually set as following if not otherwise specified: basis function number $k = 100$, watermark length $3m = 24$ and watermarking amplitude $\rho = 0.01$. 
5.6 Watermarking Results

Figure 5.5: Computation times for embedding the watermarks into various models.

5.6.1 Overall Performance

Computation times of our watermark embedding process are measured and summarized in Figure 5.5 as a function of input model sizes. Other recent spectral watermarking schemes like [OMT02, CWPG04] need much longer running times than ours, as they have to solve large eigensystems which will be impractical for meshes with more than $10^4$ vertices. Even when taking the differences of computing hardware into account, we still find that (e.g. for the bunny model reported in both papers) our algorithm runs more than 100 times faster, which are two orders of magnitude.

Actually, as our watermarking method runs so fast, experiments show that the performance bottleneck arises from the memory consumption. For example, our non-optimized watermarking proto-system needs more than 1GB Linux process memory to deal with the Iphigenie mesh and the basis functions themselves require about 400MB space already. However, this is just the reality that all spectral-based approaches have to face as basis functions always need to be computed for both watermark embedding and extraction phases.

In addition, as our watermarking scheme only modifies the “low-frequency” component of a given shape, the visual differences between the original mesh and the watermarked mesh are almost imperceptible (cf. Fig. 5.1, 5.6 and 5.10). And this will make the watermarking scheme more robust to malicious attacks.
Figure 5.6: The Max model (left, 200K vertices), watermarked mesh (middle) and the test mesh under both similarity transform and simplification attacks (right, 1% of original size). The extracted correlation is $R = 1$.

5.6.2 Robustness Against Attacks

We test our watermarking method with different mesh models to verify the robustness of embedded watermarks under various types of real world attacks. The watermarks will be generated randomly many times for the same testing object and the extracted correlation $R$ (cf. Eqn. 5.2) will be the mean value. If the correlation is larger than 0.75, the ownership will be claimed.

**Similarity Transforms** can be handled by the first registration process (cf. Fig. 5.6). As our registration results are precise enough, the similarity transform attack will have very few influences on the correlation $R$;

**Cropping** is dissolved by the resampling process to mark the cropped vertices and neglect them in the basis functions when performing the spectral analysis. The correlation $R$ is always 1 (cf. Fig. 5.7);

**Simplification** is simulated with a standard QEM-based greedy decimation scheme [GH97]. The resampling process will equalize the mesh topology for watermark extraction (cf. Fig. 5.8). Note that even under extreme simplification, ownership still can be successfully asserted (cf. Fig. 5.6 and 5.8);

**Additive noise** will randomly modify the vertex positions in the normal directions. Figure 5.9 shows that our watermarking scheme is robust against strong noises;
5.6 Watermarking Results

Figure 5.7: The cropped Horse model (left, 51K vertices) and the cropped Igea model (right, 134K vertices). The correlation is $R = 1$ in both cases.

Remeshing attack tries to improve the shape quality of mesh primitives where the initial topology will be greatly destroyed (cf. Fig. 5.10). Thus the resampling process is also necessary for watermark extraction.

Smoothing will apply certain iterations of Laplace operators [Tau95] to the watermarked mesh. Figure 5.11 compares the robustness effects of different iteration numbers to the extracted correlations.

The above testing scenarios show that we can correctly assert the ownership of the watermarked mesh. We also test our watermarking scheme with false-positive attacks, e.g., to perform the same watermark extraction step to a mesh without actually having embedded the designated watermarks or with other watermarks embedded. The extracted correlations are always below 0.5. Regarding the specified threshold, our watermarking algorithm will not incorrectly assert that a model is watermarked when it is not.

In summary, without perceptible differences between the original and watermarked meshes, our watermarking scheme is robust against a lot of different real world attacks, even the combined attacks as well (cf. Fig. 5.6), thus can lead to accurate ownership assertion and copyright protection.
5 3D Digital Watermarking

Figure 5.8: The Rocker arm mesh (in Fig. 5.3) under simplification attack (left, 2% of original size) and the resampled mesh (right). The correlation is $R = 0.96$.

Figure 5.9: The watermarked Dragon model (left, 438K vertices) and attacked by additive noise (right) with maximum deviation of 1% to main bounding box diagonal length. The correlation is $R = 1$.

5.6.3 Comparing to Laplace Basis Functions

Figure 5.11 compares our new orthogonal basis functions derived from RBFs to the Laplace basis functions (LBFs), the (leading) eigenvectors of the Laplace matrix of the input mesh [KG00] adopted by recent spectral watermarking schemes [OMT02, CWPG04]. It is interesting to see that our basis functions can capture more shape details than LBFs with the same number of basis functions. This is because ours are geometry-aware derived functions while LBFs are merely derived from the mesh topology. On the other hand, it is still easy to find that our watermarking scheme can
5.7 Summary

Figure 5.10: The Isis mesh (left, 188K vert.), watermarked mesh (middle) and remeshed mesh (right, 8K vert.). The correlation is $R = 1$.

have almost the same robustness against attacks and watermarking quality as previous spectral methods based on LBFs.

5.7 Summary

In this chapter, we have presented a novel spectral watermarking scheme using a new set of orthogonal basis functions derived from radial basis functions. Security can be ensured for distributed and mobile geometry processing applications by providing copyright protection and ownership assertion of the digital multimedia data. Moreover, while observing similar watermarking quality and robustness against attacks as other related approaches, our method runs much faster by two orders of magnitude thus leading to efficient watermarking of very large models.

The proposed watermarking algorithm is fast, but it is also very flexible. In fact, not only for parametric triangle meshes, we also find that our watermarking technique using
RBFs can be easily applied to explicit point-sampled geometry by slightly changing the extraction process as adding registration and resampling tools for point sets. Since RBFs are used, the method can also be extended for watermarking higher dimensional data, e.g. points with colors, meshes with texture coordinates, and even animated meshes.
Figure 5.11: The Bunny mesh (35K vertices) approximated in first row with 100 LBFs (left) and RBFs (right) and watermarked in second row with corresponding basis functions. The two bottom rows show smoothing attacks on RBF watermarked mesh with 5, 15, 50 and 100 iterations of Laplace operators respectively from top to bottom and left to right. The extracted correlations are 1, 0.96, 0.83 and 0.8 respectively. Under same attacks, the LBF watermarking correlations are 1, 0.96, 0.83 and 0.83.
Part II

Explicit Representations
6 Optimized Splat Sub-Sampling

The explicit geometric structures we are considering in this thesis include point models (set of infinitesimal points) and splat models (set of overlapped 3D ellipses) (cf. Section 2.2). Unlike the standard parametric triangle meshes that have to be connected with each other, point or splat models do not observe any topology restrictions. This advantage is critical as it allows for partial data loss and is independent from packet order during the network transmission, which makes distributed and mobile processing systems more robust. In addition, having more flexibility, explicit geometric data can be feasibly applied with more sophisticated optimization schemes to achieve higher output quality.

On the other hand, due to the limited linear approximation power of explicit point models, nowadays point sets usually contain a large number of points in order to have an acceptable approximation quality. In this respect, aiming at compact explicit geometric data structures, we will present in this chapter (cf. [WK04]) an optimized algorithm for sub-sampling of point sets which is especially designed for splat-based representations and makes full use of their flexibility (cf. Fig. 6.1). Since no topological consistency conditions between nearby splats have to be observed, we will apply an unconstrained global optimization technique. By this we can considerably reduce the splat count for a given model while still observing a prescribed error tolerance. In contrast to previous approaches we are not focussing merely on the relation between splat centers but rather consider each splat as a linear surface segment with finite spatial extent.

6.1 Background and Motivation

As an explicit representation, point-based geometry has gained quite some attention recently due to its conceptual simplicity [GPA+03, GPZ+04, KB04]. Individual point
Figure 6.1: Optimized sub-sampling of the Iphigenie (left, 352K points) using 30181 circular splats. The error tolerance is set to 0.05% of the bounding box diagonal. The center figure is rendered with EWA-filtered splats and the right zoom-in figures show the sample density and distribution.

samples can be computed or measured on any kind of surface description and a sufficiently dense set of samples already provides an effective surface approximation that is suitable, e.g., for high-quality rendering [PZBG00, RL00]. The major advantage over ”classical” geometry representations is that point sets do not require any connectivity information. This significantly simplifies their processing since no topological consistency conditions have to be satisfied.

In order to guarantee a visually continuous appearance of the displayed objects, the concept of purely point-based representations is usually generalized to *splat-based models* [ZPBG01], i.e., infinitesimal points are replaced by ellipses or rectangles, either in object space or in image space. Due to their intuitive geometric interpretation, especially
object space techniques, like surface splatting, have been used widely in applications ranging from high-quality rendering and level-of-detail handling of complex models to interactive shape design [ZPBG01, ZPKG02, PKKG03].

Although surface splats require the derivation of attributes such as orientation and spatial extent for each splat, the conceptual simplicity of point-based representations is largely preserved. However, in a strict sense, we are dealing with a piecewise linear surface representation just like triangle meshes with the important difference that the linear pieces join in a $C^{-1}$ fashion rather than $C^0$.

On the other hand, in most approaches the generation of surface splat models still has the flavor of a sampling procedure since the position and normal information is considered to be associated with the center of each splat while the spatial extent (radius for circular splats and minor/major axes for elliptical ones) is just needed to bridge the gap between two neighboring samples. Our approach in this chapter, in contrast, takes the full geometry of the circular or elliptical object space splats into account when carefully placing samples on a given surface such that the collection of all splats forms a hole-free representation of the input geometry.

Even if surface splats can be processed extremely fast by exploiting the programmable features of current graphics hardware [DVS03, BK03], we would like to minimize the number of splats that are needed to represent a given object since the processing time is nevertheless proportional to the number of geometric primitives. This is why geometry simplification techniques have been developed for splat-based representations, most of them being derived from well-established mesh decimation or hierarchical clustering schemes [GGK02].

6.2 Previous Work on Explicit Geometry

Using points as rendering primitives was first introduced by Levoy et al. [LW85] and later followed and improved by [GD98, PZBG00, RL00, ZPBG01, WFP+01, BWK02] aiming at efficient and high-quality rendering of complex models. Most recently, various applications dealing with point-based geometry have been intensively discussed including shape acquisition and authoring [MPZ+02], high quality rendering [ZPBG01, LRZ02, BSK04],
Optimized Splat Sub-Sampling

(re-)sampling and simplification [SD01, PGK02], as well as as shape editing [ZPKG02, PKK03]. Complete surveys can also be found in [GPA+03, GPZ+04, KB04]

Pure point representations are discrete samplings of the input geometry and hence proper reconstruction filters have to be applied in order to enable hole-free rendering. Approaches applied in image space simply render “large” points as squares or circles [GD98, SD01, KV03b]. Object space approaches use (disk-shaped or elliptical) surface splats [ZPB01, LRZ02, BWK02, Paj03], quadratic [KV03a], or even higher-order [ABF+03, OBA+03] patches to reconstruct the surface geometry. In this chapter we are using circular or elliptical object space splats, since they seem to provide the best quality/performance tradeoff when implemented on programmable graphics hardware [LRZ02, BK03, ZRB+04].

Many existing sub-sampling methods focus only on the relation between splat centers and hence require extra effort to estimate the actual spatial extent of splats to fill the gap between samples. Linsen [Lin01] and Alexa et al. [ABF+03] adopted greedy schemes to iteratively remove samples from the input point cloud. The greedy nature of these algorithms cannot guarantee a globally uniform point distribution. Moenning et al. [MD03] use fast marching farthest point sampling for point set simplification. While it is simple and efficient, reliable error control is not supplied and seems not trivial to embed.

In [PGK02], Pauly et al. adapted various mesh simplification techniques to simplify point-sampled geometry which, however, cannot take an a-priori approximation error tolerance into account. Their pure greedy simplification produces results with small a-posteriori error but also with non-uniform sampling density. This is why they post-optimize the result with a particle simulation scheme which, alas, increases the approximation error. Our scheme uses a relaxation scheme that takes approximation error into account and uses the complete anisotropic geometry of the elliptical splats – not just their centers. In this sense it differs significantly from previous isotropic sampling approaches like [PGK02] or [SAG03].

Other approaches [RL00, BWK02, Paj03, KV03b] to splat simplification are based on hierarchical clustering schemes. The input points are rearranged in a hierarchical spatial partitioning data structure and splats are created for every node by analyzing the local surface properties. This technique is simple and fast but since there is no optimization
strategy involved, the results are usually overly conservative and tend to contain lots of redundant splats.

6.3 Sub-Sampling Overview

Let a surface $S$ be given by a set of sample points $P = \{p_i\}$ which are sufficiently dense in the sense that they form an $r$-sample of $S$ for some value $r < 1$ [ABK98]. If $r$ is chosen small enough then there exists a constant $k$ such that fitting a least squares plane to any point $p_i$ and its $k$ nearest neighbors yields a reliable estimate of the surface normal direction at $p_i$. A consistent normal orientation can be propagated via a minimum spanning tree [HDD+92]. We denote by $N_k(p_i)$ the set of $k$ nearest neighbors to $p_i$ measured by Euclidian distance and the graph $\mathcal{N} = (P, E)$ represents this non-symmetric neighborhood relation where the edge $(i, j)$ belongs to $E$ iff $p_j \in N_k(p_i)$. The actual distance $d_i = \|p_i - p_k\|$ to the $k$-th neighbor can be used as an estimate for the local sampling density and we associate a surface area element $\omega_i = \pi d_i^2$ with each sample point $p_i$. The graph structure of $\mathcal{N}$ can be computed most efficiently by using a hierarchical binary space partition [Sam90]. Our goal is to find a minimum set of surface splats $T = \{t_j\}$ that approximates $P$ with an error below some prescribed tolerance $\varepsilon$.

Depending on the application, the user can choose if circular or elliptical splats should be used. Elliptical splats can adapt to the local curvature of the surface usually leading to sparser sets $T$ but they need more storage space per splat. A circular splat $t_j$ is given by its center $c_j$, its normal vector $n_j$, and its radius $r_j$. For elliptical splats we replace the radius $r_j$ by two additional vectors $u_j$ and $v_j$ to define the major and minor axes.

For every input sample $p_i$ we measure its distance to $T$ by orthogonal projection onto the splats $t_j$, i.e.,

$$\text{dist}(p_i, T) = \text{dist}(p_i, t_j) = |n_j^T(p_i - c_j)|$$

if

$$\|(p_i - c_j) - n_j^T(p_i - c_j) n_j\|^2 \leq r_j^2$$ (6.1)

for circular splats or

$$\left(u_j^T(p_i - c_j)\right)^2 + \left(v_j^T(p_i - c_j)\right)^2 \leq 1$$ (6.2)

for elliptical splats. If $p_i$ projects into the interior of several splats, we choose the minimum distance. If (6.1) or (6.2) do not hold for any $t_j$ we set $\text{dist}(p_i, T) = \infty$. 

85
For a given set of splats $T$ and an error tolerance $\varepsilon$, conditions (6.1) and (6.2) imply a coverage relation $C_\varepsilon \subset P \times T$ which includes all pairs $(p_i, t_j)$ for which (6.1) or (6.2) holds and $\text{dist}(p_i, t_j) \leq \varepsilon$. We define the surface patch $Q_j = C_\varepsilon(*), t_j)$ corresponding to a splat $t_j$ as the set of all samples $p_i$ for which the relation $(p_i, t_j) \in C_\varepsilon$ holds. The area of the patch $Q_j$ is given by

$$\Omega_j := \sum_{p_i \in Q_j} \omega_i.$$ 

The optimized sub-sampling task can now be formulated as the minimum dominating set problem [CLRS01] for the 2-colorable graph $(P \cup T, C_\varepsilon)$ whose connectivity is defined by the coverage relation $C_\varepsilon$. Since the dominating set problem is known to be NP-hard, we have to find an approximate optimization algorithm.

Our algorithm proceeds in three steps which we explain in the following subsections. First, a maximum splat $t_i$ is computed for each input sample $p_i$ whose size is limited by the prescribed error tolerance $\varepsilon$. These splats are centered at $p_i$ in the sense that $p_i$ projects to the center $c_i$ but we do not require $p_i = c_i$. This relaxed condition turns out to be crucial for the effective reduction of the splat count. Notice that for extremely complex input data $P$ we can apply simple pre-decimation [PGK02] before generating the initial splats to speed up the computation. This does not have significant impact on the final result.

From the initial set of splats, we select a subset which safely covers the whole surface. This is done by a greedy procedure where the selection criterion guarantees that neighboring splats have sufficient overlap to provide a hole-free approximation of the input surface $S$. Since the error tolerance $\varepsilon$ has been taken into account in the splat generation step, any selection of splats $t_j$ such that the union of their corresponding patches $Q_j$ completely covers $P$ automatically satisfies the approximation tolerance.

In the third step, the greedy solution is further improved by a global relaxation procedure. The idea is to iteratively replace subsets of splats by new sets that have fewer elements or at least a better splat distribution. Notice that for circular splats, isotropic distribution is optimal while for elliptical splats anisotropic distribution according to minimum and maximum curvature is optimal. Since our algorithm takes minimum and maximum curvature into account when creating elliptical splats and since the relaxation is controlled by mutual overlap of the splats, we can guarantee to always produce a near-optimal splat distribution.
6.4 Initial Splat Generation

Initially we generate a circular splat \( t_i = (c_i, n_i, r_i) \) for each sample \( p_i \) in the (possibly pre-decimated) input set \( P \). Optionally this initial splat can then be extended into an elliptical splat \( t_i = (c_i, n_i, u_i, v_i) \) in the next step (cf. Sec. 6.4.1).

Starting with a seed point \( p_i \) we first estimate the local normal direction \( n_i \) by fitting a least squares plane to \( p_i \) and its \( k \) nearest neighbors [Jol02]. Then we grow the splat by adding neighboring sample points in the order of their projected distances (6.1) to \( p_i \). For each new point \( p_j \) we compute the signed distance

\[
h_j = n_i^T (p_j - p_i)
\]

and the growing stops as soon as the interval \([h_{\text{min}}, h_{\text{max}}]\) becomes larger than \(2 \varepsilon\). The center of the splat is then set to

\[
c_i = p_i + \frac{h_{\text{min}} + h_{\text{max}}}{2} n_i
\]

and the radius is set to

\[
r_i = \| (p_j - c_i) - n_i^T (p_j - c_i) n_i \|
\]

where \( p_j \) has the largest projected distance (6.1) before the prescribed error tolerance is violated (cf. Fig. 6.2).

![Figure 6.2: Growing a splat \( t_i \) initially centered at \( p_i \). Symbols ⋄, □ and ○ stand for conquered, front and un-covered samples respectively. The left figure shows a view in tangent direction and the right figure is viewed in normal direction.](image)

The splat growing procedure can be implemented quite efficiently by breadth first traversal of the neighborhood graph \( N \) in a "fast marching" manner. Since we do not update the normal direction \( n_i \) while the splat is growing, there is no need to update the
ordering of the neighbor samples or to recompute the signed distance interval \([h_{\text{min}}, h_{\text{max}}]\). Notice that re-fitting a least squares plane to the conquered sample points after the splat growing reduces the \(\| \cdot \|_2\)-error but it might actually increase the \(\| \cdot \|_{\infty}\)-error and hence could violate the prescribed error tolerance \(\varepsilon\).

### 6.4.1 Elliptical Splats

After a maximum *circular* splat \(t_i\) seeded at the point sample \(p_i\) is generated, it is optionally possible to continue the growing procedure into the minimum curvature direction to obtain an *elliptical* splat which better adapts to the local anisotropic surface curvature while still keeping the error tolerance (cf. Fig. 6.4). In addition to the center \(c_i\) and normal \(n_i\) we need two non-normalized tangent vectors \(u_i\) and \(v_i\) to define the major and minor axis of the elliptical splat according to (6.2).

In order to find a robust estimate of the (normalized) principal directions \(\gamma_{\text{min}}\) and \(\gamma_{\text{max}}\), we use the shape operator of [CSM03]. Since this operator is defined for triangle meshes, we triangulate the surface patch \(Q_i\) by projecting the corresponding sample points into the supporting plane of the splat and computing a 2D Delaunay triangulation.

With \(r_i\) being the radius of the initial circular splat \(t_i\), we define the minor axis of the elliptical splat by \(v_i = \gamma_{\text{max}}/r_i\).

\[
\lambda = \begin{cases} 
\lambda < 0 & \text{if } \lambda < 1 \\
\lambda > 1 & \text{if } \lambda > 1 \\
\lambda = 1 & \text{if } \lambda = 1
\end{cases}
\]

![Figure 6.3: A circular splat with center \(c_i\) and radius \(r_i\) is extended into an elliptical splat with semi-axes \(u_i\) and \(v_i\).](image)

Let \(\sqrt{\lambda}\) be the (unknown) aspect ratio of the resulting elliptical splat then we can order the neighboring samples \(p_j\) by increasing aspect ratio, i.e., by

\[
\lambda_j := \frac{(\gamma_{\text{min}}^T (p_j - c_i))^2}{r_i^2 - (\gamma_{\text{max}}^T (p_j - c_i))^2}
\]
6.4 Initial Splat Generation

where we do only consider samples for which the denominator is positive. We continue the splat growing procedure described in the last section but this time we add the neighboring samples $p_j$ ordered by increasing $\lambda_j$. Again, the growing procedure stops when the interval $[h_{\text{min}}, h_{\text{max}}]$ becomes larger than $2\varepsilon$ and if $p_j$ is the last added neighbor before the error tolerance is violated, the major axis of the elliptical splat becomes

$$u_i = \frac{1}{r_i \sqrt{\lambda_j \gamma_{\text{min}}}}.$$

![Figure 6.4: A torus approximated with error tolerance 0.2% by 510 elliptical splats (left and center) and by 734 circular splats (right).](image)

6.4.2 Hole-free Approximation

In the selection and optimization steps we will choose a subset of active splats from the initially generated set of candidate splats. During the process, we have to verify if the current subset of splats actually covers the whole surface $S$. This is difficult in general since we only have a discrete set of sample points. A much simpler condition is to check if each sample point $p_i \in P$ is covered by at least one splat. Testing the latter simply requires to iterate over all active splats, tag all covered samples based on the relation $C_\varepsilon$, and eventually check if there are un-tagged sample points left.

However, as shown in Fig. 6.5, covering all sample points $p_i$ by active splats is not sufficient to guarantee a hole-free approximation since holes can appear in-between the sample points. Hence we have to modify the definition of the patch which is safely covered by a splat $t_i$.

After the growing procedure for a splat $t_i$ stops, the set of conquered sample points defines a local surface patch $Q_i$ which projects into the interior of the splat defined by
the splat center \( c_i \) and its radius \( r_i \) or by its minor and major axes \( u_i \) and \( v_i \) respectively. As a by-product of the 2D Delaunay triangulation within the supporting plane of the splat (cf. Sec. 6.4.1) we can compute the convex hull of the projected sample points which also lies completely in the interior of the splat due to convexity. Now we define the interior patch \( \bar{Q}_i \) as the subset of \( Q_i \) which excludes all the boundary vertices, i.e., all samples that lie on the convex hull.

If we restrict our coverage relation \( C_\varepsilon \) to the interior coverage \( \bar{C}_\varepsilon \) then the test if all sample points \( p_i \) are covered by the current selection of active splats is a conservative estimate for the condition that the whole surface \( S \) is approximated in a hole-free manner. Here we assume that the initial sampling is sufficiently dense. Severely under-sampled regions should be considered as intentional holes in the surface \( S \).

Since the hole-detection as well as all the following operations are only based on the coverage relation \( \bar{C}_\varepsilon \), we can simply consider splats as sets of samples and we do not have to distinguish between circular and elliptical splats in the rest of the algorithm.

![Figure 6.5:](image)

**Figure 6.5:** The left figure shows that all sample points are covered but the small blue region between the samples remains as a hole. The middle and right figures show splat-based models with and without holes for the same error threshold. The green points on the left indicate the boundary points \( Q_j \setminus \bar{Q}_j \) of that splat.

### 6.5 Greedy Selection

From the set of candidate splats \( T \) we select an active subset \( T' \) that covers the whole surface \( S \), i.e., which satisfies

\[
P = \bigcup_{t_i \in T'} \bar{Q}_i.
\]
according to our above discussion of hole-free approximation. Since finding the minimum subset $T'$ is NP-hard [CLRS01] we have to find an approximate solution. As with most geometrical optimization problems, we find that a greedy selection strategy provides a suitable solution.

Similar to [DDSD03] we rank the splats according to their incremental surface area contribution. For every candidate splat $t_j \in T$ we sum the area elements $\omega_i$ associated with those sample points $p_i \in \bar{Q}_j$ that are not already covered by previously selected splats. Then in each step we select the splat which adds the maximum surface area to the active set and update the area contribution of the remaining candidates.

To implement the area contribution update, we explicitly store the relation $\bar{C}_e$ two times. Once organized by patches $\bar{Q}_j$ and once organized by sample points $p_i$. When a splat $t_j$ is selected, we iterate over all samples $p_i \in \bar{Q}_j$ and subtract the area $\omega_i$ from each patch that overlaps $p_i$. By this, the computation costs for the update only depend on the size of the splats and the degree of overlap but not on the total number of splats in $T$.

### 6.6 Global Relaxation

Even if it can be proven theoretically that the greedy selection produces a splat set $T'$ which is close to the optimum [Hoc96, Hro02], there are still situations where some splats are redundant (cf. Fig. 6.6). Moreover, the local decision about which splat to select usually causes a disturbing non-uniformity of the splat distribution (cf. Fig. 6.7). This has to be overcome by some global optimization scheme where we can exploit the fact that splat-based surface representations do not have to respect any consistency requirements. Hence we can add and remove splats in arbitrary order as long as we preserve a full hole-free coverage of the input samples.

Our global relaxation procedure mimics the behavior of repulsing particles on the surface [Tur91]. Since we do not have a continuous surface representation we have to use the neighborhood structure which is encoded in the connectivity of the neighborhood graph $\mathcal{N}$. This neighborhood relation between sample points can be transferred to splats since each splat $t_i$ is uniquely associated with the sample point $p_i$ which has been used
Figure 6.6: Greedy selection produces redundant splats. Splats $a$, $b$, $c$ and $d$ are selected consecutively based on their incremental area contribution (left). Eventually $a$’s area is fully covered by the remaining splats and could be removed (right). A greedy scheme cannot detect such situations since no backtracking beyond earlier decisions is possible.

as a seed when growing $t_i$. In this sense we misuse the notation $N_k(t_i)$ to refer to those splats $t_j$ whose seed points $p_j$ are in $N_k(p_i)$.

The local movement of a splat-particle $t_i$ is achieved by removing $t_i$ from the active set and replacing it with another splat $t_j \in N_k(t_i)$. The choice of the new splat $t_j$ is controlled by a local relaxation force. In contrast to previous approaches [Tur91, PGK02], we derive this force by taking the complete splat geometry into account and do not simply consider the relation between splat centers.

We use two operations to improve the splat distribution and to remove redundant splats. In the first operation we iterate over all active splats and check if there is another splat in the vicinity that has less overlap with its neighbors. In the second operation, we check for each splat if it can be removed, i.e., if the hole resulting from its removal can be re-covered by locally ”moving” the neighboring splats. The overall optimization procedure alternates between relaxation phases and redundant splat removal phases. Since the convergence in our experiments turned out to be quite fast, we are usually applying each phase just once. More iterations can still improve the result slightly but the effect per computation time is decreasing.

We introduce the kernel of a splat $t_i \in T'$ as the set

$$K_i := \bar{Q}_i \setminus \bigcup_{t_j \in T', j \neq i} \bar{Q}_j$$  \hspace{1cm} (6.3)

which consists of all sample points that are only covered by $t_i$ and not by any other active splat $t_j$. The kernel is critical to preserve the hole-free property of the active
Figure 6.7: A Female-torso model (left, 171K sample points) is approximated by 422 circular splats after greedy selection (middle two). Global relaxation further reduces the number of splats to 333 (right two). The figures show both, EWA-filtered splats for approximation quality and smaller splats for distribution quality. The error tolerance is \( \varepsilon = 0.47\% \) of the bounding box diagonal. Notice the improved splat distribution after the global relaxation step.

splat set \( T' \) during global relaxation: if we remove \( t_i \) we have to make sure that \( K_i \) is covered again after the relaxation.

Let \( V_j = \bar{C}_e(p_j, \ast) \cap T' \) be the set of currently active splats that cover a given sample point \( p_j \). Then the kernel \( K_i \) for a splat \( t_i \) consists of all points \( p_j \in \bar{Q}_i \) such that \( V_j \) contains just one element \( \{t_i\} \). In practice it turns out that it is more efficient to pre-compute the kernels \( K_i \) once in the beginning and then to update them after each relaxation step. The same holds for the coverage sets \( V_j \).

### 6.6.1 Relaxation

With this operation we are improving the regularity of the splat distribution. Since local non-uniformity in the sparse set \( T' \) is indicated by large overlap of neighboring splats, we iterate over all active splats and try to replace each splat \( t_i \in T' \) by another candidate splat \( t_j \in T \setminus T' \) such that the hole-free property is preserved.

Using the kernel notation for splats, we find that the set

\[
U_i := \{ t_j \in N_k(t_i) \mid K_i \subset \bar{Q}_j \} \subset T \setminus T'
\]
defines the set of choices for hole-free relaxation. Notice that $U_i$ cannot contain an active splat $t_j \in T'$ due to the definition (6.3) of the kernel.

The overlap area between two splats is measured by summing the surface area elements of all samples that are covered by both splats

$$\text{overlap}(t_i, t_j) := \sum_{p \in Q_i \cap Q_j} \omega_l.$$ 

Hence in each step, the relaxation replaces a splat $t_i \in T'$ by another splat $t_j \in U_i$ which minimizes the maximum overlap to any other active splat. Notice that it can turn out that $t_i$ already is a local optimum. In this case we simply keep $t_i$ active.

The relaxation loop over all active splats is performed several times until convergence (no more splat replacements take place) or until a maximum number of iterations is reached (10 to 15 in all our experiments).

To obtain the set $U_i$ we check for each $t_j \in N_k(t_i)$ if it is not active and if $K_i \subset \hat{Q}_j$. Then for each element $t_j \in U_i$, we find its maximum overlap to any other active splat $t_l$ from the set

$$W_j := \bigcup_{p_m \in \hat{Q}_j} V_m,$$

where overlap($t_j, t_l$) is computed by accumulating the area elements $\omega_m$ for all sample points $p_m \in \hat{Q}_j$ whose coverage set $V_m$ contains $t_l$.

By using hash-tables to associate splats $t_i$ with their indices $i$, the complexity of these local operations depends only on the number of samples per splat and not on the total number of splats.

Since the relaxation force is defined in terms of mutual overlap area, it turns out that the density of splats depends on their sizes. Hence, for circular splats, the relaxation generates a uniform and isotropic splat distribution and for elliptical splats the distribution is anisotropic according to the orientation and aspect ratio of the splats (cf. Fig 6.4).

### 6.6.2 Removing Redundant Splats

If the kernel $K_i$ of an active splat $t_i$ is empty then this splat can be removed without creating a hole in the splat-based surface approximation. If the kernel is not empty we
can still try to replace neighboring splats in a relaxation procedure such that the hole $K_i$ which is caused by removing $t_i$ is covered by those splats. Here we can exploit the full flexibility of splat-based representations since no topological restrictions have to be taken into account.

We iterate over all active splats in $T'$ and in each step we tentatively remove one splat $t_i$. In order to fill the resulting hole $K_i$, we apply a relaxation operator to all currently active splats $t_j \in W_i$ that have a non-empty overlap with $t_i$. The restriction to this set $W_i$ is not necessary but it accelerates the implementation since we can use the quick overlap computation procedure sketched in the last section. If after the relaxation the hole $K_i$ could not be closed, we backup to the splat configuration before the tentative removal of $t_i$ and continue with the next splat.

The relaxation force in this case favors splats that cover a maximum portion of the hole $K = K_i$. Just like in (6.4), we define for each splat $t_j \in W_i$ the set of possible choices $U_j$. Among those choices, we pick the splat $t_l$ that maximizes $\text{overlap}(t_l, K)$. After each relaxation step we have to update the remaining hole $K \leftarrow K \setminus \bar{Q}_l$ to account for the local change. Notice that the definition of $U_j$ automatically guarantees that no other holes can appear during relaxation.

6.7 Results and Comparisons

We have applied our sub-sampling technique to a wide range of different unstructured point-set models obtained from laser scanning to demonstrate the quality and performance. In all experiments we set the $k$-neighborhood to 10 such that the only user-defined input parameter is the error tolerance $\varepsilon$. The optimized splat sub-sampling so works fully automatic.

In Table 6.1 we report the overall running times of the sub-sampling algorithm measured on a Pentium4 2.8GHz CPU, 2GB memory system. Note that the additional time used for elliptical splats compared to the circular ones mainly results from computing the local surface curvature tensors. Due to the exact $\| \cdot \|_\infty$ error control in the splat generation phase and the global optimization in the relaxation phase, the running times are higher than, e.g. [PGK02] where only approximate error measures and a pure greedy approach is used. Nevertheless the timings are acceptable as a pre-processing step. By
applying another round of relaxation and redundant splat removal we can usually reduce the output complexity by another $1-2\%$ but this further increases the computation time.

Table 6.2 shows the timings and splat counts for the Igea model (cf. Fig. 6.10) with different error tolerances $\varepsilon$. The timings are measured separately for the three phases of the algorithm. As one expects, the computing time for the splat generation increases for larger error thresholds. This comes from the fact that while the number of candidate splats depends on the number of input samples, the size of these splats depends on the tolerance. Conversely, the time for the global relaxation increases with smaller error threshold since it depends on the number of active splats and not so much on their size. Adding all timings together, it turns out that the overall computing costs have a global minimum for some intermediate error tolerance (cf. Fig 6.8). If $\varepsilon$ becomes larger, the splat generation dominates the computing costs and if $\varepsilon$ becomes smaller, the relaxation will then dominate.

### Table 6.1: Running-times of the algorithm on various models. The ‘type’ stands for the splat shape: (C)ircular or (E)lliptical.

<table>
<thead>
<tr>
<th>model</th>
<th>#points</th>
<th>$\varepsilon(%)$</th>
<th>type</th>
<th>time(s)</th>
<th>#splats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charlemagne</td>
<td>598386</td>
<td>0.1</td>
<td>C</td>
<td>935</td>
<td>17445</td>
</tr>
<tr>
<td>Buddha</td>
<td>546730</td>
<td>0.1</td>
<td>C</td>
<td>592</td>
<td>13249</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>E</td>
<td>1048</td>
<td>9405</td>
</tr>
<tr>
<td>Dragon</td>
<td>437645</td>
<td>0.2</td>
<td>C</td>
<td>354</td>
<td>4450</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>E</td>
<td>1048</td>
<td>9405</td>
</tr>
<tr>
<td>Iphigenie</td>
<td>351750</td>
<td>0.05</td>
<td>C</td>
<td>663</td>
<td>30181</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>E</td>
<td>1207</td>
<td>20714</td>
</tr>
<tr>
<td>Max</td>
<td>199169</td>
<td>0.2</td>
<td>C</td>
<td>189</td>
<td>1271</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>E</td>
<td>394</td>
<td>698</td>
</tr>
<tr>
<td>Female-torso</td>
<td>171094</td>
<td>0.2</td>
<td>C</td>
<td>236</td>
<td>861</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>E</td>
<td>394</td>
<td>698</td>
</tr>
<tr>
<td>Igea</td>
<td>134345</td>
<td>0.2</td>
<td>C</td>
<td>116</td>
<td>1621</td>
</tr>
<tr>
<td>Horse</td>
<td>50697</td>
<td>0.3</td>
<td>C</td>
<td>27</td>
<td>1138</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>E</td>
<td>42</td>
<td>898</td>
</tr>
<tr>
<td>Bunny</td>
<td>34835</td>
<td>0.3</td>
<td>C</td>
<td>14</td>
<td>1371</td>
</tr>
<tr>
<td>Torus</td>
<td>30000</td>
<td>0.2</td>
<td>C</td>
<td>12</td>
<td>734</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>E</td>
<td>19</td>
<td>510</td>
</tr>
</tbody>
</table>
6.7 Results and Comparisons

<table>
<thead>
<tr>
<th>( \varepsilon ) (%)</th>
<th>times (s)</th>
<th># splats</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>create</td>
<td>greedy</td>
</tr>
<tr>
<td>0.8</td>
<td>352</td>
<td>73</td>
</tr>
<tr>
<td>0.4</td>
<td>149</td>
<td>38</td>
</tr>
<tr>
<td>0.2</td>
<td>72</td>
<td>12</td>
</tr>
<tr>
<td>0.1</td>
<td>34</td>
<td>7</td>
</tr>
<tr>
<td>0.05</td>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td>0.02</td>
<td>11</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 6.2: Timings and (circular) splat numbers after the greedy selection and global relaxation for the Igea model with decreasing error tolerance \( \varepsilon \).

Figure 6.8: Overall computing costs vs. error tolerance \( \varepsilon \) for the Igea model (cf. Table 6.2).

The rate by which the number of splats increases for decreasing error threshold \( \varepsilon \) in Table 6.2 indicates that our piecewise linear \( C^{-1} \) continuous surface reconstruction scheme in fact has linear precision and hence quadratic approximation order. This justifies that splat-based geometry representations and triangle meshes are equally powerful in approximating freeform geometry. Figures 6.1, 6.7, 6.9 and 6.10 demonstrate the visual quality of the splat-based representations. Whether the models are globally smooth or have many small features, our sub-sampling algorithm always produces splat-based approximations that have high visual fidelity, uniform sample density, no holes, and are guaranteed to stay within the prescribed error tolerance.
6 Optimized Splat Sub-Sampling

Figure 6.9: The Charlemagne model (left, 598K points) approximated with disk-shaped splats by setting the relative error threshold to $\varepsilon = 0.03\%$ (middle two, 66401 splats) and $\varepsilon = 0.1\%$ (right two, 17445 splats).

6.7.1 Elliptical vs. Circular Splats

We first compare the approximation power of two types of splats. Since a torus has a simple geometry with clear curvature changes, we use it as a benchmark object in Fig. 6.4. Our results on other less synthetic objects are similar but sometimes overlayed by effects caused by fine geometric detail. We found that for a given error tolerance $\varepsilon$, the required number of elliptical splats is usually about 30% less than the number of circular splats. Besides the additional computation time during the splat generation phase, elliptical splats also require slightly more per-pixel computation during rendering. However, as demonstrated in Fig. 6.10, the improved visual quality when using Phong shading [BSK04] definitely justifies the use of elliptical splats.

6.7.2 Comparing to Point Cloud Simplification

We compare the results of our algorithm to the greedy iterative point cloud simplification scheme proposed in [PGK02] using circular splats as [PGK02] only considers circular ones. By using the technique of [Paj03], we derive splat radii for [PGK02]’s output and then compute the $\|\cdot\|_\infty$ approximation error from the input point cloud to the output splats.

Fig. 6.11 shows the results of the greedy point cloud simplification and of our algorithm on the Torus model. With about three times the computation time, our algorithm produces a splat-based model with the same number of splats but with a significantly
6.7 Results and Comparisons

Figure 6.10: From left to right in top row are the sub-sampled Igea models with 218, 1621, 10737, and 29847 filtered circular splats and decreasing error $\varepsilon = 0.8\%$, 0.2\%, 0.05\%, 0.02\% respectively. The bottom row shows approximations with 10737 circular splats (left) and 8556 elliptical splats (middle two) at same error 0.05\% and their respective zoom-in views (right two). Notice the improved visual quality of elliptical splats (far right).

smaller approximation error (which comes from the superior splat distribution). The more uniform distribution of the splats in our case also equalizes the sizes of the splats and hence improves the visual quality. If we allow for an equally high error tolerance then our global relaxation scheme can reduce the splat count by one third. This indicates how far away from the optimal solution the greedy solution is.

6.7.3 Comparing to Mesh Simplification

We also compare our results to the well-established greedy mesh simplification algorithm [GH97]. Triangle meshes have the advantage over circular splats that they can adjust anisotropically to the local minimum and maximum curvature while disk-shaped splats can only adjust isotropically to the maximum curvature. On the other hand, splat representations are more flexible since they do not have to satisfy $C^0$ continuity conditions between the linear pieces. Our experiments indicate that the two advantages
6 Optimized Splat Sub-Sampling

![Figure 6.11: Same number of 734 circular splats generated by point cloud simplification [PGK02] (left) and by our algorithm (center) with respective running time 4.5 sec and 12 sec and approximation error 0.29% and 0.2%. The right figure shows our sub-sampling result at 0.29% error using only 493 splats created in 16 seconds.](image)

somewhat balance each other but the flexibility of splat-based representation almost always leads to better visual quality as well as to lower $\| \cdot \|_\infty$ approximation error when using the same number of primitives (cf. Fig. 6.12). Even more interestingly, elliptical splats seems to have both these two advantages.

6.8 Summary

Explicit splat-based models are another ideal geometric representations for distributed and mobile geometry processing systems, since, on the one hand, they have the same quadratic approximation power as the standard parametric triangle meshes, and on the other hand, without topological constraints, they are more flexible than triangle meshes. This flexibility is even important for distributed and mobile applications as this will make splat models robust against packet loss during the network transmission.

In this chapter we presented a sub-sampling scheme that converts a dense set of point samples into a much more compact and sparse set of circular or elliptical object-space splats that provides a hole-free approximation of the original data up to a prescribed error tolerance $\varepsilon$. The simplicity and flexibility of splat models are further exploited that our scheme achieves comparably low splat counts and uniform splat distribution by applying a global optimization scheme. The scheme is not as fast as pure greedy approaches but the increased computation costs are paid off by a considerably improved output quality.
Figure 6.12: Comparisons of the Torus shape approximations with about 730 geometric primitives: simplified (by [GH97]) irregular mesh (left), regular sampled mesh (center left), circular splats (center right) and elliptical splats (right). The approximation errors are 0.64%, 0.58%, 0.2% and 0.14% respectively. The models are rendered with Phong-shaded triangles and filtered Phong splats [BSK04].
7 Progressive Splatting

As we have mentioned in the previous chapter (Chapter 6), the sheer data size of explicit point-based models has increased so fast that they have brought great challenges for subsequent processing tasks like modeling, rendering and network transmission. This would be even more critical for low-end hardware environments like mobile phones or PDAs, which surely is one major motivation of our optimized splat sub-sampling method to reduce the complexity of these explicit geometric data structures.

On the other hand, although our optimized splat sub-sampling algorithm produces high quality results (cf. Chap. 6), the coarsified data structures are only single resolution, i.e., models of different resolution have to be generated and maintained separately. This is not flexible for some applications like progressive transmission and rendering where progressive or continuous level of detail hierarchies have proven to be very effective when it comes to locally adapt the resolution level of the 3D geometry model to the application-dependent quality requirements.

While for parametric polygonal meshes, their progressive structures have been well established as we already discussed in Section 3.5, less work has been done for the explicit representations, especially for the splat sets. Most previous work to generate progressive splat-based geometry has utilized the straightforward hierarchical space partitioning method or only focused on infinitesimal isolated points (centers of splats). They usually produce poor outputs at coarser approximation levels which could degrade the visual appearance of level of details rendering in the very beginning.

In this chapter, we will transfer the concept of the well-known progressive meshes [Hop96] from parametric triangle meshes to explicit splat models (cf. [WZK05]). Our progressive splat decimation procedure uses the standard greedy approach, but unlike previous work, it uses not only the splat centers, rather the full splat geometry (both center and its extent) in the decimation criteria and error estimates to improve the result quality.
With two modified error metrics generalized from [CSAD04], our new greedy framework offers much better quality than other progressive splat decimators (cf. Fig. 7.1). Therefore it is preferable for those progressive applications like rendering and transmission. The quality that we obtain is coming close to the our recently proposed single-resolution high-quality sub-sampling techniques which are based on global optimization while being much faster by an average factor of 3.

To further reduce the data size of the explicit splat models, we will also introduce here a compression scheme to the progressive splat structures based on geometric prediction and entropy coding.

### 7.1 Related Work

Considerable work has been presented during the last years relating to point-based graphics research [GPZ+04, KB04]. To construct multiresolution point-based models, most previous work has adopted the hierarchical space partitioning strategy aiming at efficient visualization of complex datasets. The input point set is clustered within a hierarchical space partitioning data structure and representative points or splats are created in each node by analyzing the local surface properties. Rusinkiewicz and Levoy [RL00] used bounding sphere hierarchies to perform recursive point rendering. Piecewise constant average points are computed for all spheres in the structure. This idea was followed in [DVS03] and the pre-computed hierarchy was re-arranged into a sequential format that can be efficiently processed by nowadays graphics hardware. Many other papers also used
7.1 Related Work

octrees as underlying hierarchical structures [BWK02, Paj03, KV03b]: while [KV03b] only concentrated on \(C^{-1}\) piecewise constant points, [BWK02, Paj03] can also generate multiresolution \(C^{-1}\) piecewise linear splat geometry. More recently [SPL04] converted the output of [Paj03] to multiple continuously stored harddisk files to render large splat samples in an out-of-core fashion. In general, hierarchical space partitioning techniques are straightforward and efficient. However, due to the lack of algorithmic flexibility, they are often not able to produce progressive models of sufficient quality especially in coarse approximation levels. This results in poor splat representations in the beginning phases of progressive splatting (cf. Fig. 7.2).

![Hierarchical space partitioning will lead to non-uniform splat sizes (left) or splats connecting two distant parts (right) in the coarse level due to the less flexible global partitioning. Splats are shown as thick lines and displaced for clear illustration, surfaces as dotted curves. Further subdivision (dashed lines, right) could improve these problems but 1-to-\(n\) partitioning prevents the granularly resolution control of progressive models.](image)

To achieve better quality, the well established greedy optimization schemes used in mesh simplification has been adapted by various works [Lin01, PGK02, FACOS03], to simplify point-sampled geometry. Their results can naturally lead to a progressive surface format for progressive splatting. Unfortunately, they all focused only on the relationships between splat centers and hence require extra effort to estimate the actual splat spatial extent. On the contrary, taking the full geometry of surface splats into account, we have presented an optimal splat sub-sampling algorithm [WK04] that performs a greedy phase followed by a post global relaxation optimization step. The output quality of this method is superior compared to most other similar schemes, which has inversely compromised its progressive capability due to its non-continuous single-resolution sub-sampling property.
7.2 Progressive Splat Decimation Algorithm

Targeting at progressive splatting, we introduce an iterative greedy splat decimation framework working directly on $C^{-1}$ piecewise linear surface splats. Intuitively, it functions as the counterpart of the well-known progressive meshes [Hop96] in the $C^0$ piecewise linear polygonal meshes setting and the iterative point simplification [PGK02] in the $C^{-1}$ piecewise constant points setting.

Specifically, given the input point set, *initial splats* are first created for all point samples. Then all possible *splat merge operators* are arranged in a priority queue according to an *error metric* measuring errors caused by respective operators with top element having minimum error. *Iterative operations* are usually performed repeatedly with applying the top operator and updating possibly affected operators in the queue until the desired number of splats is reached. More details will be discussed in the upcoming sections.

7.3 Initial Splat Creation

The input of our algorithm is a set of unstructured points $P = \{p_i\}$ sufficiently sampling and describing a smooth manifold boundary surface $S$ of a given 3D object. Each output splat $T_i$ is a general 3D ellipse given by its center $c_i$, its unit normal vector $n_i$, and two additional *non-unit* vectors $u_i$ and $v_i$ defining its major and minor axes.

Similar to [PGK02] and [WK04], in order to analyze local surface properties as well as the associated initial splat $T_i$ of a point sample $p_i$, the $k$-nearest neighbors $N_k(p_i)$ have to be computed beforehand. Then a least square plane $L$ can be found for $p_i$ and $N_k(p_i)$ defining the normal $n_i$ of $T_i$, with center $c_i = p_i$. As we set all initial splats to be circles, initial $u_i$ and $v_i$ can be any two orthogonal vectors parallel to $L$ with same length $r$,

$$r = \max \left\| (p_j - c_i) - n_i^T (p_j - c_i) n_i \right\|,$$

for all $p_j \in N_k(p_i)$.

In the meantime, a virtual graph $\mathcal{N} = (P, E)$, where the edge $(i, j)$ belongs to $E$ iff $p_j \in N_k(p_i)$, is also formed to represent the above neighborhood relationship. This graph $\mathcal{N}$ will be the supporting *dynamic* topology during the iterative splat merge
operations, i.e., as an extension of the common edge collapse operator \cite{GGK02}, a splat merge operator $\Phi$ will merge two splats $T_l$ and $T_r$ associated with endpoints $p_l$ and $p_r$ of an edge $e \in E$ into one larger splat $T_m$. Thus the ordering queue will initially contain splat merge operators corresponding to all edges in $E$. Once an operator $\Phi$ is applied, $e$ and other degenerated edges will be removed from the edge set $E$ and $p_l$ and $p_r$ will be replaced by a new point $p_m = c_m$ in the point set $P$.

7.4 Error Metrics and Splat Merge Operators

In order to ensure similar approximation quality as that of the established mesh cases, two different error metrics measuring distance deviation and normal deviation respectively are also generalized and embedded into the new splat decimation framework. As splat merge operators heavily depend on specific error metrics, we also introduce their detail information at same time in this section.

7.4.1 $L^2$ Metric

The $L^2$ error metric is based on Euclidean distance measurement. To be able to compute the deviation error caused by a splat merge operator $\Phi$ with respect to the original point set, an additional array of indices $\{f_i\}$ to the original points is kept for each splat $T_i$ and initialized with a single index $\{i\}$ referring to the initial point $p_i$. When merging two splats, their index arrays will be united and assigned to the new splat. Then for a merge operator $\Phi$, to merge splat $T_l$ and $T_r$ to new splat $T_m$, the approximation error is defined as:

$$
\varepsilon_{\Phi} = \|e\| \cdot \sum_{f \in \{f_m\}} |\text{dist}(p_f, T_m)|^2, \quad \{f_m\} = \{f_l\} + \{f_r\}.
$$

(7.1)

Note that the above error metric has been weighted by the length of the edge $e$ measuring the distance between two merged splat centers to penalize merging two distant splats which otherwise would produce over-sized splats.

Given the $L^2$ error metric (7.1) and two splats $T_l$ and $T_r$ to be merged, the new splat $T_m$ can be determined by applying Principle Component Analysis (PCA \cite{Jol02}) to the point set $P_m = \{p_f\}, f \in \{f_m\}$ in 3D directly rather than the projected point set in 2D as \cite{Paj03}. Afterwards, we will have the average point $\bar{p}$ as well as three real
7 Progressive Splatting

![Image of splat merge operators]

Figure 7.3: Splat merge operators according to $L^2$ error metric (left) and $L^{2,1}$ metric (right).

![Image of Igea model]

Figure 7.4: Igea model (134K points) coarsified to same number of 13K splats using $L^2$ error metric (left) and $L^{2,1}$ metric (right) with respective run time 52 sec. and 31 sec. and relative Hausdorff distance error 0.017% and 0.022%.

Eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$ and the corresponding eigenvectors $v_1$, $v_2$, $v_3$. Then for $T_m$, center $c_m = \bar{p}$, normal $n_m = v_3$, and two axes $u_m$ and $v_m$ will have direction $v_1$ and $v_2$ respectively with a length ratio $\sqrt{\lambda_1/\lambda_2}$. The final axis lengths are scaled simultaneously so that the elliptical splat $T_m$ covers all points $P_m$ in 2D when they are projected onto the splat plane (cf. Fig. 7.3).

Once a splat merge operator $\Phi$ is applied, the graph $\mathcal{N}$ will be updated as we have mentioned in Section 7.3. Similar to standard mesh simplification, all neighboring merge operators to $T_l$ and $T_r$ in the ordering queue have to be updated with new merge errors or removed from the queue if they are degenerated and not valid any more.
7.4.2 \(L^{2,1}\) Metric

\(L^{2,1}\) error metric measures the deviation of normal directions and is extended from the original metric first presented in [David04]. In this case, the error computation is simpler and we do not need to keep the index array either. Given the splat merge operator \(\Phi\), the respective area \(|T_l|\) and \(|T_r|\) of two splats \(T_l\) and \(T_r\) to be merged, similar to (7.1), the edge-length weighted error is calculated as:

\[
\varepsilon_{\Phi} = \|e\| \cdot (|T_l| + |T_r|) \cdot \|n_l - n_r\|^2.
\] (7.2)

According to the \(L^{2,1}\) metric, the geometry of new splat \(T_m\) is defined as center

\[
c_m = \frac{|T_l| \cdot c_l + |T_r| \cdot c_r}{|T_l| + |T_r|};
\]

and normal

\[
n_m = \frac{|T_l| \cdot n_l + |T_r| \cdot n_r}{|T_l| + |T_r|}.
\]

The extent of splat \(T_m\) is computed in the same way as for the \(L^2\) metric with the only difference that rather than projecting the point set \(P_m\) (which we do not keep), we uniformly sample \(n\) points (usually 8 is enough) on both boundaries of splat \(T_l\) and \(T_r\) and project all of them to the splat plane of \(T_m\) to find the main axis directions and proper scaling (cf. Fig. 7.3).

We have compared the effects of the different \(L^2\) and \(L^{2,1}\) error metrics in Fig. 7.4. It is not surprising that we have similar observations as [CSAD04]: \(L^{2,1}\) metric can better capture the anisotropy of the surface geometry, while \(L^2\) metric will generate a more uniform splat distribution. Due to its simple computational efforts, algorithms using \(L^{2,1}\) metric naturally run much faster than those using \(L^2\) metric. Because the error is measured as one-sided maximum Hausdorff distance from the original point set to coarser splat approximations (the percentage of the major bounding box diagonal length), it is also clear to see that this distance-based error measurement favors distance-based \(L^2\) metrics over normal-based \(L^{2,1}\) metrics.

7.5 Progressive Splat Structures

The sequence of splat merge operators \(\{\Phi_i\}\) can be recorded during the splat decimation procedure. Together with the remaining coarse base splat set \(T_B\), the splat merge op-
7 Progressive Splatting

Operators \( \{ \Phi_i \} \) and their straightforward inverses, splat split operators, form a progressive splats format in the similar way as progressive meshes [Hop96] with the major difference that progressive splats do not maintain topology information.

Looking into details, the progressive splats format is composed with a base splat set \( T_B \) and a set of continuous detail operators \( \{ \Psi_i \} \). Each single detail operator \( \Psi_i \) is derived from corresponding recorded splat merge operator \( \Phi_i \) and will contain three splat indices \( l, r, m \) and the geometry of three splats \( T_l, T_r, T_m \). The data storage is amount to 48 Bytes per operator under single floating point precision. For refinement, the splat of index \( m \) will be split and replaced with two smaller splats \( T_l \) and \( T_r \); For coarsification, two splats of respective indices \( l \) and \( r \) will be merged and replaced with a larger splat \( T_m \). Note that by utilizing this progressive splats format we can perform both up-streaming and down-streaming without any extra data storage, and thus can produce splat models at arbitrary resolution. This also will support efficient applications like progressive rendering and transmission of surface splats.

7.6 Compressed Progressive Splat Transmission

With above progressive splat data structures, we can easily create continuous level of detail hierarchies from any given point-set. While on the other side, we also find that these structures still have some redundancies in the storage scheme of progressive splatting files. In order to further reduce the data size and for efficient network transmission, we will introduce here a compression scheme for the progressive splat representations. Note that as splat sets do not have explicit topological information, we only have to compress their geometric information.

Various algorithms have been proposed for geometry compression, most of which follow a three-step procedure: pre-quantization of vertex positions, prediction of quantized positions, and entropy coding of prediction residuals [GGK02]. We will adapt this idea in the splatting setting and consider the compression of full geometry of surface splats.

In general, the structure of progressive splatting can be described as a forest in which trees are full but not necessarily balanced. Two child splats always lie within the parent splats in the tangential direction. Based on these observation we can have a reasonable prediction rule when we are trying to compress the progressive splat structures.
Applying this prediction rule to a splat’s geometric properties results in a nearly normal distribution of errors which is certainly good for compression. Scalar quantization is then applied to each geometric component, creating a series of symbols, which are finally compressed by entropy coding.

### 7.6.1 Structure Transformation

The purpose of the compression scheme is to compress the detail operators \( \{\Psi_i\} \) during the progressive splat transmission. Each detail operator will contain three elliptical splats \( T_l, T_r, T_m \) and every splat possesses 4 properties: position, normal, major-axis and minor-axis. Same as above, a splat can be denoted as a tuple of four vectors \( T = (p, n, u, v) \). As we can see, not all properties are independent, because the normal vector is the cross product of the two axes, \( n = u \times v \).

In order to de-correlate the coordinates of vectors \( p, n, u, v \), a transformation from global cartesian coordinates to local spherical coordinates can be applied. This transformation yields independent local coordinates, which distribute more densely around the original point. Usually, according to the distribution graph (cf. Sect. 7.6.2), the more densely a component distributes, the higher compression rate we can reach.

So instead of storing \( (p, u, v) \), a mixed tuple is stored

\[
T = (\mid p \mid, p_\theta, p_\phi, \mid n \mid, n_\theta, n_\phi, \mid u \mid, u_\theta, u_\phi, \mid v \mid).
\]

The spherical coordinates are computed as follows, e.g. for position \( p = (x, y, z)^T \),

\[
\mid p \mid = \sqrt{x^2 + y^2 + z^2}, p_\theta = \tan^{-1}(y/x), p_\phi = \tan^{-1}(z/\sqrt{x^2 + y^2}).
\]

It is clear to see that the number of components is kept the same. But as the probability distribution of \( n \) component is much denser than \( v \) component (cf. Fig. 7.6), in order to achieve a better compression rate, the newly proposed components are used for compression.

### 7.6.2 The Prediction Rule

Having the parent splat \( T_m \), the positions of child splats \( T_l \) and \( T_r \) can be predicted as \( p_l = p_m + u_m/2 \) and \( p_r = p_r - u_m/2 \), the normals can be predicted as \( n_l = n_m \) and \( n_r = n_m \). In addition, the major axes can be predicted as \( u_l = u_m/2 \) and \( u_r = u_m/2 \) and the length of minor axes can be predicted as \( \mid v_l \mid = 0.9|v| \), which empirically turns
out to be the best results from our experiments. In order to visualize the distribution of predict errors, we use histograms here. In each histogram, the horizontal axis denotes the value cells, the vertical axis is the variable frequencies that fall into each cell. The higher and narrower a peak we have in the histogram, the better the compression rate that we will reach. The respective histogram of prediction errors are shown in Figures from 7.5 to 7.7.

![Figure 7.5: Histograms of position errors (|p|, pθ, pϕ).](image)

![Figure 7.6: Histograms of normal errors (nθ, nϕ) (left two) and of v-axis errors (|v|) (right).](image)

### 7.6.3 Quantization

Quantization techniques can be classified into scalar/vector and uniform/non-uniform ones. Scalar quantization is simply a “rounding-off” process, while vector quantization tries to partition a high-dimensional space into a set of cells that approximate the actual values in cells. As already shown in [KSS00], scalar quantization will result in best error distortion curve, so scalar quantization is applied here.

Different quantization bits are tested and results using more than 7 bits quantization are acceptable (cf. Fig. 7.8). Since the precision of input data is $10^{-6}$ (restricted by the
7.6 Compressed Progressive Splat Transmission

**Figure 7.7:** Histograms of u-axis errors ($|u|$, $u_\theta$, $u_\phi$).

Input ascii data file), quantization with over 20 bits is surely meaningless. In practice, quantization more than 12 bits will result in almost the same visual appearance.

**Figure 7.8:** Comparison of different quantization bits (6bits, 8bits, original).

### 7.6.4 Entropy coding

Suppose the source can be considered as a first-order model, which means all symbols are independent from each other, but with different probabilities. Given this kind of source data, Shannon’s law [Com] says that the entropy can be calculated as following,

\[
H = \sum_{i=1}^{n} p_i \log_2 p_i \text{ bits/symbol},
\]

where $n$ is the number of different symbols and $p_i$ their respective appearance probabilities. The entropy coding approaches we are considering include Huffman coding, arithmetic coding and range coding.
Huffman coding \([\text{Huf}]\) can perform as the optimal entropy only if the number of symbols is a power of 2. This is because in Huffman coding, we can only assign integer number of bits for a symbol. In practice, in order to obtain a high encoding and decoding speed, many techniques can be employed. For example, canonical Huffman code is chosen, and lookup encoding and decoding are used to avoid tree construction.

Unlike Huffman coding, arithmetic coding \([\text{Ari}]\) can achieve the sub-optimal performance of entropy. Because in arithmetic coding, each symbol is assigned to decimal number of bits. Suppose we have a set of symbols and the probability of each symbol. The idea behind arithmetic coding is to have a probability line, 0-1, and assign to every symbol a range in this line based on its probability, the higher the probability, the larger range which assigns to it.

Range coding \([\text{Ran}]\) is an alternative of arithmetic coding which works based on an integer number instead of floating number, thus more suitable for implementation. Although range coding algorithm performs 1% worse than arithmetic coder, the speed is twice faster. We have tested both Huffman coder and range coder, while arithmetic coder is not chosen because of its low speed.

### 7.6.5 Experimental Results

We will show some brief experimental results in this section. Table 7.1 compares the compressed data file size of different quantization bits and the compression ratio. It also shows the efficiency of our compression scheme. The running times are summarized in Fig. 7.9 as well as the compression performance via PSNR curve.

**Figure 7.9:** (Left) Running time of different coders with respect to different splat numbers. (Right) PSNR to bpp curve of the Horse model.
7.7 Results and Comparisons

<table>
<thead>
<tr>
<th>Quantization bits</th>
<th>Compressed size (Huffman, bytes)</th>
<th>Compressed size (Arithmetic, bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>512000</td>
<td>508246</td>
</tr>
<tr>
<td>8</td>
<td>586868</td>
<td>584416</td>
</tr>
<tr>
<td>9</td>
<td>670760</td>
<td>668450</td>
</tr>
<tr>
<td>10</td>
<td>765732</td>
<td>763353</td>
</tr>
<tr>
<td>11</td>
<td>879304</td>
<td>877154</td>
</tr>
<tr>
<td>12</td>
<td>1029476</td>
<td>1027372</td>
</tr>
<tr>
<td>13</td>
<td>1252168</td>
<td>1250079</td>
</tr>
<tr>
<td>14</td>
<td>1620096</td>
<td>1618003</td>
</tr>
</tbody>
</table>

Table 7.1: Comparison of compressed file size with respect to different quantization bit numbers. The uncompressed file size is 2643468 bytes.

We also compare our results to other popular compression schemes. Compressed progressive meshes [PR00] declares that 11 bits/△ can be reached, which is approximately 22 bits/point. Progressive compression of point-sampled models [WGE+04] reports that they can compress the position component to 25 bits/point with a PSNR of 55 dB. Our compression scheme has a compression rate of 22 bits/point (position component) with a PSNR of 73 dB, i.e., with similar compression rate, our progressive compression introduces less errors.

If we compress a progressive splat data structure using gzip (or bzip), only a compression rate of 95% w.r.t. the original size can be reached. But with our algorithm, a compressed file with 10 quantization bits only needs 29% size of original file, while still having hardly detectable visual differences.

7.7 Results and Comparisons

We have tested our progressive splatting algorithm with various point-based datasets on a standard Linux PC with 2.8 GHz CPU and 2 GB main memory. If not otherwise specified, k-nearest neighbors is always set to k = 8 and the $L^{2.1}$ error metric is applied to produce the reported results. Shaded results to show the overall visual quality can
Figure 7.10: Progressive splatting of Dragon model (438K points) from left to right, top to bottom with 2K, 6K and 20K splats.

already be found in Fig. 7.1 and Fig. 7.10, where splats also shrink to demonstrate their shapes and distribution.

7.7.1 Comparison to Splat Decimation Methods

We conduct comparative analysis of our progressive splatting algorithm (denoted as PSP) with the other two typical progressive splat decimators, the LOD point rendering (LOD) [Paj03] and iterative point simplification (IPS) [PGK02], and the single-resolution optimal splat sub-sampling scheme (OSS) [WK04] respectively. For LOD, splats in the same Octree levels will be collected and for IPS, an extra step is necessary to convert its point-based output to the splat representation using a strategy similar to the one discussed in Section 7.3.

Quality The result quality of different methods is estimated in the following aspects:

- **Error measurement** captures the statistical distances between decimated splat approximations and the original point model. Fig. 7.11 (right) compares the three progressive splat decimators while in Fig. 7.12 errors caused by our progressive PSP algorithm and discrete OSS scheme are reported.
7.7 Results and Comparisons

<table>
<thead>
<tr>
<th>$n_{\text{splat}}$</th>
<th>PSP</th>
<th>LOD</th>
<th>IPS</th>
<th>OSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>415</td>
<td>2.18</td>
<td>2.34</td>
<td>4.14</td>
<td>1.63</td>
</tr>
<tr>
<td>2591</td>
<td>2.52</td>
<td>2.71</td>
<td>4.12</td>
<td>2.15</td>
</tr>
<tr>
<td>11588</td>
<td>3.23</td>
<td>3.22</td>
<td>4.23</td>
<td>3.13</td>
</tr>
</tbody>
</table>

Table 7.2: Area ratio for different splat decimation methods, normalize to the initial surface area of the triangle mesh. This factor measures how much overdraw occurs in the rasterization of the splats.

- **Visual quality** depends on the rendering effects as well as the splat shapes and distribution (see Fig. 7.12 and Fig. 7.14).

- **Area ratio** between the area sum of splat approximations and the mesh surface area of the original point model (see Tab. 7.2). With same number of splats, the smaller the ratio, the smaller the area of splats to be rasterized in the fragment shader of GPU, and the faster the rendering speed.

Considering of all above three criteria in combination, it is not difficult to tell that, among the three progressive splat decimators, our PSP algorithm always performs better than both LOD and IPS. Especially on coarser scales, we find that because LOD merely adopted octree space partitioning scheme and IPS only considered points, i.e. splat centers rather than whole splats (one reason for its large area ratio too), they could not produce as promising results as ours. In addition, although the single-resolution OSS usually produces best quality due to its global optimization, OSS can not create progressive splat representations and even in some aspects (e.g. error measurement) our PSP method comes quite close to the best OSS solution already.

**Timing** Computation times of different splat decimation methods are shown and compared in Fig. 7.11 as functions of input model size. No wonder that LOD runs fastest as it has a quite simple algorithmic structure. Although both using the same greedy framework, our PSP algorithm is slower than IPS which has adopted efficient quadric error metric (QEM) [GH97], as it has to compose and solve least-square systems in each splat merge step. And it is not a surprise that the best-quality OSS needs most running time because of its complex global optimization techniques. Nonetheless, since high computational costs have been traded with improved output quality, and since all
splat decimation schemes are pre-processing procedures, the amount of running time we have reported is always endurable.

Discussion In general, LOD and IPS run very fast while producing results of inferior quality. On the contrary, OSS can create output with best quality but is significantly slower. Our progressive PSP algorithm is intended to fill this gap: it runs only a little slower than LOD and IPS while much faster than OSS, and it offers result quality very close to the best OSS. In fact, what we have provided here is also a practical guide to choose different splat decimation strategy based on the specific quality-time tradeoff.

7.7.2 Comparison to Mesh Simplification

We also compare in Fig. 7.13 our PSP splat decimation algorithm to QEM mesh simplification method [GH97] that can produce progressive meshes [Hop96]. The two methods can produce quite similar results due to the same greedy piecewise linear approximation nature, but it is also worth to mention the lower approximation error of our results as splats are more flexible and can overlap with each other while triangles are rigidly connected in a $C^0$ fashion.
7.8 Summary

Multiresolution representations are always necessary to deal with large datasets for distributed and mobile processing systems since they can provide the unique progressiveness characteristic. For explicit geometric data, previous work on progressive splat decimation usually utilizes a hierarchical space partitioning strategy to create corresponding multiresolution formats, or adopts a greedy framework with the strategy of focusing only on the relationships between splat centers. As we have shown here, although running very fast, these two strategies can not produce plausible results at the coarse approximation level. This may be problematic in some cases, e.g. during progressive rendering from low resolution to higher one, much lower rendering quality will appear in the very beginning (cf. Fig. 7.14).

In this chapter we have presented a greedy progressive splat decimation algorithm to produce progressive explicit geometric data structures. Not just the splat centers, the full splat geometry has been utilized in the decimation criteria and error estimates. It offers much better quality than other progressive splat decimators and also comes quite close to the recently proposed globally optimized single-resolution sub-sampling techniques while runs much faster. In summary, our splat decimation algorithm can be a new candidate to generate progressive splat models with a good time-quality tradeoff.
Figure 7.13: Isis mesh (188K points) decimated by our PSP algorithm (left two) and QEM simplification (right two) to same 2K splats and vertices with respective run time 31 sec. and 28 sec. and relative error 0.51% and 0.63%.

In addition, we have also introduced a compression scheme for these progressive splat data structures to make them even more compact.
Figure 7.14: Visual comparisons of LOD (top row), IPS (top middle), our algorithm PSP (bottom middle) and OSS (bottom) where the bunny model (35K points) is approximated with same number of 415 (left), 2591 (middle) and 11588 (right) splats for LOD, IPS and PSP and 419, 2577 and 11564 splats for OSS respectively.
Part III

Implicit Representations
8 Piecewise Linear Distance Fields

An implicit geometric data structure defines a 3D surface as the kernel of a function $F : \mathbb{R}^3 \rightarrow \mathbb{R}$, i.e., as the set of points $(x, y, z)$ such that $F(x, y, z) = 0$. Comparing to previous discussed parametric or explicit surface representations, the implicit surface definition simplifies geometric queries such as inside/outside tests to mere function evaluations. Although there are many different choices for the function $F$ to represent a given surface, the most common one is the signed distance field which assigns to every point in space its Euclidean distance to the surface.

It is clear to see that a precise signed distance field may require huge amount of data to sufficiently sample the whole 3D space, which is certainly unrealistic for distributed and mobile geometry processing systems with frequent data exchanges through the network.

In this chapter, we will present an algorithm to compute a piecewise linear, not necessarily continuous approximation of the signed distance field for a given object (cf. [WK03a]). As the approximation is highly adaptive, the proposed piecewise linear distance fields are also highly demanded for distributed and mobile geometry processing applications as they are proven to be compact and progressive implicit geometric data structures (cf. Fig. 8.1).

Our approach is based on an adaptive hierarchical space partition that stores a linear distance function in every leaf node. We provide positive and negative criteria for selecting the splitting planes. Consequently the algorithm adapts the leaf cells of the space partition to the geometric shape of the underlying model better than previous methods. This results in a hierarchical representation with comparably low memory consumption and which allows for fast evaluation of the distance field function.
Figure 8.1: For a given polygon mesh surface (a) we compute a binary space partition (BSP) tree and (b) assign a linear distance function to every leaf cell. Figure (c) shows the discontinuous zero-set of the resulting piecewise linear scalar field and (d) shows another zero-set with a smaller approximation error and hence a finer space partition. Our goal in this paper is, for a given error tolerance, to choose the BSP splitting planes in an effective way.

8.1 Background and Motivation

The initial motivation of this work is that the design of efficient algorithms always has to be accompanied by the design of suitable data representations. This seems to be even more crucial for distributed and mobile processing tasks where more types of geometric structures have to be dealt with in the same time. To this end, e.g., we consider scenarios where a geometry processing algorithm is equipped with an additional representation of the signed distance function to obtain direct access to implicit shape information. Typical applications are, e.g., mesh decimation or mesh fairing where one frequently has to check the current deviation from the originally given surface. Here the error estimates can be found by evaluating the implicit representation while the actual processing is performed on an explicit representation. More examples can be found for mesh repairing with volumetric techniques [NT03, Ju04].

In this setup it is easy to identify the major requirements that an implicit representation has to satisfy.
• **Approximation power**: For freeform shapes the implicit representation usually provides only an approximation of the original surface. Hence an explicit to implicit conversion algorithm has to take the maximum error tolerance as an input parameter. The approximation power of a representation measures the rate by which the memory consumption increases if the tolerance is decreased.

• **Adaptivity**: To minimize the memory requirements one is also interested in a representation that adapts to the shape of the underlying object. In its simplest form, adaptivity of an implicit representation means that high accuracy is guaranteed only in the vicinity of the surface itself. Far away from the surface the approximation of the signed distance function does not have to be as precise. More subtle adaptivity further takes the fact into account that flat surface regions can be approximated much easier than highly curved regions.

• **Efficiency**: A critical property is the complexity of the evaluation algorithm. The ideal situation would be constant complexity but this usually contradicts the goal of adaptivity. By using hierarchical representations it is at least possible to bound the complexity by the logarithm of the precision tolerance.

In the following we are proposing an highly compact implicit representation that is able to approximate the signed distance function of a given surface up to any prescribed precision (cf. Fig. 8.1). As input format, we are focusing on polygon meshes but in principle it would be easy to generalize the construction to arbitrary input data. The major features of our representation are that ...

• ... we use a piecewise linear instead of a piecewise tri-linear representation since both have the same approximation power but linear functions need less coefficients and are faster to evaluate.

• ... we are not requiring the approximate distance field to be a continuous function. Again, this does not have any effect on the approximation power but it decouples neighboring cells since no $C^0$ boundary conditions have to be taken into account. In particular we do not need any balancing mechanism that propagates local refinement operations into neighboring cells to bound the level jump in a hierarchical representation.
8 Piecewise Linear Distance Fields

- ... we use a general BSP tree which means that the splitting planes that define the partition cells do not have to be axis aligned. As a consequence the partition can adapt even better to the surface since, from a certain refinement level on, most splits will cut the surface in normal direction rather than in tangential direction. We derive geometric heuristics that promote good splits and prevent bad ones.

8.2 Previous Implicit Data Structures

We are focusing on representations that approximate the signed distance field by a trivariate piecewise polynomial function defined over a space partition. The most basic data structure till now is a uniform 3D grid \( [s_{ijk}] \) that stores a scalar value at each grid node \((i, j, k)\). This discrete data is interpolated into the voxel cells by tri-linear functions \([LC87]\).

Since for signed distance fields the accuracy in the vicinity of the (zero-) surface is usually more important than further away from the surface, the uniform grid is not memory efficient because the spatial scalar field is sampled with the same rate everywhere and this makes it difficult to deal with high-resolution/precision objects. This redundancy can be avoided by adapting the sampling density to the distance from the surface \([WG92]\). The three colors “white”, “black”, and “gray” are assigned to cells that are completely inside, completely outside, or intersect the surface, respectively. Then starting from some initial cell decomposition, the gray cells are recursively refined until a prescribed error tolerance is met. If we stick to cubical voxel cells then this approach leads to adaptively refined octrees, the so-called 3-color-octrees \([Sam90]\).

To further reduce the redundancy, we can restrict the local refinement to those cells for which the tri-linear interplant deviates more than the prescribed tolerance from the actual distance field. This restriction leads to even better adaptivity since extreme refinement is only necessary in the vicinity of the highly curved surface regions \([FPRJ00]\).

Besides these variations of voxel-based piecewise tri-linear distance field representations there are alternatives like the permission grid \([ZG02]\) where piecewise constant functions are used or wavelet-based representations with higher order basis functions \([NS01]\). While the approximation power of piecewise constant functions is not sufficient for practical precision requirements, higher order functions provide good approximation. How-
Figure 8.2: The piecewise approximation of the signed distance field to the contour shown in (c). The 3-color quadtree in (a) uses 102040 cells for a given approximation tolerance $\varepsilon$ and produces a $C^0$ piecewise bi-linear distance field. In (b) the same approximation error can be obtained by using a piecewise linear $C^{-1}$ approximation. In this case the approximation can adapt to the local curvature such that 895 cells are sufficient. In (d) we show a BSP-tree decomposition. By using a linear distance function for every cell we obtain the same approximation error with only 254 cells. For the selection of the splitting planes we used the medial axis information as brown segments shown in (c).

ever, since the corresponding basis functions span over several neighboring voxels, the inter-dependencies between these voxels make it necessary that a whole neighborhood of cells has to be refined to guarantee consistency. This is called balancing and in practice it usually causes the generation of many redundant cells.

8.3 Piecewise Linear Signed Distance Fields

We compute approximative piecewise linear signed distance fields as our unique implicit geometric data structures. The major requirements mentioned in the motivation also has been taken into account.

First of all we want to achieve sufficient approximation power. In practice, piecewise tri-linear functions have been used successfully. We however prefer to use piecewise linear functions instead. These have the same approximation order but need less coefficients and consequently they are faster to evaluate and need less memory. In 2D, continuous piecewise linear functions have already been applied successfully to approximate the distance functions of curves [TR96, LDM01]. A linear function in 3D is given by a
normal vector $\mathbf{n}$ and an offset value $d$ (vs. 8 scalar values for a tri-linear function). In our implementation we quantize the possible orientations of the normal vectors to 16 bit by using the refined-octahedron-indexing scheme of [BWK02] and the offset $d$ as another 2 Bytes.

In order to avoid the balancing effect that causes a local refinement operation to spread over neighboring cells, we do not enforce any continuity of the distance function between neighboring cells which also differs from the previous 2D applications [TR96, LDM01]. Using $C^{-1}$ continuous functions instead of $C^0$ does not affect the asymptotic approximation power but the additional degrees of freedom usually lead to a space partition with significantly fewer cells for the same approximation tolerance.

Trying to most flexibly adapt the size, shape and orientation of the cells in the space partition, we are using a general BSP-tree data structure. This allows us to use both, the location and the orientation of the splitting planes for the adaptation. In Figure 8.2 we show a 2D example and demonstrate the effects of the various improvements.

### 8.4 Generation Steps

Given a polygonal mesh $\mathcal{M}$ as input surface plus an error tolerance $\varepsilon$, our goal is to generate an nearly optimized piecewise linear approximation of the corresponding signed distance field. For this we have to choose the BSP splitting planes in an effective way such that the approximation errors within each cell stays under the prescribed error bound while using as few cells as possible. The recursive algorithm is given by the pseudo-code in Fig. 8.3. Initially we call the procedure with an empty root node $T$ and the complete mesh $\mathcal{M}$ as input parameters.

In the generation algorithm, there are two major sub-procedures: (1) how to approximate the distance function to a mesh $\mathcal{M}$ by a linear function $f$ with minimum error and (2) how to select a proper BSP splitting plane that generates an optimal space/shape partition. These issues will be addressed in the following sections.
8.4 Generation Steps

CREATE (tree T, mesh M):

Find a linear function $f$ that approximates the distance function to $M$ and estimate the approximation error $\delta$

if ($\delta > \varepsilon$)

    select a splitting plane $P$;
    split $M$ into $M_{\text{left}}$ and $M_{\text{right}}$ by $P$;
    $T.plane = P$;
    CREATE (T.left, $M_{\text{left}}$);
    CREATE (T.right, $M_{\text{right}}$);

Figure 8.3: Pseudo-code for the linear distance fields generation algorithm.

8.4.1 Least Squares Linear Approximation

For the mesh $\mathcal{M}$ that lies within some cell of the BSP tree, we need to compute a linear function $f$ that approximates $\mathcal{M}$’s distance function. Here we use the standard least squares approach to the point set $\{P_i, i = 1 \ldots n\}$ formed by all vertices in $\mathcal{M}$. The center of gravity is $\bar{P}$ and the point variance set is $\{Q_i = (x_i, y_i, z_i)^T = P_i - \bar{P}, i = 1 \ldots n\}$. Then from the covariance matrix of $\{Q_i\}$ we compute three real eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$ and we denote the corresponding eigenvectors by $V_1$, $V_2$, $V_3$ respectively.

The normal direction $N$ of the optimal approximation plane in the least squares sense is then given by $V_3$ (direction of minimal variance). To derive the optimal offset value $d$ we compute $d_i = N^T P_i$ for all mesh vertices and set $d = (\max\{d_i\} + \min\{d_i\})/2$. Now the function $f(X) = N^T X - d$ approximates the signed distance function up to a factor of $\pm 1$. This factor can be determined by comparing $N$ to the average normal direction across the mesh patch $\mathcal{M}$.

Moreover we immediately find an estimate for the approximation error which is $\delta = (\max\{d_i\} - \min\{d_i\})/2$ where we exploit the fact that triangles lie within the convex hull of their corner vertices. Notice, however, that this estimate is only valid over that part of the plane $N^T X = d$ that is covered by the orthogonal projection of the mesh $\mathcal{M}$. In most of the practical configurations this region will cover the whole cell for which
the distance field approximation is computed. In some exceptional cases it can happen
that the cell extends beyond this region which makes further refinement necessary. This
is explained in detail in section 8.4.3.

8.4.2 BSP Splitting Plane Selection

We present three different strategies for the selection of the splitting planes. The first
strategy uses a simple and fast heuristic. In the second strategy we use the medial axis
of the given surface to find splitting planes that lead to an improved segmentation of
the surface and in the third strategy we additionally disallow splitting planes that lead
to problematic configurations where the mesh falls into several components.

Strategy A:

A natural candidate for the splitting plane can be derived from the above least square
approximation. The eigenvector \( V_1 \) indicates the direction of maximum variance and
yields the (statistically most efficient) splitting plane’s normal vector \( N_P \). The offset
value \( d_P \) is adjusted such that some balancing criterion is met, e.g., the resulting sub-
meshes \( M_{\text{left}} \) and \( M_{\text{right}} \) have the same number of vertices or the same surface area, or
we set \( d_P \) such that the splitting plane contains the center point \( \bar{P} \).

Strategy B:

The least squares approach computes the statistical distribution of the mesh vertices
in \( \mathbb{R}^3 \) but it does not take the geometric shape of the mesh into account. As in Fig. 8.4
it is a good heuristic to segment the mesh along curvature maxima since this produces
sub-patches \( M_{\text{left}} \) and \( M_{\text{right}} \) with maximum expected flatness. It can be detected
by looking at the medial axis (MA) [Blu67, ABK98] of the surface: for each curvature
maximum there is a branch of the medial axis that points to it.

Computing the exact medial axis is difficult and unstable [ACK01, GK04], hence
we use the sphere growing method [BK02] to compute discrete sample points on the
medial axis. Before recursive BSP generation, we store for each mesh vertex \( V_i \) the tuple
\((C_i, w_i, b_i, j)\), where \( C_i \), representing a MA sample, is the center of the smaller sphere
among two possible maximum spheres touching \( V_i \); \( j \) is the index of another vertex
touching that sphere; \( w_i \) is the sum the triangle areas adjacent to \( V_i \) to compensate for
varying vertex density in the input mesh $\mathcal{M}$; $b_i$ is a flag indicating if this MA sample lies inside or outside the object.

During the recursive procedure, the new splitting plane candidate is the least squares plane fitting to the MA samples which is most likely to split the mesh in a highly curved region. To reduce the disturbing effects that occur when the MA falls into several components (inside and outside the surface) we classify the samples by flags $b_i$ and only use those ones that belong to the majority. Even if it may happen that the medial axis has several inside (outside) components this heuristic leads to sufficiently reliable results once the partition is fine enough such that only one major curvature maximum lies within a cell.

A weighted least squares fitting is used to compute the splitting plane by MA samples, i.e., we start by computing the weighted average:

$$
\bar{C} = \frac{\sum_i w_i C_i}{\sum_i w_i}
$$

and then define the weighted variance set as $\{Q_i = w_i (C_i - \bar{C})\}$ to form a covariance matrix. The resulting least squares plane puts more emphasis on MA samples which belong to mesh vertices in sparsely sampled regions and less emphasis on samples from denser regions. The weight coefficient in fact guarantee a constant impact per surface area. The improvement of the approximation due to the MA based splitting plane selection strategy is demonstrated in Fig. 8.7. If the ratio of largest eigenvalue of the weighted covariance matrix to the smallest one falls below some threshold (25 in our
implementation), we consider the medial axis information not as reliable and simply fall back to selection strategy A.

Once the splitting plane is found, we separate the given mesh $\mathcal{M}$ into two sub-mesh parts together with the MA samples. In order to guarantee that they are not influencing each other after the split, we remove the subset of MA samples $(C_i, w_i, b_i, j)$ where the two vertices $V_i$ and $V_j$ belong different sub-meshes by setting the weight coefficients $w_i$ of the corresponding samples to zero.

**Strategy C:**

Selection strategy B promotes good splits that separate the mesh into sub-meshes with maximum expected flatness. In the third strategy we add another criterion that prevents bad splits. Here a bad split is characterized by the property that one of the sub-meshes $\mathcal{M}_{left}$ and $\mathcal{M}_{right}$ falls into several components. Those splits are disadvantageous since the least squares fitting described in section 8.4.1 assumes that the mesh has just one connected component. In fact the resulting linear approximation of the signed distance function is only valid over the regions that are covered by the projection of the corresponding mesh. If the mesh has several components, the distance estimate is invalid in between the projections of these components (cf. Fig. 8.5). In order to find a simple criterion to detect bad splits, we observe that such splits typically cut off a cap from an apical point of the surface. Hence we have to identify all potentially bad splits and disallow them in the recursive splitting procedure.

We extend the elegant 2D Hough transform [Bal81] into 3D. Given a plane $P(x, y, z) = x \cos(\theta) \cos(\phi) + y \sin(\theta) \cos(\phi) + z \sin(\phi) - c = 0$ in Euclidean space, we can transform it into Hough space where it is represented by a single point with coordinates $(\theta, \phi, c)$. Hence, a collection of planes in Euclidean space corresponds to a collection of points in Hough space.

Let $P$ be a vertex of the mesh surface $M$ and $N$ its normal vector. According to the above discussion we want to disallow all splits that cut through the $\varepsilon$-vicinity of $P$ in almost tangential direction where $\varepsilon$ is the prescribed error tolerance. First we compute the points $P^+$ and $P^-$ by shifting $P$ in positive and negative normal direction by $\varepsilon$.

All almost tangential planes are characterized by the fact that their first two Hough coordinates $(\theta, \phi)$ are within some interval

$$[\theta_i - \nu, \theta_i + \nu] \times [\phi_i - \nu, \phi_i + \nu]$$

(8.1)
Figure 8.5: Left: the splitting plane (thin line P1) chops off a ‘cap’ in the low-right subspace leaf cell enclosed by P1 and P3 whose linear approximation (thick line a1) is not proper although its approximation error is within the tolerance; Right: After the translation of that splitting plane to the parallel one (thin line P0), the ‘cap’ can be avoided.

where \((\theta_i, \phi_i)\) are defined by the normal vector \(N\) and \(\nu\) is some directional tolerance set by the user. In our implementation we usually set this tolerance to \(\nu = \pi/12\). If we fix a certain normal vector and consider all parallel planes while we are moving from \(P^-\) to \(P^+\) then the third coordinate \(c\) varies from \(c^-(\theta, \phi)\) to \(c^+(\theta, \phi)\). As a consequence we can describe the set of potentially bad splitting planes in the vicinity of the vertex \(P\) by the Hough volume enclosed between the surfaces \(c^-(\theta, \phi)\) and \(c^+(\theta, \phi)\) as \((\theta, \phi)\) varies over the interval (8.1). The union of all these volumes for every vertex \(P\) of the mesh \(M\) defines the set of bad splitting planes that should be avoided in the recursive procedure (cf. Fig. 8.6, left).

We discretize the Hough space and then scan convert the above volume. For the discretization we do not need a very high resolution in \(\theta\) and \(\phi\) direction. Since \(\theta\) runs over the interval \([0, 2\pi]\) while \(\phi\) only runs over the interval \([0, \pi/2]\) we usually set the corresponding resolution to \(200 \times 50\). The resolution in \(c\) direction is more critical since if it is chosen too low then the rasterization of the Hough volumes will leave hardly any empty cells. Empty cells, however, are used later to indicate that a certain splitting plane is not forbidden. In our implementation we use a resolution of \(500\).

During the recursive procedure, given a certain splitting plane, we first check if the corresponding raster cell in the discretized Hough map is empty. If not we cannot use the selected plane, we have to look for an alternative that is as close as possible. We
do this by searching in the Hough map in the $c$ direction for nearest empty cell. In Euclidean space this corresponds to moving the splitting plane in parallel in the normal direction. In case we cannot find an empty cell for the current $(\theta, \phi)$ coordinates, we simply fall back to strategy B.

Note that for every raster cell of the Hough map, we only need 1 bit to indicate if it is occupied or not. Given the above resolution for the 3D Hough map, we only need 610KB which does not cause significant memory overhead for the algorithm.

### 8.4.3 Final Linear Approximation Correction

In Section 8.4.1 we found that the linear approximation in each cell is only valid over the projection of the corresponding mesh patch. In some rare cases, it may happen that a cell is unnecessarily extended such that this validity condition is not satisfied (cf. Fig. 8.6, right). In this case some extra cell refinement is necessary to prune the linear approximation properly. This is done as a post process after the recursive BSP generation.

For every leaf node we check the validity of the linear approximation by projecting the vertices of the corresponding mesh patch into the zero-plane. We denote these projected points by $\{R_i\}$ and let $C$ be their center of gravity. We also compute the zero-plane
8.4 Generation Steps

polygon by intersecting the zero-plane with all splitting planes that define the current cell. Let \( \{G_i\} \) be the corners of this polygon. For every corner \( G_i \) we find the nearest point \( P_c \) in \( \{R_i\} \) in the direction of \( N_i = C - G_i \) and compute their distance \( d \). If it is larger than the prescribed tolerance \( \varepsilon \), we split the current cell along a plane that is defined by \( P_c \) and the normal vector \( N_i \) such that all mesh vertices are lying on one side with a minimum distance of \( \varepsilon \). This generates an empty cell but this cell is guaranteed to lie outside the error bound \( \varepsilon \) as well as a non-empty cell (cf. Fig. 8.6, right). This procedure will be repeated in that non-empty cell until the distance validity is guaranteed.

8.4.4 Discussions

The availability of several splitting plane selection strategies allows the user to adjust the trade-off between processing speed and approximation quality. Figure 8.7, and Table 8.1 show examples of how effective the medial axis (MA) heuristics is in reducing the number of cells in the space partition and how much computing time is spent to achieve this. Also these initial computation effort will be amortized since once generated, the distance fields can be used in many applications.

Figure 8.7: The zero-sets of the piecewise linear distance fields to the Fan model generated without (left) and with (right) MA information (strategy B).
Table 8.1: The performance of two piecewise linear distance fields with same maximum error tolerance 0.03. The original model has 101,136 triangles and a bounding box diagonal length 7.6.

8.5 Results and Applications

All the experiments have been performed on a commodity PC with P4 2.8 GHz CPU. If not specified otherwise, error tolerances are given as a percentage of the model’s bounding box main diagonal length.

The performance of our algorithm is summarized in Table 8.2. The major parts are used to compute the sample points on the medial axis and to create the discretized Hough maps. Some typical zero-sets of the linear distance field approximations are shown in Fig. 8.1, 8.7 and 8.10.

Table 8.3 and Fig. 8.10 show various levels of piecewise linear distance fields to the Max model which are computed for different error tolerances. The storage of each node has been calculated as follows: inner BSP nodes need 14 Bytes which stores a splitting plane in 4 Bytes (normals and offsets quantized both in 2 Bytes), two children pointers in 8 Bytes and another 2 Bytes tagging the leaf status of its two children; A leaf node with linear function needs 4 Bytes to store that approximation plane. The average depth indicates the average distance evaluation time and it is computed for all leaf cells that are not empty. For a leaf cell $i$ with a depth $d_i$, we clip the linear function against the cell boundary and compute the area $A_i$ of the resulting convex polygon. Then the average depth of a BSP tree is depicted as $\frac{\Sigma(A_id_i)}{\Sigma A_i}$ and this depth value corresponds to the expected path length that a distance query traverses to find the proper leaf cell.

The above results show that our piecewise linear distance field approximation is compact and adapts very well to the shape of the underlying surface. We also have exploited this representation in various applications by providing fast access to the distance estimates for a given model: Error bounded mesh processing can be easily and efficiently
8.5 Results and Applications

<table>
<thead>
<tr>
<th>model</th>
<th>faces (input)</th>
<th>ε (%)</th>
<th>time (min.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>A+B</td>
</tr>
<tr>
<td>Bunny</td>
<td>69,656</td>
<td>0.1</td>
<td>1.6</td>
</tr>
<tr>
<td>Bust</td>
<td>61,388</td>
<td>0.06</td>
<td>1.4</td>
</tr>
<tr>
<td>Fan</td>
<td>101,136</td>
<td>0.4</td>
<td>4.0</td>
</tr>
<tr>
<td>Horse</td>
<td>96,962</td>
<td>0.2</td>
<td>3.0</td>
</tr>
<tr>
<td>Max</td>
<td>99,999</td>
<td>0.5</td>
<td>3.2</td>
</tr>
</tbody>
</table>

**Table 8.2:** The running-times of the algorithm on diverse models with different splitting selection strategies.

<table>
<thead>
<tr>
<th>ε (%)</th>
<th>averg. ε (%)</th>
<th>max. depth</th>
<th>averg. depth</th>
<th>inner cells</th>
<th>linear func.</th>
<th>storage (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.6</td>
<td>14</td>
<td>8.7</td>
<td>263</td>
<td>228</td>
<td>4.5</td>
</tr>
<tr>
<td>0.5</td>
<td>0.3</td>
<td>16</td>
<td>9.9</td>
<td>601</td>
<td>507</td>
<td>10.2</td>
</tr>
<tr>
<td>0.25</td>
<td>0.15</td>
<td>24</td>
<td>11.3</td>
<td>1545</td>
<td>1222</td>
<td>25.9</td>
</tr>
<tr>
<td>0.06</td>
<td>0.04</td>
<td>24</td>
<td>13.6</td>
<td>6468</td>
<td>6171</td>
<td>112.5</td>
</tr>
<tr>
<td>0.03</td>
<td>0.02</td>
<td>24</td>
<td>14.7</td>
<td>13908</td>
<td>13617</td>
<td>243.3</td>
</tr>
</tbody>
</table>

**Table 8.3:** The statistics of piecewise linear distance fields to the Max model when decreasing the error tolerance.

implemented. For decimation (cf. Fig. 8.8), we have similar results as the latest error-bounded method, Permission Grids [ZG02], while their grids need over 100 times more space than our compact representation. Global error control is achieved by checking every atomic operator in the pre-computed approximation. It is natural to extend this to error bounded mesh smoothing. **Surface extraction** using [LC87, KBSS01] can be applied to our representation by evaluating the distance function on a uniform grid (cf. Fig. 8.10). **Level of detail** models is a straightforward extension of our hierarchical representation (cf. Fig. 8.10) and volumetric **CSG operators** (cf. Fig. 8.9) can also be applied.
Figure 8.8: The Bunny decimated by a QEM [GH97] based algorithm with our distance field approximation to guarantee a global error bound of 1% (left, 700 triangles, 7.1 sec.) and the standard QEM algorithm without error control (right, 700 triangles, 2 sec.). The corresponding Hausdorff errors are 0.93% and 1.5% respectively. [ZG02] simplified to 687 triangles with similar running time.

8.6 Summary

Aiming at compact and progressive implicit geometric data structures for distributed and mobile applications, in this chapter, we have proposed a new construction algorithm for the approximation of the signed distance field of a 3D geometric model with piecewise linear functions defined over a binary space partitioning.

For this purpose, we also explained several splitting plane selection strategies that promote good splits and prevent bad ones. The resulting structures require only small data space to reach reasonable approximation quality and their hierarchical format makes them flexible for progressive applications. We have shown the efficiency of this generic geometry representation with various results and some beneficial geometry processing applications.
Figure 8.9: The mesh (39673 triangles) is extracted from the union of two piecewise linear distance fields to cat model and torus model.
Figure 8.10: In the top row, from left to right are the zero-sets of the piecewise linear distance fields to Max model with error tolerance 1%, 0.25% and 0.03%. The bottom row are meshes extracted from the corresponding distance fields with 4004, 22324, 90092 triangles respectively (cf. Table 8.3).
Recovering the inherent structure and finding a faithful approximation of a given surface’s geometry have always been challenging tasks in computer graphics research. Optimal surface structures can lead to geometry data structures with best quality/data size tradeoff which of course will benefit distributed and mobile processing applications. At least, compact and efficient structures are always welcome.

On the other hand, as we can see in previous parts, parametric or explicit geometry representations, e.g., the standard triangle meshes or splat sets, are usually a set of irregularly connected geometric primitives. Each single primitive can be planar triangles or ellipses which are too simple to tell the underlying structures of the surface geometry.

Moreover, due to the steadily increasing availability and complexity of geometric models, better understanding of the underlying surface structures and characteristic geometric features becomes more and more critical nowadays. Accurately recovered surface structures will provide best blueprints for most geometry processing tasks which require optimal surface approximations, like multiresolution modeling, optimal domains for subdivision modeling, remeshing, high quality LODs, progressive compression, hybrid shape representations, etc. In addition, traditional CAD applications including reverse engineering and rapid prototyping can benefit from such representations as well.

In this chapter, we will present an automatic and robust surface structure recovery algorithm (cf. Fig. 9.1) which can provide highly compact implicit information of the input parametric surface geometry and reveal its inherent characteristics (cf. [WK05b]). It extends the powerful optimization technique of variational surface approximation [CSAD04]. In addition to planes, we allow for other higher order implicit surface elements like spheres, cylinders and more complex rolling-ball blend patches to embrace a new hybrid variational framework.
9 Robust Structure Recovery

Figure 9.1: The face model (16K faces, left) approximated by 5 spherical (middle) and 5 planar (right) proxies. Our hybrid scheme can more faithfully separate the distinct face parts like the forehead, the eyes, the cheeks and the nose.

By this, surface structures represented with only a small set of high-order implicit primitives can be recovered very elegantly and more robustly than with typical reverse engineering processes [VMC97], especially for mechanical CAD models which inherently consist of clear geometry structures. Moreover, as a side product of the improved structure recovery, our hybrid approximation method typically provides a higher approximation quality compared to the standard variational approximation scheme in particular when a very coarse segmentation is utilized.

9.1 Related Work

Many approaches have been proposed trying to find the optimal structure to approximate a given surface’s geometry, with certain constraints either on the approximation tolerance or the target complexity. Unfortunately, this optimization problem has been proven to be NP-hard [AS94], which is also the reason why most approaches employ heuristics based on local geometry characteristics.

There are quite a lot well-studied mesh simplification and decimation methods, some of which address the above approximation problem by incrementally performing atomic decimation operators like vertex removal or (half)edge collapse according to a certain priority ordering [Hop96, GH97, Gar99, GGK02, LRC+03]. The resulting coarse mesh already reveals in some sense the structure of the original dense mesh and can func-
tion, e.g., as the base mesh of a multiresolution surface representation. An alternative approach are the ‘dual’ methods to mesh simplification, i.e. the face clustering approximations [KT96, She01, GWH01, SSGH01, SWG+03, MK05], where progressive region merging operators are applied in the same greedy way to create a set of face clusters for geometric segmentation.

Surface remeshing techniques are another family of surface approximation approaches that can adapt to the underlying surface structures. They commonly re-sample the geometry carefully with well-shaped surface elements (often triangles or quads) in either isotropic or anisotropic fashion [KVLS99, GVSS00, BK01, AMD02, SAG03, ACSD+03, MK04]. Although delicate information like sharp features or curvature lines can be used to guide the remeshing process, the approximation quality often can not be guaranteed especially for very coarse scale approximations. In addition, a global minimization of some specific error metric can hardly be achieved in this kind of schemes.

Recently, the powerful optimization technique of variational shape approximation was proposed in [CSAD04]. The approximation task is casted as a discrete, variational geometric partitioning problem which is driven by minimizing the global approximation error. As we will extend this scheme, a more detailed description will be given in Section 9.2. The variational framework is simple yet effective and elicits a provable good trade-off between conciseness and geometric distortion. However, its structure recovery power and the approximation capability are somewhat limited by the restriction to planar elements, yielding unfavorable results as the number of proxies gets very small.

We extend the standard variational approximation method by allowing for more different primitives (e.g. spheres and cylinders) to represent the local geometry of a surface region. In this sense, our work is also closely related to reverse engineering [VMC97, PR98, BKV+02, PLH+04], which converts raw 3D data points into concise and explicit geometry representations. While for reverse engineering the segmentation phase and the surface fitting phase are usually separated, we merge both of them in the variational framework, i.e., segmentation and fitting are alternatingly improved driven by Lloyd iteration [Llo82], hence leading to an automatic process and better recovery of surface structures.
9.2 Variational Surface Approximation

In the following we give a more detailed description of the variational surface approximation scheme [David04] in order to set up the notations for our discussion. Trying to find an optimal piecewise-linear approximation of the input geometry, two major phases, partitioning and fitting, are repeated alternatively based on Lloyd’s clustering idea to minimize the total approximation error. Variational geometric partitioning is first applied to the input mesh $\mathcal{M}$. Each region $R_i$ in the partitioning $\mathcal{R}$ will contain a set of triangles $\{T_i\}$ with barycenters $\{g_i\}$ (note we can use $g_i$ to fit surfaces based on triangles rather than vertices). Then each $R_i$ is approximated by a planar shape proxy $P_i = (x_i, n_i)$, where $x_i$ and $n_i$ represent the “average” point and normal. After the fitting phase, the proxy set $\mathcal{P} = \{P_i\}$ approximates the whole geometry. As an alternative to the standard $L_2$ error metric, the normal-based $L_2^{1,1}$ metric is also used to evaluate the partition quality as well as to guide the fitting. For more details, the reader is referred to the original paper [David04].

The second ingredient of the variational surface approximation framework is geometric re-partitioning where each triangle is assigned to the best fitting proxy. The two steps, fitting and re-partitioning, are iterated utile convergence. This global optimization procedure is easy to implement and usually leads to better results compared to those obtained by greedy approaches. The strengths of variational shape approximation are twofold. Locally, the iterative technique is very sensitive to anisotropic bending of the surface which leads to an almost perfect alignment of the surface partition to the principle curvature directions. Globally, the technique reliably detects and merges flat regions which can be approximated by a single planar proxy.

The goal of our extension to this technique is to add more shape detection power such that higher order shape primitives like spherical or cylindrical regions can be properly recovered as well. We achieve this by integrating other classes of proxy geometries while leaving the rest of the algorithm almost unchanged. The resulting hybrid variational shape approximation techniques proves to converge to high-quality partitioning and faithful shape approximations even when the number of proxies is very small (cf. Fig. 9.13).
9.3 Hybrid Variational Surface Approximation

Aiming at accurate surface structure recovery, we derive our hybrid scheme from the standard variational surface approximation. In addition to the standard planar surface elements, we also employ other higher order primitives like spheres, cylinders and more complex rolling-ball blend patches, in the variational framework (cf. Fig. 9.13 and 9.2). As hybrid surface elements lead to more geometric fidelity, we can recover more surface structures while using much fewer primitives (e.g., fewer regions in the partitioning). This results in more flexibility and even better blueprints for subsequent geometry processing tasks.

Figure 9.2: The fan disk model (13K faces) approximated with 24 hybrid proxies. Left two are partitioning outputs and right the remeshing result with 168 triangles.

To compose our new variational approximation scheme, in the following, we will first present the definitions of the new hybrid surface elements and then explain how they can be embedded into the standard framework. Error metrics will be modified correspondingly and finally diverse surface fitting techniques are discussed.

New Surface Elements In order to faithfully approximate a local surface region \( R_i \), we use the following elements as basic shape proxies:

- Planes \( P_i = (x_i, n_i) \).
- Spheres \( S_i = (c_i, r_i) \), \( c_i \) for sphere center and \( r_i \) for radius.
- Cylinders \( C_i = (x_i, d_i, r_i) \), \( x_i \) a point on the axis, \( d_i \) its direction and \( r_i \) radius.
• Rolling-ball blend patch $B_i = (c_i(t), r_i)$, $c_i(t)$ the center trajectory and $r_i$ the ball radius. Specifically $c_i(t)$ is a uniform quadratic B-Spline curve plus two rays extending on the two ends in its tangent directions.

The motivation for this choice is mainly due to the fact that most technical CAD objects consist of patches from these four categories. Accordingly, the full proxy set now will be $P = \{P_i\} \cup \{S_i\} \cup \{C_i\} \cup \{B_i\}$. While obtaining higher approximation power, we still note that, except the blending patch element, the other two new surface primitives have similar memory consumptions as planes and hence will not bring more overhead to the approximation setting. Even though the cylinder is just a special case of the rolling-ball blend patch, we keep both of them for compactness reasons, since cylinders need less data than blend patches.

**New Elements Embedding** We can embed the above proxy geometries into the variational framework with modified error metrics (explained below). Otherwise the geometric partitioning phase is kept untouched.

The major changes appear in the proxy fitting phase. For each partitioned region $R_i$, we compute not only the best fitting plane $P_i$, but at the same time the best fitting sphere $S_i$, cylinder $C_i$ and blend patch $B_i$. From those fitted basic elements, we choose the one as the optimal shape proxy $SP_i$ that minimizes the following fitting energy:

$$E(R_i, SP_i) = \sum_{T_i \in R_i} L^*(T_i, SP_i).$$

where $L^*$ refers to the $L^2$ or $L^{2,1}$ metric.

To start the Lloyd iteration, we set all initial elements to be planes and perform a flood-based partitioning. The initial seeds can be randomly selected or by farthest point initialization as in [CSAD04]. Then in the first proxy fitting phase, better new types of surface elements can possibly be found and the iteration can continue.

Since the fitting algorithms for the new surface elements are more complex and thus slower, we use progressive partitioning In our implementation to speed up the whole process, i.e. we only use planes for fitting in the first iterations until the partitioning is close to stable (e.g., the change of approximation errors becomes very small for two consecutive iterations, this is also the terminating condition for the whole procedure). Then we also allow spheres and cylinders to fit surface regions and continue the iterations.
9.3 Hybrid Variational Surface Approximation

Finally we permit the rolling-ball blend patches in the iterations to reach the final stable geometric partitioning.

**Modified Metrics** For the $L^2$ distance metric, the extension to the new proxy shapes is straightforward: let $T_i = (v_1, v_2, v_3)$ be a triangle of area $|T_i|$ in the region $R_i$ associated with a shape proxy $SP_i = P_i \mid S_i \mid C_i \mid B_i$ and $d_1, d_2, d_3$ be the distances of vertices $v_1, v_2, v_3$ to the proxy $SP_i$. Then the $L^2$ metric can be locally approximated by the exact solution for linear proxies [CSAD04],

$$L^2(T_i, SP_i) = \frac{1}{6}(d_1^2 + d_2^2 + d_3^2 + d_1d_2 + d_1d_3 + d_2d_3)|T_i| \quad (9.1)$$

The only non-trivial question is how to compute the distance $d(v, B_i)$ from a vertex to a rolling-ball blend proxy. Here we first calculate the distance $d(v, c_i(t))$ to the center trajectory, and then subtract from it the radius $r_i$, thus $d(v, B_i) = |d(v, c_i(t)) - r_i|$.

To compute the $L^{2,1}$ error metric, the normal information $N_i$ of a shape proxy $SP_i$ near a triangle $T_i$ with normal $n_T$ and barycenter $g_i$ is required, i.e., $N_i = n_i$ for planes $P_i$, $N_i = g_i - c_i$ for spheres $S_i$, and $N_i = g_i - c_{\perp}$ for cylinders $C_i$ and blend patches $B_i$ where $c_{\perp}$ is the projection of $g_i$ onto the axis of the cylinder or onto the center trajectory of the blend patch. Note that $N_i$ has to be normalized and its direction might have to be flipped ($N_i = -N_i$) when for most triangles $T_i$ in the region $R_i$, $(n_T \cdot N_i) < 0$ holds. Finally the $L^{2,1}$ error can be computed as follows:

$$L^{2,1}(T_i, SP_i) = \|n_T - N_i\|^2|T_i|. \quad (9.2)$$

**Surface Element Fitting** In practice we find that, though fitting a plane according to the $L^{2,1}$ metric is fairly simple as pointed out in [CSAD04], it is not clear how this can be extended to our new types of shape proxies, since for spheres and cylinders, the normals do not change under scaling which implies that the radius of the proxy cannot be determined. Hence we will always use the $L^2$ metric to guide the fitting step for both metrics and fortunately it works well for all of our test scenarios. The specific fitting techniques will be presented in the following sections.
9 Robust Structure Recovery

Figure 9.3: A surface patch (orange) approximated with a sphere (left) and a cylinder (right).

9.3.1 Sphere Fitting

We use the robust least-square method of [Pra87] for sphere fitting (cf. Fig. 9.3). The sphere is represented implicitly as

\[ f(x, y, z) = A(x^2 + y^2 + z^2) + Bx + Cy + Dz + E = 0. \]

For each region \( R_i \), we have to find a sphere that best approximates the barycenters \( \{g_i\} \) which are weighted by the corresponding triangle area \( |T_i| \) to compensate for irregular sampling. The solution can be found by minimizing the quadratic form

\[ F(A, B, C, D, E) = \sum_i [f(g_i)]^2 |T_i|, \]

under the constraint

\[ Q(A, B, C, D, E) = B^2 + C^2 + D^2 - 4AE = 1 \]

which enforces a unit gradient of \( F \) near the sphere’s surface.

For this, only a \( 5 \times 5 \) general eigensystem has to be solved by using Lagrange multipliers. Usually the point set \( \{g_i\} \) will not degenerate to a line or a single point due to the teleporting mechanism in the variational partitioning, and hence we always get a proper solution. Once the parameters are computed, the approximating sphere \( S_i = (c_i, r_i) \) is specified by the coefficients \( A, ..., E \).
9.3.2 Cylinder Fitting

To geometrically fit a cylinder $C_i$ to the region $R_i$, we first use the curvature tensor field $\Gamma$ [Cohen03] to determine the direction $d_i$ of the cylinder axis. A robust statistical analysis is applied to the minimum curvature directions $\{\gamma_{min}\}$ measured at all vertices in $R_i$ to get the average direction $\bar{\gamma}$ and their deviation $\Delta \gamma$. If the deviation $\Delta \gamma$ is larger than some threshold, the region is considered isotropic thus there is no need to further approximate it with an anisotropic cylinder. Otherwise, we let $d_i = \bar{\gamma}$ and continue the fitting procedure.

We project the barycenters $\{g_i\}$ onto the plane $P$ passing through the origin $O$ with normal $d_i$ (cf. Fig. 9.4). We create a local frame $\Psi$ on $P$ and the projections $\{g_i^\perp\}$ are fitted with a 2D circle $(c_i^*, r_i^*)$ to finalize the geometric scale the cylinder (cf. Fig. 9.3). This can be computed in a similar way as the sphere fitting by minimizing

$$F(A, B, C, D) = \sum_i [f(g_i^\perp)]^2 |T_i|,$$

under the constraint

$$Q(A, B, C, D) = B^2 + C^2 - 4AD = 1,$$

where

$$f(x, y) = A(x^2 + y^2) + Bx + Cy + D = 0$$

is the circle representation.

![Figure 9.4: Cylinder fitting. Barycenters are projected to a plane (left) and a 2D circle is fitted (right).](image)
9.3.3 Rolling-Ball Blend Patch Fitting

Compared to the above methods, fitting a rolling-ball blend patch is more complicated and hence we need to derive a heuristic. We present here a three step fitting algorithm which is summarized in Fig. 9.5.

![Figure 9.5](image)

**Figure 9.5:** Rolling-ball blend patch fitting. A surface patch (top left) is first fitted with a B-Spline curve (bottom left, blue) then a set of circles is found (top right, centers in black) and finally the center trajectory is fitted with another B-Spline curve (bottom right).

1. Fit a curve \( h(t) \) to the barycenters \( \{g_i\} \) of the region \( R_i \). \( h(t) \) is defined as a uniform quadratic B-Spline curve with double end-knots:

\[
h(t) = \sum_{k=0}^{n} N_{k,2}(t)p_k, \quad 0 \leq t \leq 1.
\]

The number \( (n + 1) \) of control points \( p_k \) is restricted to 15 in most cases the suppress oscillations as shown in Fig. 9.6. The initial parameterization \( \{u_i\} \) for \( \{g_i\} \) is computed by projecting \( g_i \) to a least-square line \( L^* \) approximating the point set \( \{g_i\} \) [Jol02]. Then a fitting routine is used to find the positions of \( p_k \) by minimizing the following least-squares functional:

\[
f(\{p_k\}) = \sum_i \|h(u_i) - g_i\|^2.
\]

Since the initial parameter values may not be distributed uniformly, and because the tangent directions at the end points of the fitted curve tend to point towards the barycenters with parameters 0 or 1, we add linear constraints with large weights \( \omega \) at these two
ends \((\omega \ast (p_0 - 2p_1 + p_2) = 0 \text{ and } \omega \ast (p_{n-2} - 2p_{n-1} + p_n) = 0)\) into the above system to compensate for the instability (cf. Fig. 9.7).

2. Fit a set of circles \(\{(c_i, r_i)\}\). First we compute corrected parameter values \(t_i\) for each barycenter \(g_i\), such that: \([g_i - h(t_i)] \perp h'(t_i)\). Then we sort \(\{g_i\}\) according to \(\{t_i\}\) and for each subsequence \(g_i \cdots g_{i+m}\) (\(m\) is usually 30), we project them to a plane \(Q_i\) defined by the point \(h((t_i + t_{i+m})/2)\) and the normal \(h'((t_i + t_{i+m})/2)\). A 2D circle is fitted similarly to Section 9.3.2 and mapping it back to 3D we get a circle \(b_i = (c_i, r_i)\) within the same 3D plane \(Q_i\).

3. Fit the trajectory \(c(t)\) to the circle centers. We first have to check the validity of these circles by computing the average radius \(\bar{r}\) and its deviation \(\Delta r\). If \(\Delta r\) is larger than some threshold, there would be no consistent blending patch for the region \(R_i\) and we stop the fitting procedure. Otherwise, we simply fit another B-Spline curve, the center trajectory \(c(t)\), to the centers \(\{c_i\}\) and set the final radius of the blend patch to \(r_i = \bar{r}\).

As the center trajectory \(c(t)\) is only a finite curve with two endpoints, it is easy to imagine that during the partitioning phase, the rolling-ball blend patch could not grow at these ends. Hence we extrapolate the blend patch by extending two rays at the two ends of \(c(t)\) in tangent direction. By this an infinitely extended cylinder is added to make region growing possible. This strategy also recalls the importance of the linearity constraint shown in Fig. 9.7.

Figure 9.6: A surface patch fitted by B-Spline curves (blue) with 10 (left) and 40 (right) control points (red). More control points leads to jaggy curves and thus to an incorrect parameterization for rolling-balls fitting.
9 Robust Structure Recovery

Figure 9.7: Portion of a cylinder patch fitted by a B-Spline curve (blue, 15 control points) without (left) and with (right) linearity constraint at the end points. Note the improved quality of final fitted center trajectory (orange).

9.4 Results

By adopting more types of surface primitives, our hybrid variational surface approximation scheme can accurately recover more surface structures and geometric details. As a side product, it also can offer higher approximation quality than the original variational method if the number of proxies is fixed, especially for technical CAD models. To compare our results with other previous work, we first introduce two comparison criteria:

Remeshing Quality The segmentation which emerges from the hybrid variational approximation can be exploited to control a decimation algorithm such that it generates coarse meshes (as remeshing output) which reflect the global structure of the input mesh very well. We prefer this method over the original one in [CSAD04] because it proves to be robust even in situations when the number of proxies is extremely low.

Our decimation algorithm for surface remeshing is a very simple variant of the standard QEM technique [GH97]. All we have to do is to multiply the priority values by a factor $F$ which depends on the status of the corresponding vertex, i.e., $F = 1$ for vertices in the interior of a region, $F = 100$ for vertices on region boundaries, and $F = 1000$ for anchor vertices which belong to the boundary of three or more regions. Although this decimation technique turns out to produce fairly good meshes, in general we use this remeshing procedure mainly to generate coarse triangle meshes which allow us to compare our results to the results obtained by [CSAD04].

Vertex to Proxy Projection We not only project anchor vertices onto surface proxies as [CSAD04], but instead we map all vertices of the input mesh onto the geometric proxies that they are associated with. Projections of those vertices shared by several
proxies are averaged accordingly. This operator is used to compare the \textit{pure} geometric information captured by the surface proxies which are no longer planar in our case.

Figure 9.13 shows a complete comparison using above two quality criteria. We always use the $L^{2,1}$ metric if not otherwise specified. The original variational approximation could not provide plausible results when the number of proxies is chosen too low. In contrast, hybrid approximation can recover more surface structures and geometric details than the standard variational approximation even when only one fifth of the proxies are used (30 hybrid proxies vs. 150 planar ones). Based on the well recovered surface structure, our hybrid scheme can also produce remeshed approximations with higher quality compared to the original method.

![Figure 9.8:](image)

The rocker arm model remeshed based on 30 hybrid proxies (left, 496 faces) and decimated by QEM [GH97] (center left, 500 faces) with respective Hausdorff errors 0.19 and 0.23. The right two shows the corresponding color coded error plots. Note that our hybrid approximation result preserves more surface features and geometric structures.

The global optimization properties for the variational shape approximation naturally carry over to our hybrid extension. To demonstrate this, we compare our scheme to QEM mesh simplification [GH97] in Fig. 9.8. It is obvious that both in terms of visual quality as well as approximation error, our scheme performs better than the greedy approach.

More complicated examples are shown in Fig. 9.9 and Fig. 9.10 for technical CAD models where the color coding scheme is same as in Fig. 9.13. Notice how well the original surface’s geometric structures are recovered and expressed by only 50 or 29 hybrid proxies respectively. Although we are targeting at CAD models as our primary application, we also tested our algorithm with more organic mesh models as shown in Fig. 9.11 and Fig. 9.12. It is interesting to see that equally good structure recovery capability can be observed.
Due to the increased number of surface primitive types and their more involved surface fitting algorithms, with progressive partitioning, our hybrid method needs about 3–5 times more running time than the original scheme. However, all datasets shown in the paper have been processed in less than 3 minutes. This seems worthwhile because the higher computational costs are traded for better surface structure recovery and higher approximation quality.

9.5 Summary

Dedicated to faithfully and reliably recovering surface structures, we have extended the powerful optimization technique of variational shape approximation by allowing different types of surface primitives to represent the local geometry of clustered regions. Other than the standard planes, we also include spheres, cylinders, and more complex rolling-ball blend patches to form a new hybrid variational framework. In addition to the robust segmentation and global optimization properties which have been observed
9.5 Summary

Figure 9.10: From left to right, top to bottom are respectively the original input model (58K faces), its hybrid variational partitioning with 29 proxies, the approximation via vertex projection and the remeshed model with 400 triangles.

for the standard variational approximation scheme already, our hybrid extension can better extract the geometric configurations and hence produce better approximations with fewer primitives. This automatic and high quality structure recovery property has been verified by the partitioning and approximation results for various mechanical and organic objects.

The robustly and precisely recovered surface structures are actually composed by primitives of four different implicit functions. Due to the efficiency of our proposed method, the number of implicit primitives is usually small for a reasonable approximation quality. From these points of view, our robust structure recovery algorithm can provide highly compact implicit representations for the input 3D geometric models, which certainly favors the geometry applications in the distributed and mobile computing environments.
Figure 9.11: A torso model (80K faces, left), its hybrid partitioning with only 22 proxies (middle) and the recovered structures with approximation via vertex projection (right).

Figure 9.12: The hybrid partitioning of the bunny model (left, 70K faces) with 31 proxies (middle) and the recovered structures with approximation via vertex projection (right).
Figure 9.13: The rocker arm model (60K faces) approximated from left to right, by 150, 75, 30 planar and 30 hybrid proxies. Top row shows their respective partitioning (gray color for planes, red for spheres, green for cylinders and blue for blend patches), middle row: approximation results via vertex projection onto the corresponding proxies and bottom row: remeshed outputs with 578, 498, 508 and 496 triangles respectively and absolute maximum Hausdorff errors 0.18, 0.23, 0.26 and 0.19. Note the drastic decrease of the approximative quality when fewer planar proxies are used (center right), comparing to more favorable results of our hybrid scheme (far right) where most geometric structures can be captured with only 30 proxies.
10 Conclusion

Three-dimensional geometry is now being established as a new digital multimedia data type after text and sound in the 1980s and images and video in the 1990s. When it is combined with the rapid evolution of network technology which has brought more powerful and ubiquitous computing systems, we come to the unique research topic of this thesis, the interdisciplinary distributed and mobile geometry processing.

Within such distributed and mobile network-computing environments, the major challenge always emerges from the efficiency of underlying geometric data structures, since like all other computer science subjects, efficient algorithms always have to be supported by efficient data structures. We have summarized that our ultimate geometric data structures have to meet the following requirements: (1) Compactness, to save the network bandwidth and to have a fast processing speed; (2) Progressiveness, to make users view/process the important part of geometric data in the very beginning of the transmission; (3) Robustness, against the package loss; (4) Security, to provide ownership assertion and copyright protection of digital geometry data.

To fulfill these requirements, we have presented a wide range of high-quality algorithms to produce efficient data structures for distributed and mobile geometry processing. More specifically, for parametric representations, we have presented an in-core multiple-choice decimation algorithm as well as an out-of-core stream decimation method to produce compact and progressive representations for moderate-sized or massive parametric triangle meshes. In addition, an efficient digital watermarking scheme which can provide secure representations for triangle meshes is also introduced which can be easily extended to explicit geometric data like point sets.

For explicit representations, we have developed an optimized splat sub-sampling method for point sets to produce high-quality, compact, and robust splat-based representations. To provide the progressiveness property, a splat decimation algorithm is introduced, which can also reduce the data size of explicit splat models.
For implicit representations, we have offered a compact and progressive linear approximation to the implicit signed distance fields of the input data. A robust structure recovery algorithm is also presented to extract highly compact implicit information of the parametric surface geometry.

There still remains open problems for geometric data structures in the new distributed and mobile processing scenarios. As we are considering distributed geometry applications, it will be natural if we can have distributed and parallel algorithms to increase the performance of algorithms for generating efficient geometric data structures themselves.

In addition, it has already been shown that most geometric data structures nowadays are only symbolical representations, which only record the positions of every sample vertices. We are looking for semantic geometric representations which can describe the real surface geometric information in a higher-level and can be more compact and reasonable for distributed and mobile geometry processing applications.

Forthcoming Applications and Future Directions

What can we do next with 3D geometry? The answer will be definitely broadened by our newly proposed distributed and mobile geometry processing research directions. One key factor for the acceptance and success of a new multimedia data type is its usability and flexibility, and all our algorithms presented in this thesis are exactly trying to convey more this usability and flexibility for 3D geometry data.

Based on our highly efficient geometric data structures, we would like to see more and more improved traditional practical applications as well as new forthcoming applications for distributed and mobile geometry processing. While large-scale distributed applications mainly come from the industrial part like product design, computer simulation or movie production, small-scale mobile applications will come to be increasing ubiquitous in our everyday lives, especially in the vast number of mobile devices like mobile phones or PDAs for personal entertainment and communications.

For large-scale and distributed computing, the traditional industrial applications (cf. Fig. 10.1), e.g., Computer Aided Design (CAD) for automobile industry or paper-less product design for home/office electronics, have been strengthened by the advancing distributed technology of Computer Supported Cooperative Work (CSCW). People from
different locations can access and work on the geometric models of commercial products at same time which will greatly improve efficiency and reduce operation costs for modern companies.

Moreover, those typical computation-intensive applications like biological simulation and flight simulation can also benefit from the increasing power of distributed systems. Since huge amount of geometric data have to be delivered and displayed within these applications, we definitely require geometric data structures to be compact and progressive for efficient transmission and dynamic LOD rendering.

Last but not least, based on processing and rendering of complex geometric data, the entertainment industry like movie production will run their applications on multiple computers in a distributed way to produce high-quality special effects or computer animations.
Within these distributed geometric processing systems, parametric triangle meshes are still used for the underlying geometric data structures mainly due to their wide acceptance and availability as well as their simple computation efforts and the good support from computer hardware (e.g., graphics cards). More recently, people are trying to use different geometric structures together to represent the same geometry for more efficient geometry processing algorithms [Kob03, BBK04].

When we are coming to mobile geometry processing (cf. Fig. 10.2), we will encounter more often with small-scale personal or home applications. This is largely because of the vast number of ubiquitous mobile computing devices these days, e.g., mobile phones, personal digital assistants (PDAs), portable game consoles, and laptop computers.

Until recently, mobile platforms, previously with tiny monochrome displays and slow processors, have been equipped with high-quality true-color displays and supported by fast CPUs and in some cases with dedicated graphics hardware from major vendors, e.g., nVidia and ATI. These all have led to an interesting and quickly developing mobile graphics market [EPS+05, Pul05]. We will likely begin to see mobile geometric applications soon appearing everywhere.
With the recently standardized mobile3D APIs \cite{PAC05} for open system development, the increasing graphics capabilities of mobile devices make more and better mobile applications possible. Interesting devices and applications for mobile geometry processing can already be found with, e.g., the 3D gaming mobile phones from LG, Samsung or Nokia, and the portable game consoles (PSP) from Sony for mobile personal entertainment. We believe that many other types of mobile graphics applications will appear in the near future.

We also have developed a mobile geometry processing prototype module working on the Pocket PCs (cf. Fig. 10.2). Limited by low performance mobile clients, geometric models should be transmitted in a progressive fashion that users can see/process the important data in the very beginning: Once having the requests from clients through the wireless connection (WLAN) based on TCP/IP protocol, the server sends the base geometry model first, followed by lengthy progressive update of detail information. Clients can decide for themselves when to stop the transmission to make possible interaction and processing on the mobile devices. The rendering on Pocket PCs is achieved with a compliant implementation of the OpenGL Embedded System 1.0 API.
Moreover, many other new applications and services will appear for mobile communications with the aid of 3D geometry. Location based services from global positioning systems (GPS) and sensor networks can be improved with 3D geometric information to provide realistic guiding services. Images and geometric data can also be combined together for mobile devices to deliver real-time augmented reality [Azu95].

3D geometry also can be the general digital documents distributed on the internet (cf. Fig. 10.3 left). The clear advantage of geometry data over standard media like text and images is that, they will provide true interactivity for readers and users. This interactivity also makes the real-time interactive multimedia applications possible, e.g. online 3D shopping malls as the next generation e-commerce framework (cf. Fig. 10.3 right). Another large number of geometric applications will include online internet games (cf. Fig. 10.4), where 3D geometry works as the core element to represent and visualize the virtual environments and characters.
For the above mobile and online geometry processing applications, the high quality algorithms we present in this thesis will not only improve their rendering and network transmission performance, but also provide more flexibility for the selection of specific geometry data. In practice, we are often adopting parametric triangle meshes as they are still the current standard. But on the other hand, we are also expecting more applicability of explicit splat models as mobile geometric data structures since they are proven to be more robust during transmission and can provide higher rendering quality on low-resolution display devices.
Bibliography


Bibliography


Bibliography


[CMM+98] Richard Cole, Bruce M. Maggs, Friedhelm Meyer auf der Heide, Michael Mitzenmacher, Andréa W. Richa, Klaus Schröder, Ramesh K. Sitaraman,


Bibliography


Bibliography


Bibliography


Bibliography


Bibliography


Bibliography


Curriculum Vitae

Personal Data

Jianhua Wu
Computer Graphics Group
RWTH Aachen University, Germany
Phone: +49 241 8021 814
Email: wu@cs.rwth-aachen.de

01 Dec. 1977 Born in Zhejiang, China

Sep. 1995 – Jul. 1999 Bachelor student at Department of Computer Science and Technology, Tsinghua University, China. Finished with a Bachelor of Engineering degree.

Sep. 1999 – Jan. 2002 Master student at Department of Computer Science and Technology, Tsinghua University, China, under supervision of Prof. Shimin Hu. Finished with a Master of Science degree.

Feb. 2002 – Current Ph.D. student at Computer Science Department VIII, RWTH Aachen University, Germany, under supervision of Prof. Leif Kobbelt.

Publications

Curriculum Vitae


