FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Dynamic and Probabilistic Point-Cloud Processing

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Dipl.-Ing. Reinhold Preiner
Matrikelnummer 0430380

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer

Diese Dissertation haben begutachtet:

Prof. Dr. Marc Alexa	Prof. Dr. Robert Sablatnig

Wien, 24. August 2017

Reinhold Preiner

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Dynamic and Probabilistic Point-Cloud Processing

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Reinhold Preiner
Registration Number 0430380

to the Faculty of Informatics

at the TU Wien

Advisor: Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer

The dissertation has been reviewed by:

Prof. Dr. Marc Alexa	Prof. Dr. Robert Sablatnig

Vienna, 24th August, 2017

Reinhold Preiner

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Reinhold Preiner
Murlingengasse 56/4, 1120 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 24. August 2017

Reinhold Preiner

Acknowledgements

First, I would like to thank my PhD supervisor Prof. Michael Wimmer, who channeled my career into research and hired me as a university assistant after my master studies. He has been a distinguished group leader and a continuous source of professional advice. Thanks to Prof. Werner Purgathofer, the head of the institute, who led it in an exemplary manner and created the prerequisites for a positive working environment. Thanks to our technicians Andreas Weiner, Stephan Bösch-Plepelits, and Simone Risslegger, who created the technical prerequisites for any work that has been done. And to the administrative staff, Tammy Ringhofer, Max Höfferer, and especially Anita Mayerhofer-Sebera, the heart of the institute. I couldn't have done my research without my co-authors Stefan Jeschke, Oliver Mattausch, Murat Arikan, and Prof. Tamy Boubekeur. I also want to thank Claus Scheiblauer, Martin Knecht, Thomas Auzinger, and Stephané Calderon for their invaluable help and input to my research work. I have also been grateful to get the chance to collaborate with my colleagues from the ICGA Visualization group, especially Johanna Schmidt, Gabriel Mistelbauer and Eduard Gröller.

My research was funded by the Austrian Research Promotion Agency (FFG) through the FIT-IT project "Terapoints" (no. 825842), the EU FP7-ICT project "Harvest4D" (no. 323567), and the Austrian Science Fund (FWF) grant P24600-N23. Thanks to the people and institutions who freely provided the 3d models and 3rd-party tools used in this thesis, including the University of Konstanz, Hui Huang, Robert Sumner and Jovan Popovic from the Computer Graphics Group at MIT, Clark Mills and the Smithsonian Digitization Project 3D Lab, Matthew Berger, and many others.

It is an honour to me to have Prof. Marc Alexa and Prof. Robert Sablatnig as reviewers for this thesis.

To my family, especially my parents, in-laws, and grandparents, for all the support over the years. To my friends; those I met in Vienna, for making me enjoy the time of my PhD studies on countless occasions; and those I have known long before, for keeping up the connection throughout the years. Most of all, to my beloved wife Margarete, who provided me with the emotional and motivational support during the past years. I would not be here without you.

Kurzfassung

Das Scannen physikalischer Objekte mittels 3d-Scanner ist in den letzten Jahren zu einer alltäglichen Aufgabe geworden. Diese Geräte tasten die Oberfläche dreidimensionaler Objekte ab und geben die Abtastpunkte, auch *Punktwolke* genannt, aus. Diese Punktwolken weisen charakteristischerweise ein gewisses Rauschen, unabgetastete Oberflächenregionen (Löcher), und statistische Ausreißer auf. In der Computergrafik befasst sich das Gebiet der *Oberflächenrekonstruktion* mit dem Problem der Konvertierung dieser rohen Punktdaten zu einer sauberen, vereinfachten Repräsentation, die sich besser für die Anwendung verschiedenster Operationen sowie die Darstellung des Objektes eignet. Manche der entwickelten Methoden zielen auf die direkte Darstellung einer geschlossenen Oberfläche basierend auf einer gegebenen Punktwolke ab, benötigen aber oft auch gewisse Vorverarbeitungsschritte, um Oberflächendarstellungen von hoher Qualität zu erzielen.

Die ständig steigenden Datenraten aktueller 3d Scanner haben es kürzlich möglich gemacht, dynamische, bewegte Objekte in Echtzeit zu scannen. Diese hohen Abtastgeschwindigkeiten erfordern neue leistungsfähige Rekonstruktionsalgorithmen, die die gescannten Daten in Echtzeit verarbeiten und in hoher Qualität darstellen können. In dieser Arbeit beschäftigen wir uns daher mit der Entwicklung verschiedener Techniken zur schnellen Verarbeitung und Darstellung solcher unstrukturierter, dynamischer Punktwolken, die aus verschiedenen Quellen zu einem Computer gestreamt werden können. Im Zuge dieser Arbeit erforschen wir probabilistische Ansätze zur Beschleunigung aktueller punktbasierter Operationen, und verwenden ein Wahrscheinlichkeitsmodell der 3d-Punktdaten, um eine effiziente Methode zur probabilistischen Oberflächenrekonstruktion und -representation zu entwickeln.

Wir entwickeln ein GPU Rendering-Framework, das die jeweils notwendigen Berechnungen zur Oberflächenrekonstruktion *in situ*, d.h. zum Zeitpunkt des Renderns, durchführt, und sich dabei auf eine minimal notwendige Teilmenge der Punktdaten, also die im *Viewport* sichtbaren Punkte, beschränkt. Dazu befasst sich der erste Teil dieser Arbeit mit einem grundlegenden Problem der meisten Rekonstruktionsverfahren, der schnellen und effizienten Suche der räumlichen Nachbarpunkte in einem großen, unstrukturierten und ungeordneten Punktdatensatz. Die Verfügbarkeit der lokalen Nachbarn eines jeden Punktes ist essenziell für die Herstellung der lokalen Konnektivität zwischen den Punkten, für die Beurteilung der lokal umgebenden Geometrie der zu rekonstruierenden Oberfläche, sowie für die Anwendung von Filteroperationen, die die Qualität der Daten und somit der

resultierenden Oberfläche verbessern. Im zweiten Teil erweitern wir diesen Ansatz und beschreiben ein verbessertes Verfahren, das eine Rekonstruktion mit beliebigen Bandbreiten erlaubt und eine stabilere und insgesamt gleichmäßigere Oberflächendarstellung ermöglicht.

Im dritten Teil konzentrieren wir uns auf das Problem von Rauschen und Ausreißern in den Punktdaten, und entwickeln eine neue Methode, die ein schnelles, detailerhaltendes Resampling solcher dynamischer Punktwolken erlaubt. Hierzu beschreiben wir die Daten durch eine reduzierte Gaußsche Mischverteilung (Gaussian Mixture Model), die eine viel kompaktere Beschreibung der Daten und somit auch schnellere Operationen auf diesen erlauben. Wir zeigen, dass diese Methode nicht nur die Geschwindigkeit, sondern auch die Genauigkeit und Qualität aktueller Resamplig-Operatoren verbessert. Basierend auf den beobachteten Vorteilen dieses Wahrscheinlichkeitsmodells für punktbasierte Operationen erforschen wir im letzten Teil dieser Arbeit einen neuen Ansatz, um auf solchen reduzierten Wahrscheinlichkeitsmodellen eine glatte, durchgehende Oberfläche zu definieren. Wir entwickeln hierfür ein gänzlich probabilistisches Rekonstruktionsverfahren, und zeigen, dass wir detailreiche Oberflächen in äußerst speichereffizienter Weise darstellen können, während die Geschwindigkeit unseres Verfahrens aktuelle Methoden übertrifft.

Abstract

In recent years, the scanning of real-world physical objects using 3d acquisition devices has become an everyday task. Such 3d scanners output a set of three-dimensional points, called *point cloud*, sampling the surface of an object. These samples are typically subject to imperfections like noise, holes and outliers. In computer graphics, the field of *surface reconstruction* is concerned with the problem of converting this raw point data to a clean, more usable and visualizable representation. Some of the developed techniques aim at directly rendering a closed surface from such a point cloud, but also require certain precomputations to produce images of high quality.

The ever-increasing acquisition rates and data throughputs of even low-cost scanner hardware have recently made it possible to capture dynamic and animated objects in real time. These high-speed acquisition capabilities call for new high-performance reconstruction algorithms that are able to keep up with these acquisition rates and allow an instant high-quality visualization of the real-time captured data. In this thesis, we pick up on this problem and develop various techniques that allow a fast processing and visualization of such raw, unstructured and potentially dynamic point clouds, which might be streamed to our computer at real-time rates from any possible source. In the course of our work, we investigate probabilistic methods that allow achieving a significant acceleration of state-of-the-art point-based operators, and use statistical models of 3d point sets to develop a fast technique for probabilistic surface reconstruction and representation.

We develop a GPU point-rendering framework that performs any reconstruction computations required for a high-quality visualization *instantly*, i.e., on the fly at render time, and only on a necessary minimal subset of the data, i.e., the points visible on the *screen*. To this end, the first part of this thesis addresses a basic problem common to almost any surface-reconstruction technique, which is the fast and efficient search for spatial neighbors in an unstructured and unordered large collection of points. Knowing about a point's nearest neighbors is an essential prerequisite for establishing local connectivity, assessing the shape of the surrounding surface, and applying filter operations for improving the quality of the geometric data and thus the resulting surface. In the second part, we improve on this direct reconstruction and rendering technique and present a more elaborate method that allows working at arbitrary reconstruction bandwidths, improves

on the temporal stability of the rendered image, and produces a surface rendering of increased smoothness.

In the third part, we focus on the problem of noise and outliers in the input data, and introduce a novel technique that allows for a fast feature-preserving resampling of unstructured dynamic point sets at render time. To this end, we describe the point cloud by a sparse probabilistic Gaussian mixture model, which allows for a much more compact representation and thus much faster operations on the spatial data. We will show that this technique significantly improves on the speed and even on the accuracy and quality of a feature-preserving point-set resampling operator. Based on the observed computational benefits of this probabilistic model, the final part of this thesis investigates a new way of defining a smooth and continuous surface solely based on a sparse Gaussian mixture. We will develop an entirely probabilistic reconstruction pipeline, and show that we can describe a feature-rich surface in a highly memory-efficient way while obtaining a reconstruction performance that can compete and even improve on the state of the art.

Contents

Kurzfassung ix

Abstract xi

Contents xiii

1 Introduction 1
- 1.1 Prelude . . . 1
- 1.2 Scope of this Thesis . . . 2
- 1.3 Challenges in Dynamic Point Rendering and Reconstruction . . . 5
- 1.4 Contributions . . . 6

2 Related Work 11
- 2.1 Dynamic High-Quality Point Rendering . . . 11
- 2.2 Robust Point-Cloud Processing . . . 12
- 2.3 Modeling Point Clouds with Gaussian Mixtures . . . 13
- 2.4 Surface Reconstruction and Representation . . . 14

3 Interactive Screen-Space Triangulation 19
- 3.1 Introduction . . . 20
- 3.2 Challenges . . . 21
- 3.3 Algorithm Overview . . . 22
- 3.4 Nearest neighbor search . . . 23
- 3.5 Normal estimation . . . 25
- 3.6 Triangulation . . . 25
- 3.7 Search radii update . . . 26
- 3.8 Results . . . 27
- 3.9 Conclusion and Limitations . . . 32

4 Auto Splats 35
- 4.1 Introduction . . . 36
- 4.2 Overview . . . 37
- 4.3 Parallel Splat Communication . . . 38
- 4.4 Neighborhood Computation . . . 40

4.5 Surface Fitting . . . 44
4.6 Auto Splat Rendering . . . 45
4.7 Results and Discussion . . . 47
4.8 Summary . . . 52

5 Continuous Locally Optimal Projection **57**
5.1 Introduction . . . 58
5.2 A Review of the LOP Operator . . . 59
5.3 Motivation and Overview . . . 61
5.4 Gaussian Mixture Density Computation . . . 61
5.5 Continuous LOP in Gaussian Mixtures . . . 65
5.6 Weighted CLOP . . . 69
5.7 Accelerating Repulsion . . . 70
5.8 Robust Normal Computation in Mixtures . . . 71
5.9 Implementation . . . 75
5.10 Evaluation and Results . . . 76
5.11 Limitations . . . 84
5.12 Summary . . . 85

6 Gaussian Kernel-Product Surfaces **89**
6.1 Introduction . . . 90
6.2 Motivation and Overview . . . 91
6.3 Gaussian Mixture Computation . . . 94
6.4 Probabilistic Triangulation of Gaussians . . . 95
6.5 Kernel-Product Surfaces . . . 98
6.6 Results and Discussions . . . 105
6.7 Summary . . . 116

7 Conclusion **117**
7.1 Résumé . . . 117
7.2 Epilogue . . . 118

A Virtual Scanning Parameters **121**

B HEM Algorithm Outline **123**

C Derivation of the Kernel-Product Expectation **125**

List of Figures **127**

List of Tables **129**

Bibliography **131**

Curriculum Vitae **141**

CHAPTER 1

Introduction

1.1 Prelude

In the field of computer graphics, the representation, processing and rendering of virtual objects (we call them *models*) plays a central role in a wide range of applications. For the purpose of virtual preservation, exploration and study of objects in archeology and building history, or computer-aided applications in urban planning and architecture, to name just a few, the models of interest are mostly virtual representations of real three-dimensional objects and scenes that have been captured using different kinds of 3d-scanning devices or stereo-photogrammetric techniques.

In general, the fundamental representation of the geometry information outputted by such acquisition devices is a set of 3d point samples of the surface of a scanned object, optionally augmented with additional sample attributes like surface normal estimates, color, or even rough connectivity estimates that are inferred based on the scanner's point of view. Such 3d point sets (also called *point clouds*) offer various computational advantages due to the simplicity of their representation.

However, raw point clouds also come with several drawbacks, making them unsuitable for direct use in many applications: Firstly, individual point samples do not represent a closed continuous surface that could be directly rendered or would allow an analytic or discrete differential geometric reasoning. Second, the spatial information of their samples is often subject to blur, noise, and non-covered regions (holes) due to physical limitations of the various acquisition processes. And third, point clouds sampling a surface often contain a lot of redundant information, especially in large planar regions, and can thus exhibit a large memory footprint for storing the data.

In the past decades, a plethora *point-processing* techniques have therefore been developed in order to filter, denoise or resample such point sets, or render smooth continuous surfaces based on enriched point data. Alongside – and often based on – these methods,

a vast amount of work has been published in the field of *surface reconstruction*, aiming at the definition of a continuous surface based on the point samples, and/or their conversion to a different surface representation like polygon meshes. Most of these methods require a considerable computation time, even for moderately sized point clouds.

The increasing data throughputs of nowaday's acquisition devices have recently opened up the possibility of performing the acquisition in *real time*. Currently available cheap low-fidelity sensors like the Microsoft Kinect can stream several hundreds of thousands of depth values per frame at 30 Hz, resulting in a throughput of several million three-dimensional point samples per second. These real-time acquisition rates and ever-increasing data sizes call for an on-the-fly processing and rendering of *dynamic 3d data* from moving real-life objects, which still poses a huge challenge for current high-quality point-processing and reconstruction methods. Common applications for these kinds of online point-processing techniques can be found in various fields like the film industry, rapid prototyping, virtual and augmented reality, and others.

The research work presented in this thesis focuses on the development of fast and efficient high-quality techniques for point-based processing, rendering, and surface reconstruction, that allow for the **fast processing of dynamic point clouds**, pushing the performance limits of current methods from the offline to the interactive domain. In particular, we introduce new **probabilistic approaches** to point-cloud processing and surface reconstruction, which outperform state-of-the-art methods in speed and accuracy, and allow for new compact surface representations that drastically reduce the required storage memory footprint.

1.2 Scope of this Thesis

In the following, we will give a brief overview of the computational paths and involved data representations in point-cloud processing and surface reconstruction that are most relevant in the scope of this thesis. We will then identify the computational challenges that represent the motivation to the work presented in this thesis. A simplified illustration of the most important computational pathways in classic point-based reconstruction is shown in Figure 1.1.

Point-Based Processing

This field involves computations that operate on a per-point basis and have point sets as input and output. These computations normally precede any point-based visualization or surface-reconstruction processes, and aim at transforming, filtering and enhancing the raw point data obtained by a scanner. At the initial acquisition process, most 3d-scanning devices perform depth sensing from a stationary point of view, which means that multiple scans from different positions and angles are necessary in order to cover the entire surface of an object. This results in a number of disconnected partial point clouds that have to be aligned to a common global shape in an initial *registration* step. Physical inaccuracies

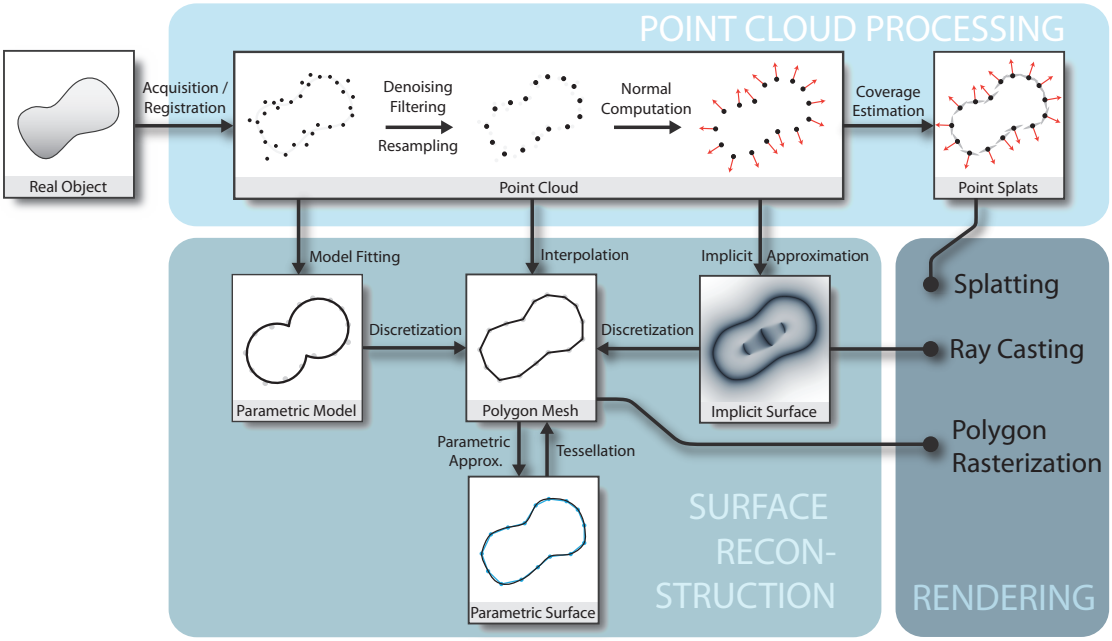


Figure 1.1: Possible computational paths in classic point-based processing, reconstruction, and rendering.

of the acquisition device and computational errors in the registration process are common sources of noise and outlier points, which disturb the sampled signal and can corrupt the consecutive reconstruction results. In many cases it is therefore beneficial to apply *denoising*, *filtering* or *resampling* operators to the data in advance in order to produce a clean point set.

An important class of further point-based processing is the computation of additional geometric attributes like surface normals or surface coverage estimates for each point. Especially per-point *surface normals* are important for lighting and shading computations in point-based rendering, but are also essential for most consecutive surface reconstruction techniques. In fact, many surface reconstruction methods also require a *consistent orientation* of these normals, for instance to allow to consistently interpolate between them, or to establish a sense of sidedness with regard to the surface. *Surface coverage* estimates are used to specify the approximate fraction of the surface a particular point sample is locally covering with respect to its distance to its neighbor points. This information is particularly important for direct point-rendering techniques, where a watertight surface is often rendered by *"splatting"* surface elements of non-zero area to the screen. The estimate itself can be given as a scalar radius, or an elliptical disc that more accurately describes the local coverage of anisotropic regions. Different splatting techniques either use box, circular or elliptical splat primitives, are either rendered as screen-aligned billboards or 3d geometry oriented with a surface normal, and use different ways of blending in order to obtain both a closed and a smooth surface rendering.

Surface Reconstruction

This term refers to methods that aim at producing an explicit or implicit representation of a continuous, closed surface based on the given input point samples. The continuity order (smoothness) of the resulting surface thereby depends on the choice of representation as well as the method of computation itself. One of the most widely used surface representations in CG applications are *polygon meshes*, which discretize smooth surfaces by manifold vertex-edge graphs representing piecewise linear, C^0 continuous surfaces. From early on, graphics hardware has been optimized to render mesh-based surfaces using polygon rasterization. Similar to point clouds, the importance of polygon meshes has also led to the development of a large number of mesh-based processing techniques like re-meshing, simplification, and discrete counterparts of geometric and differential operators, which however lie outside the scope of this thesis. A direct way of generating meshes from point clouds is the *interpolation* of neighboring points. In this case, the points themselves are chosen as the vertices of the resulting mesh, and their connectivity is inferred based on heuristic geometric reasoning, like the well-known Delaunay criterion. These methods are however highly sensitive to noise and insufficient sampling in the point data.

A common alternative class of methods that acknowledge the noisy, imperfect nature of the point samples is their approximation by a (smooth) *implicit surface*. Based on the positions and optionally normals of the point samples, these methods define particular kinds of signed or unsigned distance functions $F(\mathbf{x})$ over $\mathbb{R}^3$, and define the surface to be the iso-surface of the zero set $F(\mathbf{x}) = 0$. F can be defined as superimposition of stationary atomic functions like radial basis functions (RBFs) centered in the points, or by using a moving weighting kernel that dynamically alters the influence of each sample based on the evaluation point $\mathbf{x}$, as is the case for the family of moving least-squares (MLS) methods. The shape and smoothness of the resulting surface, its sensitiveness to the presence of noise and outlier points and the preservation of features are influenced by the size and shape of the involved weighting kernels. Most implicit reconstruction methods additionally employ oriented sample normals for the definition of F to also achieve a good approximation in the gradient domain. Through *discretization*, smooth implicit surfaces can again be converted to a piecewise linear mesh, for instance using the popular Marching Cubes algorithm, which polygonizes the points of intersection between the iso-surface and the edges of a regular grid. However, the analytic definition of these surfaces also allows for a direct visualization by *ray casting*, where the surface is found via one-dimensional numerical root-finding along individual view rays that are intersected with F.

A given polygon mesh can also be converted into a smooth surface by explicitly interpolating or approximating its vertices by a smooth 3d vector function $f(u, v)$, resulting in a *parametric surface*. The underlying control mesh thereby determines the topological structure of the resulting surface and provides vertices as anchor points for the parametric functions. These functions normally only define smooth surfaces for partial, two-dimensional mesh patches that allow a bijective mapping to the surface. For more

complex surfaces of higher genus, a global mapping is therefore often infeasible or not possible without the appearance of singularities. This leads to different strategies for decomposing surfaces into patches and blending or joining these patches at their common border, which further results in different continuity orders at patch borders and corners. Examples are Bezier and spline surfaces, but also subdivision surfaces, where smooth surfaces are obtained by applying a recursive subdivison rule to the input mesh. Similarly, the predefined topological structure of the underlying mesh allows for an efficient mesh refinement, or *tessellation*, using a given parametric surface definition. Nowaday's graphics hardware performs this tessellation on the fly during rendering in order to rasterize smooth surfaces.

Finally, to define a surface we can fit predefined *parametric models* to the point data by optimizing a reduced set of model parameters with respect to a certain error function. These models can be *geometric* priors, which allow to use prior knowledge about the shape or its parts in the reconstruction and to convert the resulting model again to a corresponding mesh representation. For the sake of completeness, it should be noted that some implicit reconstruction methods also inherently employ simple geometric or algebraic priors like planes or spheres for a local approximation of a surface. However, parametric models can also be *probabilistic* models, which are fitted to the point data and provide a simplified description of their probability distribution through a small number of statistical parameters. Although a direct conversion of such a representation to an explicit surface definition is not obvious and straightforward as for parametric geometry priors, we will show later in this thesis that this conversion can be efficiently established and yet lead to fast reconstruction timings and even a highly memory-efficient parametric surface representation.

1.3 Challenges in Dynamic Point Rendering and Reconstruction

In the following, we highlight the most important challenges for dynamic point-cloud visualization and point-based reconstruction that represent the motivation for the research results presented in this thesis.

Direct Visualization of Dynamic Point Data. The high-quality rendering of a point cloud as a smooth, closed surface generally requires a reconstruction of this surface in some of the above-mentioned forms, which is still a very time-consuming process. In particular, a complete reconstruction of a surface, for instance as a mesh, is not feasible if dynamic point clouds should be visualized directly from a scanner feed or during editing operations in real time. An alternative approach is to visualize point clouds directly through *splatting*. High-quality splatting techniques, however, still require a *local reconstruction* of the surface, by computing an elliptical surface element that represents the local tangent plane and the surface coverage in each point. These computations involve expensive *spatial neighbor queries* in the point data, which collect the neighboring

points required to assess the local shape of the surface around a given query point. These queries are normally accelerated by employing spatial search structures like octrees or kd-trees. In case of dynamic scenes, these trees would have to be rebuilt in each new frame, which means a considerable computational overhead, even when performed on the GPU. Therefore, performing sufficiently fast local neighbor queries for an on-the-fly reconstruction of point splats is the key difficulty for the problem of high-quality dynamic point-cloud rendering.

Efficient Noise Removal. Faced with noisy point data, we wish to apply robust, feature-preserving point-filtering techniques that minimize the filter bias. Most of them, however, again require expensive neighbor queries or even iterative operations on each single point, which poses an additional problem when intended for an on-the-fly application to a dynamic point stream.

Efficient Surface Reconstruction and Representation. One general problem in surface reconstruction from large point clouds is the bad performance- and memory scalability of state-of-the-art reconstruction techniques. For instance, smooth implicit surfaces are defined on the entirety of input points and thus require the storage of the whole input data set in order to represent or recreate the surface. Resampling or subsampling a point set allows reducing its memory footprint, but also comes with a certain information loss, leading to a degeneration of features in the surface. Converting such a surface to a polygon mesh allows for a strong reduction of its memory footprint as well, but only represents a discretized version of the original surface. A scalable definition and representation of a continuous surface, which remains time- and memory-efficient even for large data sets, is therefore still a challenging problem in point-based reconstruction.

Dependency on Normals. Another limitation of many surface-reconstruction methods is their dependency on consistently oriented surface normals for the points. These are, however, not always available, and can also not be guaranteed to be consistently computed due to noise, insufficient sampling, or a principal non-orientability of the sampled surface. On the other hand, methods that do not make use of reliable per-point normal information often exhibit robustness problems in the reconstruction of the local surface shape, especially in the presence of noise, thin sheets and high-curvature regions, where ambiguous interpretations of the local point data can lead to artifacts and possible surface discontinuities. Robustly reconstructing surfaces for geometrically complex, non-orientable raw point data therefore remains an open problem for state-of-the-art reconstruction methods.

1.4 Contributions

In this thesis, we will present a number of techniques that address the above-mentioned challenges and have contributed to the state of the art in the different point-based

processing stages shown in Fig. 1.1. We categorize them into four main contributions, described in the following chapters:

Interactive Screen-Space Triangulation (Chapter 3)

In this chapter, we address the fundamental core problem of any online reconstruction of local surface information for direct high-quality point rendering, which is the efficient execution of spatial neighbor queries. We develop a rendering technique that performs the nearest-neighbor search for each visible point on screen *at render-time* using the programmable GPU rasterization pipeline. By drawing for each point quads of particular sizes in screen space, we establish an information exchange of neighboring points in world space, which allows for an indexing of neighboring points and thus a subsequent local reconstruction of the surface in real time. To render a piece-wise smooth, closed surface, the umbrellas of local one-ring neighbors are triangulated for each point and rasterized by blending their depth, color and normal information to different frame buffers, which allows for applying subsequent deferred shading. This technique of *rasterized neighbor queries* therefore represents the cornerstone for an output-driven in-situ reconstruction of unordered, raw dynamic point clouds, superseding the need for the dynamic creation of spatial search data structures. The method was published in the following Technical Report:

- Reinhold Preiner and Michael Wimmer. Interactive screen-space triangulation for high-quality rendering of point clouds. Technical Report TR-186-2-12-01, Institute of Computer Graphics and Algorithms, TU Wien, April 2012.

Auto Splats (Chapter 4)

Based on the screen-space neighbor queries developed in Chapter 3, this chapter presents a technique for a dynamic computation of point normals and splats at render-time. This allows for a high-quality splat-based rendering of a continuous, smooth surface from a stream of raw, dynamic point clouds. The technique extends on the previous method by actually computing the k-nearest-neighbors in world space via an iterative range search in screen space. This alleviates the view-dependency of the reconstruction, and thus makes the resulting image stable under camera movement. Moreover, instead of explicitly indexing the neighbors for each point, the necessary accumulative operations for the normal and splat computations are all embedded in the rasterization of screen-space quads that carry a point's spatial information over to its neighbors. Like for the previous contribution, the advantage of this automatic splat-computation technique is its output sensitivity, since splats are only reconstructed for the visible part of a scene. The results of this work were published in

- Reinhold Preiner, Stefan Jeschke, and Michael Wimmer. Auto Splats: Dynamic point cloud visualization on the GPU. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization*, pages 139–148. The Eurographics Association, May 2012.

Continuous Locally Optimal Projection (Chapter 5)

In this chapter, we introduce an efficient technique that allows the integration and dynamic execution of a *resampling stage* in the online GPU splat-computation framework introduced in the previous chapter. Our method accelerates a previously offline robust L_1 resampling operator by converting the input point cloud into a sparse probabilistic representation and transferring this operator to the continuous probabilistic domain. To this end, we efficiently compute a Gaussian mixture model describing the probability distribution of the input points by a strongly reduced number of anisotropic Gaussian components. This strong reduction of spatial entities allows for a significant easing of the computational effort for the required spatial neighbor queries, making our method several times faster than its discrete counterpart and even eligible for online execution. Therefore, we are now able to dynamically resample points and compute normals and splats based on the cleaned, simplified data entirely in screen-space and at interactive framerates for any given stream of dynamic point data. The method was published in

- Reinhold Preiner, Oliver Mattausch, Murat Arikan, Renato Pajarola, and Michael Wimmer. Continuous projection for fast L1 reconstruction. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2014)*, 33(4):47:1–47:13, August 2014.

Gaussian Kernel-Product Surfaces (Chapter 6)

Finally, this chapter follows the advantageous properties of Gaussian mixtures to express large point data by a sparse, memory-efficient set of Gaussians, and develops a novel surface definition based on this probabilistic model. To this end, we introduce a probabilistic triangulation procedure for a set of anisotropic Gaussians modeling points sampled from a surface, and develop a novel parametric interpolation of these Gaussians that produces a continuous smooth surface. We show that due to the sparse probabilistic model, both the required surface reconstruction timings and the memory footprint of the resulting surface representation, i.e., a mesh of Gaussians, are several times lower than those of state-of-the-art surface-reconstruction techniques. Furthermore, since our method is only based on Gaussian mixtures computed from raw points, it is independent from the availability of preprocessed oriented normals. A manuscript describing this reconstruction technique has been authored and submitted for publication by Reinhold Preiner, Tamy Boubekeur and Michael Wimmer.

When you think outside the bun, it's scary out there
There's also lots of stinky poo.
But sometimes when you go outside of your comfort zone
You might find something brand new.

— Tenacious D

CHAPTER 2

Related Work

In this chapter, we will review previous work that relates to the contributions of this thesis listed in the previous chapter, and discuss their properties and limitations that have led to the development of the novel techniques presented in the following chapters.

2.1 Dynamic High-Quality Point Rendering

Point-based rendering has been an intensively researched topic in the past two decades, and has been subject to an excellent book [GP07] and several surveys on this field [Gro09, KB04]. Early algorithms dealt almost exclusively with the problem of rendering models that already contain complete information about point orientation and extent, which led to the introduction of *surfels* (surface elements) as point-based surface- and rendering primitives [PZvBG00], and to the development of related rendering algorithms. Ranging from explicit splat rasterization and blending [RPZ02, BWK02, BK03, BSK04, BHZK05] to screen-space pull-push interpolation for reconstructing closed surface images [MKC07, MOC08, RL08, DRL10], these methods all share the disadvantage that they rely on the precalculation of point normals and splat radii, making them unsuitable for direct online application on a dynamic point stream. Other screen-space image-reconstruction methods lift the requirement for precalculated normals: Diankov and Bajcsy [DB07] applied erosion and dilation operations to fill holes, but only achieve limited image quality. Kawata et al. [HK04] base their reconstruction on a downscaled image, which is not applicable to input with varying point densities due to a user-provided, fixed grid size. Gobetti and Martin [GM04] and Wimmer and Scheiblauer [WS06, SZW09] also abandoned the requirement of precalculated normals and allowed visualizing huge point clouds in their layered respectively instant points systems. Both methods mainly focus on the aspect of rendering huge point clouds that do not fit into graphics memory or even main memory by using clever hierarchies. While the former basically relies on a dense enough sampling to avoid holes in the reconstruction, the latter provides a heuristic to

render quad-shaped splats to obtain a closed surface. However, such quad splats do not allow for smooth interpolation and are prone to artifacts if not viewed head-on, which significantly limits the obtainable image quality.

In this thesis, we aim at a high-quality splat rendering [BHZK05], by reconstructing splats from a dynamic stream of unorganized points at render-time. A standard approach for per-point splat estimation is to compute each point's k nearest neighbors and fit a plane into the neighboring points [HDD+92]. For these methods, the performance-critical part is the computation of the k-neighborhood for a large number of points, which mostly requires spatial acceleration data structures like grids or trees. Various techniques have been developed to perform KNN-Search interactively, mostly by utilizing modern graphics hardware [ZHWG08, PLM10, QMN09, LTdF+09, GDB08, JGBZ10]. While these approaches are able to reach fast peak performances, their efficiency mostly depends on carefully chosen parameters, which only perform well up to a certain data size due to hardware limitations (number of threads per block, shared memory occupancy, etc). In Chapters 3 and 4, we will progressively develop means for performing a local splat reconstruction of dynamic point streams in screen-space, allowing for an instant high-quality splat rendering.

2.2 Robust Point-Cloud Processing

Almost any application based on point clouds, be it point-based rendering, measuring, or surface-reconstruction, requires or benefits from a prior consolidation or improvement of the data, either in form of denoising, filtering, or enrichment by additional geometric attributes like normals. Simple denoising techniques peform an inherent smoothing of the signal, e.g., by performing a moving least-squares (MLS) approximation of the data [Lev98]. These methods are generally very efficient, but typically come with a smoothing bias that degenerates high-frequency features.

Robust Denoising. Feature-preserving MLS methods have been proposed, which either find explicit piecewise smooth surface patches through repetitive forward searches [FCOS05] or again depend on preprocessed oriented normals for an iteratively reweighted robust alignment [OGG09], making them infeasible for an online execution on dynamic point data. Particularly attractive for robust *online* denoising is the Locally Optimal Projection (LOP) operator [LCOLTE07], which resamples a raw input point set without putting too many constraints on the nature of the data, i.e., it does not require a well-defined surface parametrization, nor a surface that can be locally well approximated by a plane, and also does not depend on per-point normals as additional input. LOP attracts a number of resampling points (particles) to the local median of the input points and employs repulsive inter-point forces to achieve a balanced spacing between these particles. The algorithm is related to the Weiszfeld-algorithm for finding the L_1-median [HAT11], which is known to be statistically robust against a theoretical amount of up to 50% of outliers. Although LOP is highly parallelizable because of its local support, the original algorithm

is still relatively expensive for interactive applications, since solving Weiszfeld's problem of finding the spatial median requires an iterative approximation of the L_1-minimum. Various variants of LOP have been proposed. Weighted LOP (WLOP) [HLZ+09] deals with unevenly sampled point clouds by taking into account a local density measure which relaxes the attractive forces in denser regions and hence reaches more evenly distributed points. Kernel LOP (KLOP) [LXJF13] reduces the computation cost of LOP by subsampling the point cloud using a kernel density estimate (KDE). While this reduction achieves a decent acceleration, reducing the number of discrete input samples also constrains the number of usable resampling particles [LCOLTE07], thus the general reconstruction quality deteriorates quickly for a small number of kernels. In Chapter 5, we will introduce a method that describes the whole KDE of a point set by a Gaussian mixture, which enables us to perform a *continuous* LOP resampling at high particle rates using only few Gaussian components.

Robust Normal Computation. Another fundamental problem for further point rendering or surface reconstruction is the estimation of normals for a given point set. Basic approaches use some form of local plane fitting [HDD+92], but noisy point sets with outliers and possible sharp features again require more robust normal estimations. Mitra et. al [MNG04] analyzes error bounds on normals estimated in noisy point cloud data. Robust approaches range from inscribing empty balls [DS06], smoothing and outlier removal [HLZ+09], global L_1 norm optimization [ASGCO10] to randomized Hough transforms [BM12]. Moreover, robust statistics-based methods have been shown to achieve superior results in the presence of outliers [KSNS07, LSK+10, ZSW+10, OGG09]. We will show that the continuous LOP operator described in Chapter 5 can be directly adopted to achieve an equally efficient computation of L_1-robust per-point normals.

2.3 Modeling Point Clouds with Gaussian Mixtures

Gaussian mixtures are probabilistic models that describe the probability distribution of large, complex multivariate data by a superposition of a sparse, tractable set of Gaussian components. This model has been used for the abstraction/simplification of large data in various scientific fields like image segmentation [GNN10], object recognition [Vas98] and rendering [WBKP08, JRJ11], but also for registration [JV11, DMKF16] and filtering [CB14] in point-based processing. A popular method for computing a mixture that represents a given set of input points in the sense of maximum likelihood is *Expectation-Maximization* (EM) [DLR77], which iteratively optimizes the model parameters of a predefined number of Gaussian components. A faster, *hierarchical* EM variant (HEM) proposed by Vasconcelos [Vas98] performs this parameter optimization in the process of an agglomerative clustering of a set of base Gaussians.

Gaussian pdfs have also been used for the extraction of curves and *surfaces*, where they describe data representing a one- or two-dimensional manifold [SG07, LLP+10]. For example, Suessmuth et al. [SG07] found the ridge surface of a dense Gaussian mixture by

evaluating differential geometric quantities of its pdf on a regular grid and contouring the ridge via marching cubes. To avoid the triangulation of secondary spurious ridges, which appear from its differential geometric definition, they started at a significant maximum of the pdf and traced the primary ridge from there. This method depends on the assumption that the pdf ridge to be extracted forms one *continuous* patch, which only applies to mixtures exhibiting a sufficiently dense set of Gaussians. Therefore, they convolve the Dirac distribution of the input point set with a Gaussian kernel, resulting in a mixture containing one Gaussian per point. However, such dense mixture representations are not very efficient, and for very large data become infeasible. Therefore, we would rather describe the data by a more sparse, compact mixture of larger, anisotropic Gaussians we obtain from EM or HEM. However, with increased compression of the mixture, its Gaussian components gain anisotropy, which quickly breaks the continuity of the pdf ridges, making ridge-based surface reconstruction unfit for such scenarios. In Chapter 6, we therefore develop a reconstruction technique that defines a surface over sparse, strongly compressed Gaussian mixtures, employing a pseudo-manifold parametrization over a triangulation of their Gaussian components.

2.4 Surface Reconstruction and Representation

Surface reconstruction from point clouds has been a major research topic for over 20 years. Berger et. al [BTS+14] recently gave a comprehensive overview over the current state of the art.

Implicit surface reconstruction methods rely on the intermediate estimation of a smooth implicit surface, from which an isocontour mesh can be extracted using the many variants of *marching cubes* [LC87] or *dual contouring* [JLSW02]. Implicit methods range from locally fitting tangent planes [HDD+92], using Radial Basis Functions [CBC+01], partition-of-unity blending [OBA+03], explicitly inferring point orientations for unorganized raw point clouds [MDGD+10], up to a binary inside/outside indicator function as in the widely used Poisson Reconstruction methods [KBH06, KH13]. Most of these methods assume that the surface can be oriented based on the input samples, or even that these surface samples come with estimated or measured normal vectors. Moreover, geometry may be inferred far away from the sampling if needed. Point Set Surfaces (PSS) [ABCO+01, ABCO+03, AK04] and their variants [AA04, GG07, AA09, OGG09, GAB12] aim at providing geometry processing primitives such as sampling, filtering and reconstruction, directly from unorganized point clouds, without defining an explicit topological space nor estimating a global volume of the input shape. Typically formulated through a *moving least squares* (MLS) optimization [LS81, Lev98, Lev03], these models exploit a space-projection procedure that often relies on normal vectors associated to each sample. This typically implies dense enough sample sets for inferring a reasonable orientation estimation, which may optionally be enriched with anisotropic support functions aligned to principal curvatures [AA06, WZK05] or rated regarding the local tangent-space fit quality w.r.t. the

likelihood of each sample [PMG04]. Unfortunately, even local PSS models become impractical when significantly increasing the sample count, and are typically sensitive to outliers due to their least-squares (L_2) formulation.

L_1-based reconstruction techniques have gained a lot of attention over the past years in many fields, as they are known to be less sensitive to the presence of outliers. For surface reconstruction, methods have been proposed that use recent advances in robust statistics [FCOS05, OGG09]. These methods are theoretically able to faithfully reconstruct a surface as long as there are less than 50% outliers (the breakdown point). Other methods perform a global minimization on the orientation of the input normals, using ideas from compressed sensing and sparse signal recovery [ASGCO10]. However, while these algorithms are of high quality, due to their global nature they are often extremely slow, do not scale well to large data or require special assumptions on the input data, e.g., assuming data consisting of several planar elements.

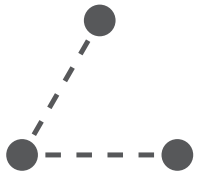
Interactive reconstruction has become an increasingly important topic since the introduction of real-time 3D acquisition devices. The seminal KinectFusion algorithm [IKH+11] builds a complete volumetric model of the environment by integrating range data over time into a 3d grid. The data can then be used for tracking and rendering by raycasting the implicit surface in the grid, but it only handles a limited amount of dynamics in the scene. For dynamic point sets, Zhou et al. proposed a method to build a GPU kD-tree in each frame to perform range queries for a per-frame computation of the k-nearest neighbors [ZHWG08], as well as an efficient way to interactively sample points into an octree structure and extract an implicit surface [ZGHG11]. To obtain high resolution and thus high reconstruction quality, the octree should tightly enclose the scene. However, for many real-world applications, this is not easily possible, especially considering large scanned data sets.

Point-based triangulation. A more combinatorial reconstruction strategy relies on the extraction of a 2-manifold set from 3d triangulations (e.g., Delaunay). Such methods come with guarantees [ACK01, DG06] but do not cope well with noisy input, and are robust only under certain sampling conditions, which are not always easy to meet with real-world data. A more flexible approach consists in greedily and locally generating connectivity among samples. For instance, lower-dimensional meshing techniques [GKS00, LP03] or front-propagation methods [CSD04] typically work well on dense enough data sets, are able to capture non-manifold regions, and are usually extremely fast to compute. In Chapter 6 we will show that such a front-propagation based triangulation method is well-suited for an efficient triangulation of points carrying local shape-descriptor information, such as anisotropic Gaussians.

Directly related to the problem of surface reconstruction is the question of how to represent the final reconstructed geometry in a suitable way. For example, in implicit surface reconstruction, a smooth shape is defined in a lossless manner by the entirety of the input points themselves. However, its evaluation requires a projection of individual

point samples to this surface, which is inefficient in general. On the other hand, a triangulation of such surface samples to a piecewise-linear mesh results in a simplified, easier-to-use representation, but only gives a lossy approximation of the original smooth surface. In Chapter 6, we triangulate a sparse set of anisotropic Gaussians sampling a surface in order to reconstruct its topology. This, however, results in a particularly coarse approximation of the surface.

Parametric surfaces allow defining smooth shapes over such coarse meshes by specifying smooth geometry functions that are parameterized over individual mesh faces. Besides the choice of the used geometry function defining surface patches over a local parametric domain, a fundamental problem is the continuous parametrization of the mesh itself. Hormann et al. [HPS08] give an overview of different classes of approaches. Different strategies for global parametrization have been developed, but produce singularities or require a segmentation into several easier-parameterizable patches for surfaces of higher genus. Grimm and Hughes [GH95] introduced the concept of *parametric pseudo-manifolds*, which allow defining smooth surfaces of arbitrary topology by blending atomic surface patches parameterized over individual partial mesh domains, so-called *charts*. Each parametric point on such a chart maps to a point on an abstract topological manifold, where bijective transition maps define the consistent registration of regions where the images of these charts overlap. One of the benefits of this concept is the clear decoupling of the topological and the geometric description of the surface. Existing approaches use different chart parameterizations, transition functions and geometry functions, thereby exhibiting different properties of continuity and smoothness. In their seminal paper, Grimm and Hughes constructed C^k surfaces with B-splines using different charts and transition functions for vertices, edges and faces. Later, unified charts over quadrangular sub-meshes have been used [NG00, CNPGVA02]. Ying and Zorin [YZ04] were the first to achieve C^∞-smooth, non-singular surfaces based on polynomial patches by using a simple star-shaped chart layout over quad complexes, which pose a significant advantage over widely used Catmull-Clark subdivision surfaces [CC78]. For triangular base meshes of arbitrary topology, Gu et al. [GHQ06, GHJ+08] proposed manifold spline surfaces based on an affine atlas, which, however, requires an explicit patching of regions around singular points in the mesh. Rational C^k-surfaces for triangle meshes have been proposed, using circular charts decomposed into different subcharts in overlap regions [VJK08, VJ09]. Siqueira et al. [SXG+09] constructed non-singular C^∞-surfaces for triangle meshes based on Bezier-patches, using much simpler charts over individual mesh umbrellas. Since we can easily replace the geometry function of these surfaces by any custom shape definition, this method is an ideal candidate for defining surfaces based on local probabilistic descriptors like Gaussian distributions, given that a manifold triangulation and the definition of a suitable geometry function based on such Gaussians is possible. In Chapter 6, we therefore construct manifold surfaces over a triangulation of Gaussian mixture components similar to Siqueira et al. [SXG+09], but define the geometry using a novel Gaussian interpolation technique.



Neighbors, neighbors, neighbors, neighbors
Have I got neighbors?

— Rolling Stones

CHAPTER 3

Interactive Screen-Space Triangulation

Figure 3.1: Possible surface visualizations of a raw 3d point set sampling a truck containing the main source of sugar for the work on this thesis. Left: Screen-aligned box splats with heuristic splat sizes. Right: Point-cloud rendering using a screen-space triangulation method.

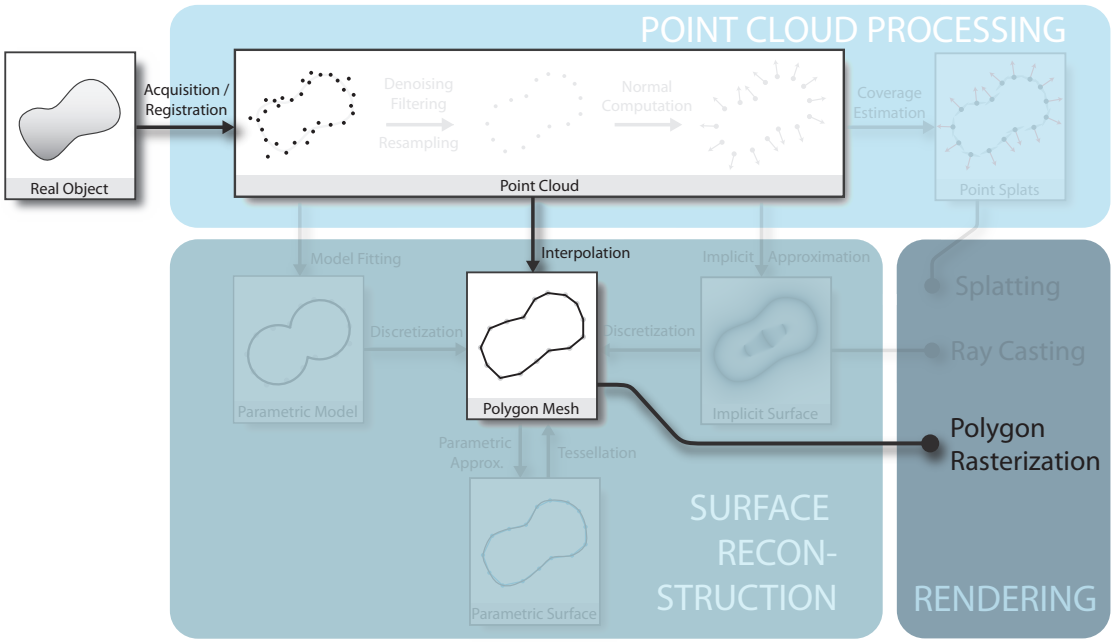


Figure 3.2: We present a real-time point-cloud visualization pipeline that performs a weak interpolation of raw input point attributes in screen space, allowing us to obtain a dense surface image using triangle rasterization.

3.1 Introduction

In this chapter, we aim at developing a rendering technique that produces an instant high-quality surface visualization for a dynamic stream of raw, unprocessed 3d point data. One of the predominant problems of current point-based rendering techniques is their dependency on preprocessed data like point radii or normal estimations. In many cases, however, neither the time nor the resources for such a preprocessing are available. For example, real-time acquisition devices are able to stream 3d point data at real-time frame rates, requiring processing and rendering methods to be performed on-the-fly without relying on offline preprocessing tasks. On the other hand, in the case of huge point clouds that already occupy a large amount of memory and need to be handled in an out-of-core visualization framework, the generation of additional attributes like normals or splats is often not intended, as this would require a significant memory overhead for such large datasets. Moreover, more sophisticated out-of-core spatial data structures allow the upload of subsets of the data at different density levels, which cannot always be easily addressed only by a single surface-coverage estimate per point. We therefore strive for a point-rendering method that only relies on raw 3d point data (optionally attributed by color information), and runs at interactive framerates independent from the source of a dynamic 3d point-cloud stream, e.g., a real-time scanner or an out-of-core point-rendering framework.

The most widely used approach for rendering dense surfaces based on raw points is the rasterization of individual point primitives of non-zero area, which have to be suitably scaled to provide sufficient overlap to produce a hole-free surface. Typical primitive shapes are disk or box splats, which have to be rendered as screen-aligned billboards if no preprocessed normal information that could be used for an appropriate alignment is available. While such rendering methods can already give a good visual impression of a closed surface, their results typically suffer from poor display quality due to the non-smooth composition of the individual splats, especially for surfaces viewed edge-on.

In order to address both the quality and the efficiency requirements stated above, we develop in this chapter a rendering technique that uses the graphics hardware to produce a high-quality reconstruction of the color, normal and depth image associated with the current camera view *directly in screen space*, which can then be used for further deferred shading. Instead of rendering screen-aligned splats as surface primitives, we first project the points onto the screen, and then compute a weak triangulation of neighboring point samples through efficient screen-space operations using the GPU rasterization pipeline. This results in individual triangle fans per point, which can be blended into the frame buffer using polygon rasterization (see Figure 3.2).

This method allows preserving small-scale details and silhouettes, while colors between available samples can be interpolated in order to produce a much smoother reconstruction than what would be possible with screen-aligned splats, as shown in the comparison in Figure 3.1. Our algorithm is especially suitable for in-situ high-quality visualization of big datasets like 3D-scans, making otherwise time-consuming preprocessing steps to reconstruct surface normals or point radii dispensable.

3.2 Challenges

When computing a local triangulation of a sparse set of points in screen space at render time, we have to address the following problems:

Efficient nearest-neighbor search. Any point-based reconstruction method that establishes a connectivity between local neighbors has to find the point samples within a certain neighborhood range. Most methods typically employ precomputed spatial acceleration structures like octrees or kd-trees for efficiently performing these neighbor queries, which we cannot afford if they have to be performed on-the-fly on dynamically changing data. Therefore, a key problem for our method is to address each point's nearest world-space neighbors for triangulation after projection to the screen. Since we have to expect a sparse, unstructured distribution of the points' projected texture locations, we also cannot rely on any particular lookup pattern. Our solution is to employ the screen-space grid as search structure provided by the graphics hardware, and to rasterize conservative splats covering the projected search region of each point to achieve these neighbor queries in parallel via per-fragment operations on the output textures.

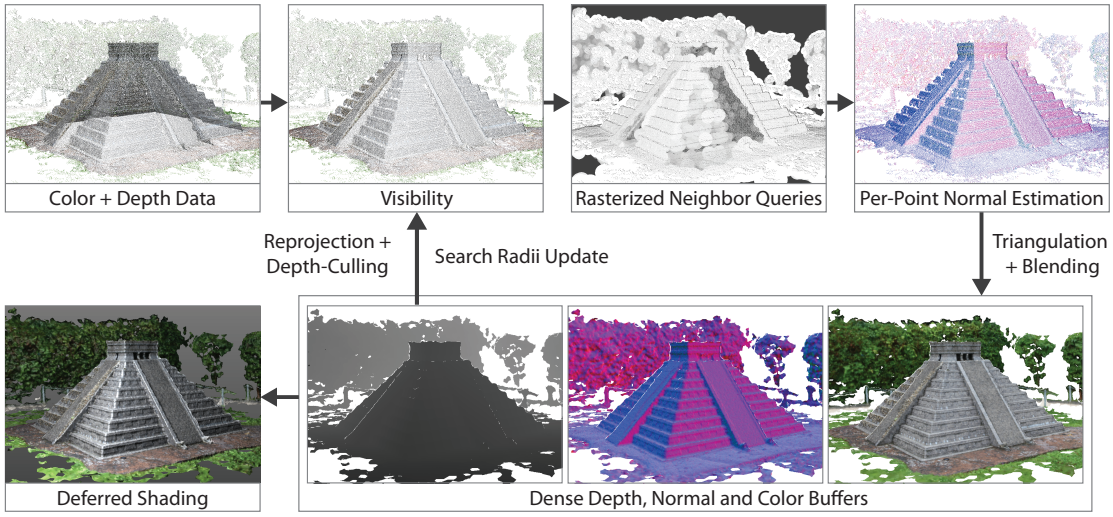


Figure 3.3: Overview of the rendering and feedback loop of screen-space triangulation.

Correct visibility. Another problem is that a screen-based surface reconstruction from a perspective image of a sparse point cloud requires correct visibility in order to avoid holes or incorrect rendering of actually occluded surface parts. However, determining correct visibility already requires a reconstructed surface or depth image for an accurate depth culling of occluded points. To tackle this problem, we make use of the *temporal coherence* between consecutive frames by reusing depth information from the previous frame via reprojection [SJW07]. Inaccurate visibility in parts of the scene that appear in a new frame due to rotation or disocclusions are thereby quickly resolved in subsequent frames.

3.3 Algorithm Overview

Our general approach is to first project a dynamic stream of unorganized, colored point data onto the screen and store their information in framebuffer-sized textures, and then start the reconstruction of a color image, a screen-space normal map and a depth map from there. The main steps of our method are outlined in Figure 3.3.

In the first step, the colored point-cloud data is rendered with z-buffering into screen-space textures, where they are stored at their projected pixel locations. This way, the front-most point mapped to a pixel will remain in the output texture and contribute to the following reconstruction. In the second step, we use the reprojected information from the depth image of the previous frame to perform depth culling on the currently projected points, thus obtaining a progressively better visibility information over time. At the same time, we also update the *neighbor search radii* of each individual point, which are also maintained in a screen-space texture. The next step performs a screen-space neighbor search for each visible point in the frame buffer. This step consists of two passes, after

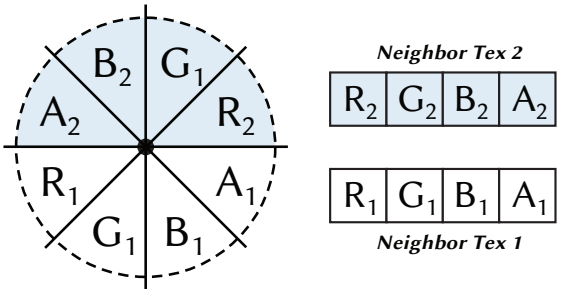


Figure 3.4: Storage layout for the nearest neighbors in the 8 surrounding screen-space sections of a point.

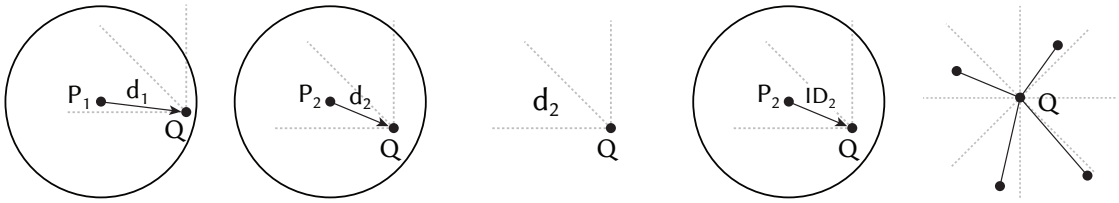
which we retrieve the screen coordinates of up to 8 nearest neighbors per point. Using the inherent connectivity information of these nearest neighbors, we can then perform a normal-estimation pass on the points. Finally, this connectivity is used to perform a local triangulation of each point's neighbors using the geometry shader, resulting in surface-covering triangle fans that interpolate the estimated normals and per-point color attributes stored in the screen-space textures. This results in three dense output buffers containing the color, depth, and the normal channels that can be used for further deferred shading.

In the following, we will describe individual steps of this rendering algorithm in more detail. Note that except for the first pass, which renders the point data into the frame buffer (object-pass), all remaining steps are screen-space passes that efficiently operate only on those pixels that contain a conservative subset of the visible point information. For convenience, we will refer to such pixels as "points".

3.4 Nearest neighbor search

In order to perform a suitable triangulation of the points in the framebuffer, each point has to be able to address its world-space nearest neighbor points. Collecting a list of the k nearest neighbors for each point in parallel would require expensive A-buffering operations [YHGT10], which we try to avoid here. Instead, we simplify the problem by searching for only *one* nearest neighbor in k individual subspaces around each point. In our implementation, we use 8 subspaces, storing up to 8 neighbors per point. To this end, we subdivide the screen-space region around each point into 8 sectors, as shown in Figure 3.4, and use two RGBA textures to store the ID of the framebuffer pixel that contains the respective nearest-neighbor point for each of this sections. Unique pixel IDs are obtained by linearizing the framebuffer grid.

We use a method similar to Van Kooten et al. [vKvdBT07], who rasterize screen-space splats in a particle system to distribute information from a point to all its neighbors within a given influence radius. In our technique, neighbor retrieval is performed via two passes, rasterizing for each point a splat that conservatively covers the projection

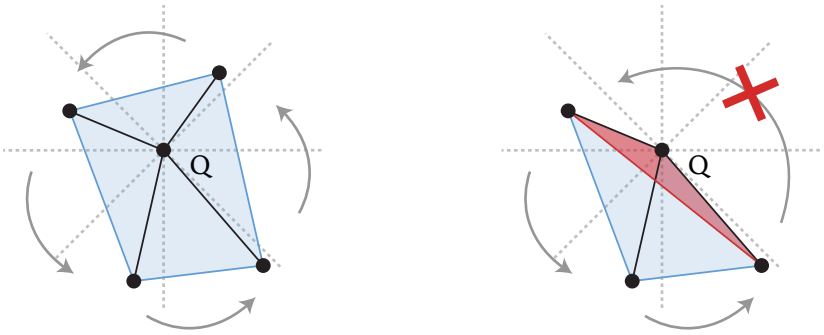


(a) (b) (c) (d) (e)

Figure 3.5: Two-pass neighbor search procedure. (a), (b) In the first pass, both P_1 and P_2 lie in the same section of Q, thus writing their distances d_1 and d_2 to the same texture channel. (c) Since $d_2 < d_1$, the distance d_2 remains in the section. (d) In the second pass, each point looks up the stored value in its respective channel. Since P_2 finds that the stored distance equals its own, it stores its ID at the respective texture channel of Q. (e) Finally, Q stores the nearest neighbor IDs for each section in its two neighbor textures.

of a point's world-space search ball. We use a vertex buffer object (VBO) containing the uv-coordinates of each point stored in the framebuffer, and pass it through a vertex shader to draw splats associated with each point. This way, we are able to exchange information between each point and the framebuffer pixels storing its neighbors within its search ball by performing respective texture writes in the fragment shader.

Figure 3.5 illustrates this two-pass nearest-neighbor computation for each section of a point Q that lies within the search ball of several points P_i. In both passes rendering the query splat of a point P_i, the fragment shader handling the pixel of Q first looks up its world-space position from the texture. If Q lies outside the world-space search radius of P_i, Q is discarded. Otherwise, it performs a write operation to the output texture channel that corresponds to the sector of Q in which P_i is located. In the first pass, the fragment shader writes the world-space distance $d_i = ||P_i - Q||$ (Fig. 3.5a and 3.5b), where we enable per-channel minimum blending using the OpenGL command `glBlendEquation(GL_MIN)`. This way, the rasterization hardware leaves the lowest distance value per channel in the render target texture, which corresponds to storing the point's distance to its nearest neighbor per section (Fig. 3.5c). In the second pass, all query splats are rasterized again. In the fragment shader, each P_i tests its world-space distance d_i against the minimum value stored in the respective section, and writes its pixel ID if it finds to be the nearest neighbor in this section (Fig. 3.5d). This time, with additive blending enabled, different neighbors of the same point can safely concurrently write their IDs to different texture channels by simply setting the shader output of the remaining channels to zero. After this second pass, each point holds the IDs of the nearest world-space neighbors within each screen-space sector of its search ball in the two neighbor textures (Fig. 3.5e). This mechanism can be referred to as a per-point radial visibility test. The image associated with the third step in Figure 3.3 visualizes the rasterization overdraw of the neighbor query splats for the pyramid scene, where the brightness of a pixel corresponds to the number of fragment shader invocations from neighboring query splats.



(a) (b)

Figure 3.6: (a) Example triangulation of a point Q and its neighbors. The geometry shader generates a triangle fan in circular order of the sections. (b) To avoid triangle overlaps, circularly consecutive neighbors spanning an angle $\geq \pi$ are not triangulated.

In the subsequent steps, each point in the screen is able to lookup its neighbors' IDs and map them back to their actual screen coordinates, thus allowing access to their neighbors' view-space depths, calculating their world-space positions by unprojection, and looking up any additional neighbor information like their colors or estimated surface normals.

3.5 Normal estimation

After we have determined the pixel IDs of the nearest neighbor points, we perform a fast screen-space normal-estimation pass. Contrary to the two previous neighbor search passes, this is done by a simple per-pixel operation on the framebuffer. Since the neighbor IDs of any point Q are stored in circular order in the neighbor textures, we are able to reproduce the circular sequence of a screen-aligned triangle fan, in which Q is the origin, as illustrated in Figure 3.6a. The normal associated with the point Q is then chosen as the average face normal of the triangles produced by this fan. To avoid slivers and back-flipped triangles, which would deteriorate the normal alignment, a triangle is only considered if the opening angle between the two consecutive neighbors is lower than π (Figure 3.6b). Note that since we are not able to reproduce any inside/outside information, the normals are always oriented towards the viewer.

3.6 Triangulation

In the last step, we invoke the geometry shader to triangulate and rasterize the fans spanned by the points and their surrounding neighbors. Again, we make use of the circular order of neighbors in the screen-space sections, and rasterize only triangles with a valid opening angle. In the geometry shader, the depth, color and normal attributes of each point are read from their screen-space texture location and passed on to the rasterization engine to be interpolated over the whole fan. The rasterized color and normal attributes of

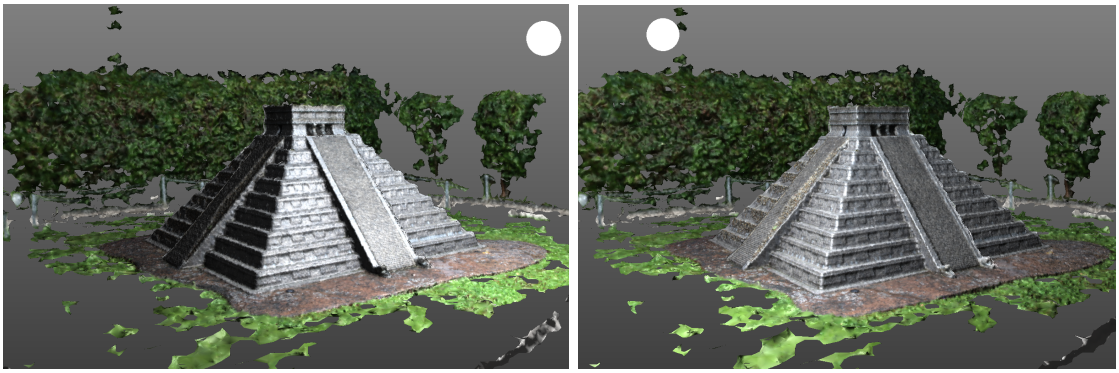


Figure 3.7: Deferred shaded phong illumination of the pyramid point cloud model under different light source positions (white spheres).

the rendered triangles are averaged by additive blending and a subsequent normalization pass of the values. For the depth channel, we need to use minimum blending to obtain a correct depth profile of the scene. However, the rasterization and interpolation of all three attributes can be performed in one pass by choosing a suitable output-texture storage layout and blending-function states. In our implementation, we write the 3-float normal and color values to the RGB channels of two target textures and the depth value into one of their alpha channels, and activate separate blending functions for the RGB and the alpha channels via the OpenGL `glBlendEquationSeparate` command to achieve a correct blending of these three attributes.

After this step, we obtain two screen-space textures containing the color, normal and depth information of the scene, which can be instantly used for any further deferred shading. Figure 3.7 shows a scene reconstructed with screen-space triangulation and shaded under a dynamic light source.

3.7 Search radii update

In order to retrieve all appropriate nearest neighbors P_i of a point Q, these neighbors have to use sufficiently large world-space search radii such that their query splats cover Q and thus allow them to "register" as neighbors by writing their IDs to the neighbor texture channels of Q (refer again to Figure 3.5). Determining the right search radius for each individual point is therefore essential for the performance of our screen-space triangulation technique.

The size of the search radii are constrained from two sides: Too small radii lead to points not being addressed by all appropriate neighbors, while too large radii result in too large query splats and thus in unnecessary rasterization overdraw, wasting computation time. The optimal radius r_i of a point P_i is thus the distance to the furthest point Q that has P_i as nearest neighbor in one of its sectors. To find this optimum for each point, we

perform an iterative adaptation that continuously optimizes the radii over the course of a few frames.

Our solution builds on the assumption that all points requiring P_i to register as neighbor are located within a finite vicinity around P_i and exhibit a unimodal distribution of distances to this point. Based on this reasoning, our approach is to constantly evaluate the distance f_i of the furthest point to which P_i registered as neighbor, and increase r_i until f_i no longer increases. In each frame, after we have performed the neighbor search step and are able to lookup the coordinates of each point's surrounding neighbors, we determine f_i by executing an additional GPU pass where each point Q writes for each of its stored neighbors P_i the distance $||Q - P_i||$ to the texture position of P_i. This is done by employing the geometry shader to lookup the neighbor IDs of each point and emitting a single vertex per neighbor that addresses their position in the screen. By activating maximum blending, each neighbor P_i then ends up with the distance f_i to the furthest point Q to which it is a neighbor.

To assess the development of these furthest-registration distances f_i for each point, we always maintain the two most recent values from the last two frames in two separate single-float textures. When projecting the points to the screen at the beginning of the next frame, the last search radius r_i, the last distance f_i and the second-most recent distance f_i' are used to update the search radius for the next frame by the following rule set, which is parameterized by a user-defined maximum connection distance r_{max}:

1. if $r_i = 0$, then P_i was not visible (occluded or outside the viewport) in the previous frame, and its search radius is initialized to $r_i = r_{max}/2$, and $f_i = f_i' = 0$.
2. else, if $f_i > f_i'$, the furthest-registration distance has increased, i.e., we previously found a new point to which P_i is a neighbor. In this case, r_i is further increased to search for more distant points by setting $r_i = r_i * \alpha$,, with $\alpha > 1$.
3. else, if $f_i = f_i'$, r_i is still increased as in 2 until either $r_i = r_{max}$, or reaching a threshold ratio r_i/f_i, where we assume that no farther point to register to will be found and roll back the search range to the last furthest-registration distance, i.e., $r_i = f_i + \epsilon$.

This results in a system of dynamic search radii that continuously strives for optimal neighborhood ranges in dynamic scenes and converges to a stable parameter state in consecutively static frames.

3.8 Results

3.8.1 Reconstruction Quality

In this section, we assess the quality of our direct point-cloud visualization technique under various aspects, and compare them to direct point rendering using box splats,

Figure 3.8: A scanned pillar point cloud rendered using screen-aligned box splats with heuristic density-based size [SZW09] (left), manually chosen optimal uniform splat size (center), and our screen-space triangulation method (right).

which is the most commonly used method to obtain dense images when no additional precomputed point attributes are available.

Texture quality. Figure 3.8 compares different renderings of an advertising pillar point-cloud model. We compare box splats with both per-point density-based splat sizes [SZW09] (left) and manually optimized globally uniform splat size (center) with our screen-space triangulation method (right). While a density-based heuristic point size aims at producing a conservative surface coverage, it is less suited for high-frequency textures due to non-smooth splat overlaps. Adjusting the splats manually to a minimal surface-covering size produces far better results, but is, however, not suitable for datasets exhibiting varying point densities and still shows some box artifacts. The result of our method is comparable to the manually adjusted result, but adaptively adjusts on the local point spacing and notably improves on image quality by omitting box artifacts, exhibiting a smoother reconstruction of texture.

Noisy data. The smooth blending of overlapping, world-space aligned triangles is especially beneficial in point clouds exhibiting strong geometric and color noise. An example is shown in Figure 3.9, which compares screen-aligned box splats with screen-space triangulation in a view of a catacomb dataset that has been scanned under difficult perspective and illumination conditions, resulting in significant color and geometric noise. Our method preserves salient texture features like the writings on the wall to the

(a) heuristic box splats (b) screen-space triangulation

Figure 3.9: Comparison of our approach and heuristic box-splatting a dataset exhibiting strong color and geometric noise. Notice the reconstruction of the fine writings on the wall.

(a) heuristic box splats (b) screen-space triangulation

Figure 3.10: A complex-geometry scan from a chandelier, rendered with with heuristic splat sizes and with screen-space triangulation.

upper left and lower right of the image, where screen-aligned box splats fail due to noise, incorrect splat orientation and non-smooth overlaps.

Silhouette quality. A major drawback of rendering box splats is the inherent appearance of staircase artifacts at object silhouettes, which becomes particularly apparent in the presence of fine geometric features like the strongly noisy chandelier shown in Figure 3.10, which generally poses a difficult case for any reconstruction algorithm. Since our technique only rasterizes valid triangles connecting exact world-space point locations along camera-oriented subdivisions of a point's the surrounding neighborhood, it is able to produce much more detailed silhouette contours. However, as in any method, noise and outliers can deteriorate the quality of the resulting contours as well.

Reconstructed normals. Essential for the quality of further deferred shading and illumination results is the quality of the reconstructed normal map. Figure 3.11 compares a box splatting of precomputed normal information of the Armadillo model point cloud

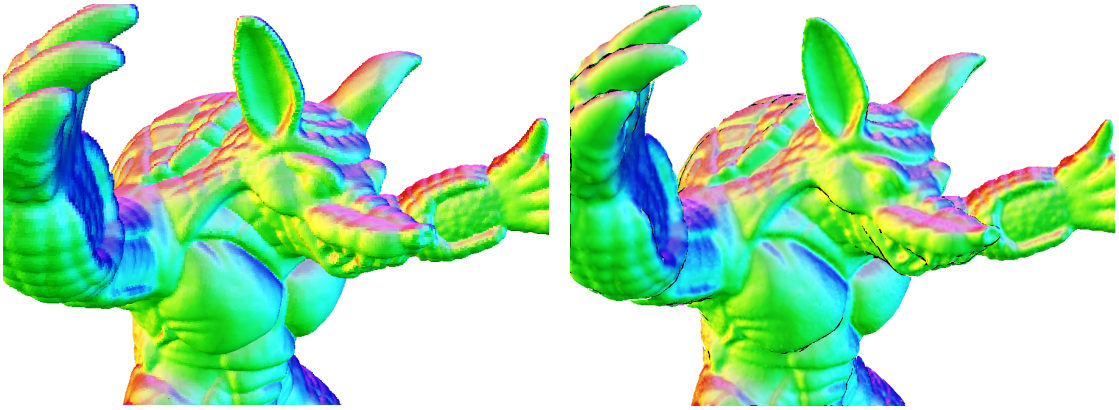


Figure 3.11: Comparison between preprocessed normals in a point splat rendering (left) and the normal map computed on-the-fly by screen-space triangulation (right).

to a dynamically screen-space triangulated normal map. The false-color comparison reveals that our method is able to faithfully reconstruct per-point normals in screen space at a quality indistinguishable to precomputed normals. However, similar to the color channel, the smooth blending of normals interpolated over a triangle fan results in a much smoother normal map than discretely overlapping box splats.

A disadvantage of this approach roots in the limited number of neighbors that can be used for the reconstruction of a local surface-covering triangle fan, as this basically constrains the bandwidth of the filter that is achieved through interpolating overlapping triangle fans. In point-cloud scenes exhibiting a noise level that is larger than the bandwidth achieved by 8 neighbors, we expect a noisy, not sufficiently filtered reconstruction. Figure 3.12 shows the normal maps resulting both from precomputed and from dynamically triangulated normals in a noisy scan of a cathedral. While our method produces much cleaner edges and silhouettes than box splats, we observe a subtle noise level in the resulting normal image on smooth surfaces, where a larger neighborhood would have had to be incorporated for the reconstruction.

3.8.2 Performance

We have tested the performance of our dynamic point-rendering algorithm on a platform with Intel Xeon X5550 2.67GHz CPU and GeForce GTX480 GPU with 1.5 GB video memory. The cathedral and the catacomb scenes are large datasets that are managed by an out-of-core point-streaming framework, dynamically streaming point-data nodes of a layered octree hierarchy to the GPU depending on the current view frustum. Figure 3.13 breaks down the average frame-time consumption of the individual shader passes of our algorithm for three different scenes. As expected, the general computation time of a frame depends on the size of the point cloud, or more concretely, the number of points stored in framebuffer textures and used for the reconstruction after projection. The triangulation pass, which invokes the geometry shader to rasterize the individual triangle

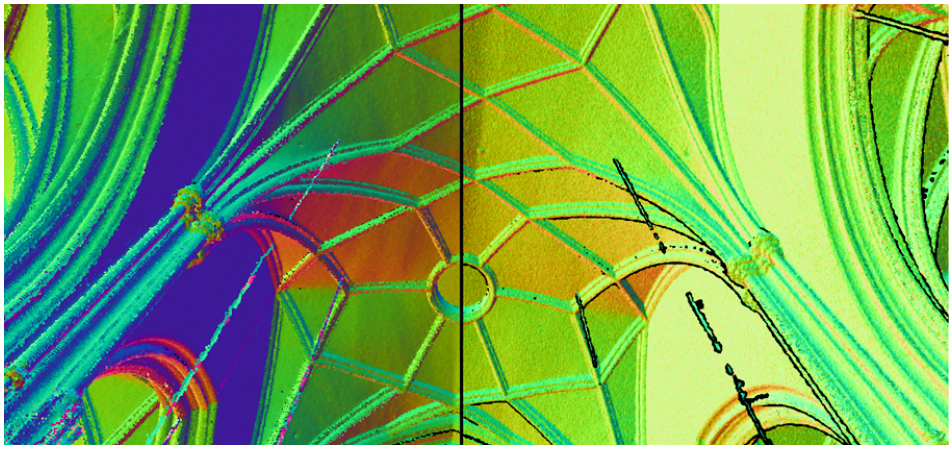


Figure 3.12: Comparison between precomputed normals (left) and screen-space triangulated normals (right) in point clouds with strong noise.

fans, consumes the largest part of the frame time. However, for huge datasets like the catacomb, where the screen is heavily saturated with pixels containing point information after projection, the two neighbor-search passes rasterizing the query splats consume the major part of the computation time. To assess the dependency of the performance on the screen resolution, we measured each scene at two different viewport sizes of about 40% difference. Characteristic of a screen-space algorithm, the graph indicates that larger viewports affect the effort spent on the rasterization operations used for neighbor queries and triangulation.

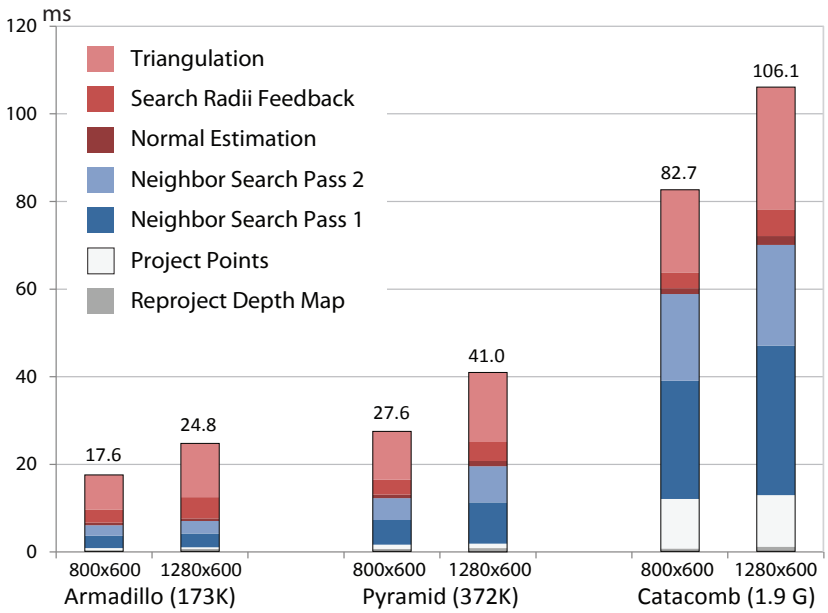


Figure 3.13: Time consumptions of the individual SST shader passes in ms for three scenes at different viewport sizes. Dataset sizes are given in parentheses.

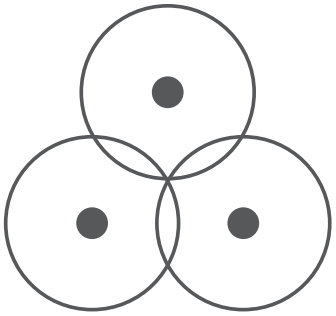
3.9 Conclusion and Limitations

In this chapter, we have developed an interactive rendering technique producing dense and smooth images from unprocessed point clouds given only their position and color information. Our method works entirely in screen space, where it finds the nearest world-space neighbors of each point, allowing for an on-the-fly normal estimation and a subsequent rasterization of local triangle fans interpolating the attributes of the neighboring points. Furthermore, we exploit temporal coherence between consecutive frames to speed up the reconstruction and obtain a constantly improving visibility information over time. The key contribution of this technique is a way of establishing communication between locally neighboring points in parallel by rasterizing query splats that cover the projection of a query ball in screen space and allow to exchange any kind of information between neighbors.

As discussed in Section 3.8.1, a major limitation of our rendering method is its limited maximum number of nearest neighbors, which constrains the filter bandwidth in scenes of strong noise. Arbitrarily increasing the number of subdivision sections in order to enlarge the number of neighbors in our framework is neither feasible nor practical, since first, this would require additional neighbor textures and heavily increase the workload on texture memory transfers for writing and reading neighbor IDs per point, and second, a triangulation of a circular order of neighbors can not be expected to produce desirable triangle-fan shapes and silhouette contours.

Another drawback of our neighbor-search strategy is the use of a screen-aligned subdivision pattern, which produces a different set of neighbors under different viewing angles. While this simplified approach allowed us to obtain a usable set of neighbors in only two neighbor-query passes, this view dependency can result in temporal flickering artifacts under camera motion.

In the next chapter, we will address these issues and develop an alternative, enhanced online reconstruction and rendering technique for dynamic unstructured point clouds that lifts the above-mentioned restrictions and is able to produce surfaces of even more improved smoothness.



Draw me a circle that's perfectly round
One curving line
Simple and fine.

— Barbara Joan Streisand

CHAPTER 4

Auto Splats

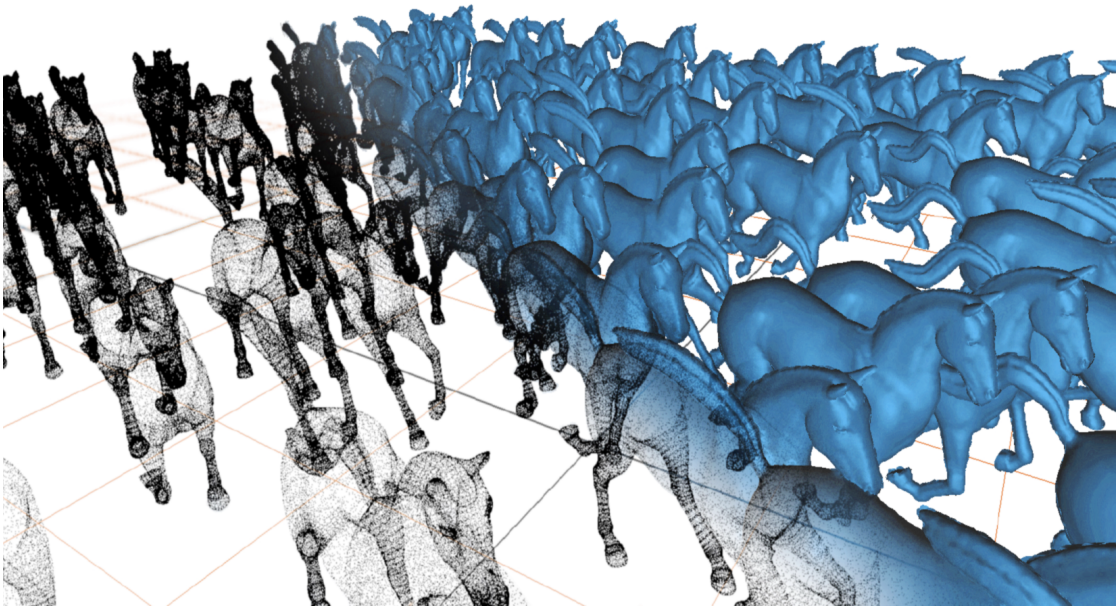


Figure 4.1: Visualization of a dynamic point-cloud scene constisting of 1 Million points. Raw 3d point data is streamed to the GPU, where our algorithm instantly reconstructs elliptical surface-aligned splats that enable a high-quality rendering and illumination. The complete frame is computed in 94 ms at a resolution of 1700×900 pixels.

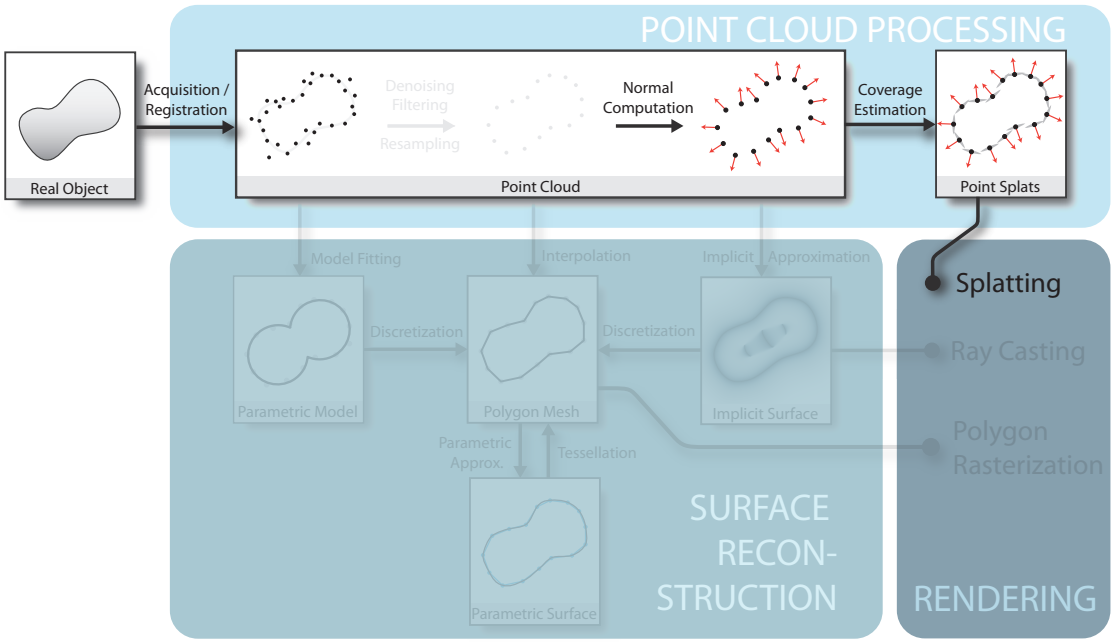


Figure 4.2: We develop a real-time reconstruction pipeline that performs the computation of normals and elliptical splat attributes for each point in screen space, enabling the instant application of high-quality splat-rendering techniques to unstructured points.

4.1 Introduction

The screen-space triangulation technique developed in the previous chapter is able to perform an instant parallel nearest-neighbor search of points in screen space, which is the key for obtaining a local reconstruction of a surface for high-quality rendering on the fly. However, its major limitations are the constrained number of nearest neighbors we are able to store per point, and the view dependency of the reconstruction, which can cause temporal instabilities of the triangulation under camera motion.

In this chapter, we lift these restrictions and develop a screen-space reconstruction technique that can work at arbitrary bandwidths, is largely view-independent (up to the projection of points to the screen), and in addition produces a much smoother surface. Instead of blending piecewise-linear triangle fans based on a view-dependent, spatially discretized neighborhood, we compute for each point correctly world-space-aligned surface normals and elliptical splats based on the true K-nearest neighbors of each point, without restriction on the size of K. These splats are used to perform elliptical weighted-average surfel splatting [BHZK05], which smoothly blends the individual splats using weight kernels of finite support. As these splats are automatically reconstructed in screen space at render-time, they have been given the name "Auto Splats". The resulting alternative computation pipeline of this approach is illustrated in Figure 4.2.

In its core, this new methods builds on the same fundamental technique of establishing

an information exchange between points in the screen through rasterization of query splats covering the projection of their neighborhood range. However, since in the previous chapter we have observed that these rasterized query splats become the major performance bottleneck for an increasing number of points, we significantly enhance their efficiency by a specific utilization the Z-Buffer capabilities of the graphics hardware.

The key technical contribution of this chapter is the execution of a parallel true KNN search, which determines a view-independent neighborhood for arbitrary bandwidths. The main insight is that in order to fit an elliptical splat into the neighborhood of a point, it is not necessary to explicitly store the individual neighbors themselves. Instead, we employ the previously developed neighbor-query rasterization technique to directly accumulate the necessary information from points within a particular radius that contains exactly K neighbors of a point. In fact, we use the same technique to compute this K-radius itself by performing an iterative range search.

In the following sections, we will describe this method and its implementation in detail, analyze its quality and performance, and assess its KNN computation speed against an online GPU kd-tree construction technique, which represents a comparable alternative for performing fast KNN queries on dynamic data.

4.2 Overview

The input to the algorithm is a set of n 3D points $S = \{x_i, y_i, z_i | i = 1..n\}$ containing the point positions of the current frame as well as optional per-point color data. In addition, a parameter k defines the number of neighbors to take into account for splat normal and radius estimation. This parameter defines a reconstruction bandwidth that adapts to the local point density, and represents a trade-off between feature preservation and noise reduction. Our method computes elliptical splats for each point, which can then be used for a weighted-average blending of the surface as well as the associated color data.

The complete computation pipeline consists of four major phases, as depicted in Figure 4.3. First, each point is projected to its 2D pixel position in the current output image, where its 3D position and color information is stored. During this projection, the occupied pixel positions are read back to a vertex buffer (VBO) using the GPU transform-feedback capabilities. In the subsequent passes, this buffer is used to address the points in the screen without having to reproject the whole point cloud again. In the next phase, each screen point determines its KNN-radius r_K, defining a minimal radial range containing exactly k nearest neighbors (Section 4.4). Based on this radius, we perform a splat fitting by computing a normal and a radius for each point (Section 4.5). Finally, the resulting splats are rendered using state-of-the-art GPU splatting techniques to produce the output image (Section 4.6). All computations are performed directly in screen space, utilizing the common graphics pipeline and programmable shaders to efficiently process points in parallel. In particular, the KNN search and normal and radius computation use a tailor-made parallel algorithm for an efficient communication between neighboring points in screen space, which will be described in the following.

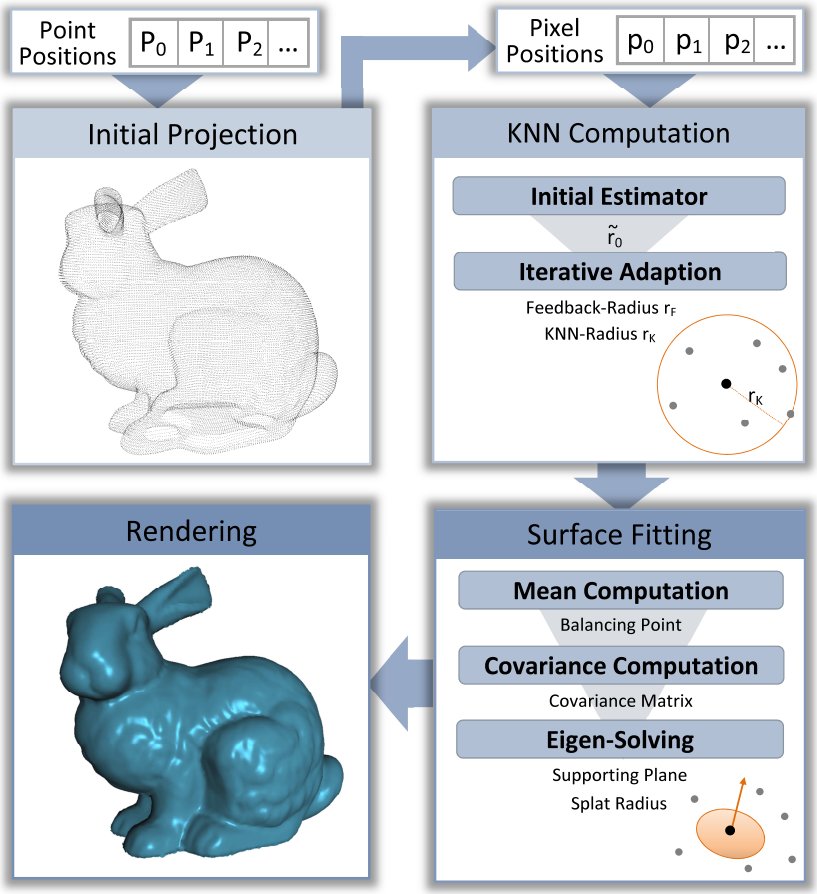


Figure 4.3: Overview of the main steps of the Auto-Splatting algorithm.

4.3 Parallel Splat Communication

A central building block of the KNN computation and surface estimation is establishing an information exchange of points in the screen through rasterization of query splats. We use these splats both to *distribute* information of a point to its neighbors as well as to *gather* information from neighbors within a certain radius. Using these two operation primitives allows us to iteratively compute the KNNs of each individual point.

Let P be a point in world space and r a radius defining a spherical neighborhood, such that any point within r can be called a *neighbor* of P. This neighborhood projects to a general ellipse on the view plane, which we approximate by a tightly covering 2D *neighborhood box*, as shown in Fig. 4.4. This box is guaranteed to contain the projections of all world-space neighbors of P. A naive way for P to gather information from its neighbors is to carry out texture lookups for all pixels covered by this box. However, this would be prohibitively expensive due to the large number of required lookups, especially for large radii. Instead, we *gather* information from points within r by performing an *indirect distribution* pass. In more detail, these two operations work as follows:

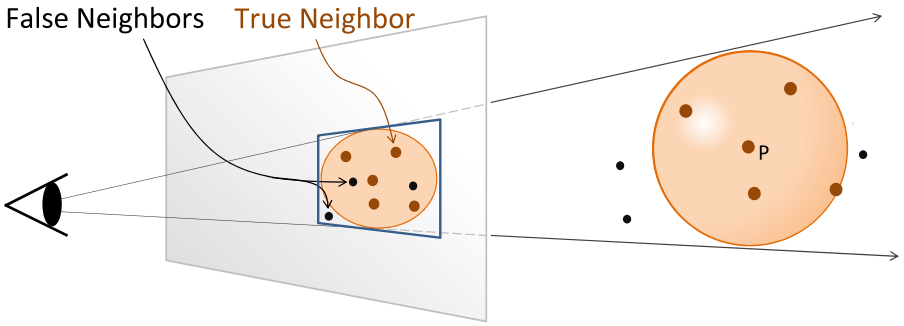


Figure 4.4: Projecting the sphere (P, r) to the view plane reduces the search space for the neighbors of P inside r to an elliptical region containing all points inside the frustum that is defined by the ellipse. An inside-outside test with the sphere on each point in this region yields the true neighbors of P.

Distribution. To pass information from P to all its neighbors within a radius r in parallel, we assign this information to the rendered box splat. For each pixel containing a point Q in this splat, we test whether $\|Q - P\| < r$ to determine whether Q is a true neighbor of P. If so, the assigned information is written to the respective pixel, otherwise, the pixel is discarded.

Gathering. In several phases of our pipeline, we need to gather information from all neighbors of P within r. To this end, each neighbor performs an indirect distribution to write its information to the pixel of P. Since this is done for all points in parallel, the radius of the sphere that defines the splat rasterized for a neighbor Q has to be the distance to the furthest point that requires feedback from Q. We call this distance the *feedback radius* r_f. As before, each point P covered by the feedback splat of Q has to test whether Q actually lies within its neighborhood radius r. If so, feedback information from Q is written to the pixel coordinates of P.

Although the process of gathering is thus just an indirect distribution pass, the major difference is that when performing an actual distribution, all neighbors within r are actually addressed, whereas for an indirect distribution within a feedback radius r_f, we only write to those points having the feedback point as neighbor (which could actually be just one of many points within r_f). To determine r_f for the neighbors of P, we perform a distribution pass that writes the world-space distance $\|Q - P\|$ to the pixel of each neighbor Q. Using the MAX-blending state of the rasterization pipeline, each point ends up with the distance r_f to the furthest point for which it serves as neighbor. Details on the usage of distribution and gathering for a parallel accumulation of neighbor information will be discussed in Section 4.4.2.

Z-Buffer Utilization. To minimize the number of fragment threads required for splat communication, we take advantage of the hardware's Early-Z culling ability, which is implemented in most modern GPUs. Within the screen region of a rasterized neighborhood

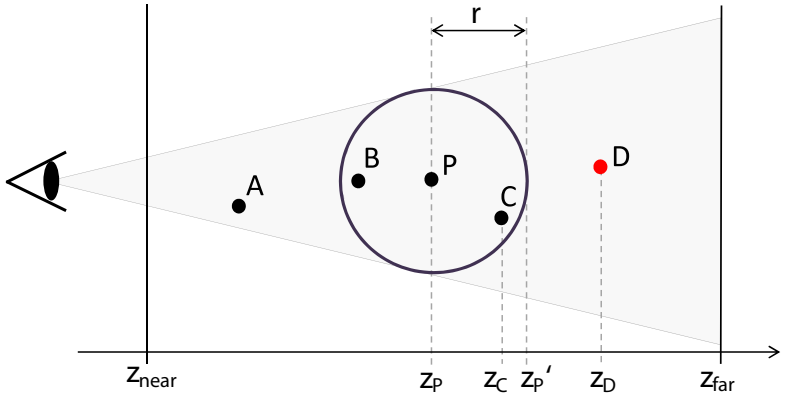


Figure 4.5: P communicates with its neighbors within a range r by rendering a neighborhood splat, which in screen space contains false positives A and D. By choosing a splat depth of z'_P, the depth test discards all splat fragments containing points with $z > z'_P$. Thus, D is Early-Z culled.

box, we only need the GPU to start a fragment thread for pixels that are occupied, i.e., those which actually contain a projected point. To achieve this, we create a depth mask of the projected points in the z-buffer at the initial projection stage. When rasterizing distribution splats, this depth profile causes empty pixels to fail the Early-Z test. This allows the graphics hardware to not launch fragment threads for entire blocks of empty pixels, which significantly speeds up the execution of these neighbor queries.

The remaining pixels in the neighborhood box contain both true neighbors (actually within world-space range) and false neighbors, as depicted in Figure 4.4. To further increase the efficiency, the depth buffer is set up in a way that allows the hardware to discard about 50% of false neighbors at Early-Z. The concept is illustrated in Figure 4.5: After the initial projection stage, the depth buffer contains the normalized depth footprint for all visible points, and the depth value 1 for all empty pixels. When drawing the neighborhood box for a given point P in the screen, we pass its fragments a biased depth $z'_P = f(z_P + r)$, where z_P is the view space z-coordinate of P, r is the neighborhood radius that defines its communication range, and f is a mapping from view-space depth to clip space depth. Setting the depth comparison function to GREATER lets the z-buffer cull all points at Early-Z that lie beyond the depth border represented by z'_P, while still maintaining Early-Z discards for empty pixels. Note that depth writes have to be disabled during the whole splat-communication phase of the algorithm to maintain the state of the initial depth footprint.

4.4 Neighborhood Computation

We define the k-neighborhood of a point P by the k-radius r_k that encloses the k nearest neighbors of P. Once this radius is found, we are able to perform distribute and gather

operations selectively on the KNNs in parallel. The k-radius r_k is found by an iterative range search in the non-continuous, monotonically increasing neighbor-count function $\sigma(r)$ over the radii r. Starting with an initial estimator $\tilde{r}_0$, the number of points within the current range of each iteration is determined and used to update the search range until a radius $\tilde{r}_i$ is found that contains $\sigma(\tilde{r}_i) = k$ points.

4.4.1 Initial Estimator

Having a good estimator for the initial range $\tilde{r}_0$ is critical for a fast convergence of the iterative search. Our strategy is to automatically determine $\tilde{r}_0$ individually for each point, which is especially important for scenes with spatially varying point densities. In contrast, the real-time kd-tree used by Zhou et al. [ZHWG08] to perform efficient range queries on dynamic data requires the user to manually set $\tilde{r}_0$ as a parameter, and needs this $\tilde{r}_0$ to be conservative , i.e., encompass r_k to obtain correct results.

We again determine $\tilde{r}_0$ directly in screen space: A low-resolution grid containing the projected points is laid over the screen, and the number of points within each grid cell is counted by performing an accumulation pass. For each frame, the grid resolution is chosen individually based on the current point count in such a way that on average, k screen points fall into one cell. For each grid cell, we choose $\tilde{r}_0$ based on its point density. Since we presume to operate on points that describe an object surface, we consider a two-dimensional distribution of the points within a cell. For A being a cell's pixel area and n being the number of points in that cell, the average cell area covered by k points is estimated by $A\frac{k}{n}$. The pixel radius r_{screen} of a circle covering the area of k points is therefore given by

$$r_{screen} = \sqrt{\frac{A\,k}{\pi\,n}}.$$

The initial world-space estimator $\tilde{r}_0$ of a point can then be derived by unprojecting r_{screen} based on the point's view-space depth. If the points in the cell describe one single unwrapped surface, this estimator roughly gives their expected KNN radii. On the other hand, if several depth layers are projected to the screen, like the front- and the back-facing part of a closed object, the number of cell points will be too high, causing the initial radius $\tilde{r}_0$ to be smaller than intended. This is acceptable, however, since the subsequent iterative range search will expand the radius in the next step.

4.4.2 Iterative Radius Search

The individual GPU passes performing the iterative k-radius search are depicted in Figure 4.6. For each point, we use its initial estimator $\tilde{r}_0$ as starting value for searching the target value r_k on the function $\sigma(r)$. Similar to a histogram-based KNN search in a kd-tree [ZHWG08], the search is performed using a multi-sampled bracketing approach that iteratively narrows the location of r_k by two bounding radii a and b, defining the lower and upper bound, respectively. In every iteration, each point's upper bound defines its

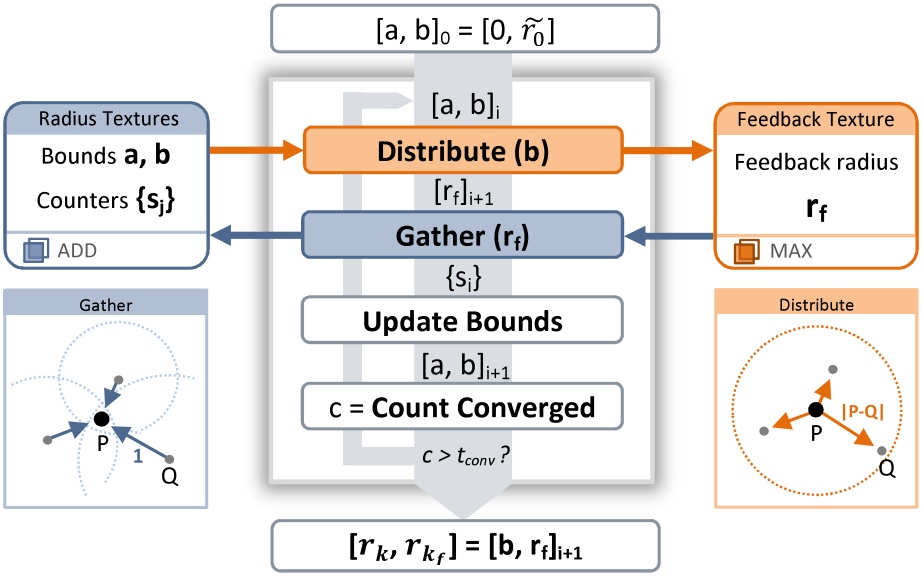


Figure 4.6: k-radius search algorithm and textures used for data writes and reads. The texture boxes show the stored main information and the used blend modes. After each iteration, the number c of converged points is counted. Iteration is finished if c exceeds a desired threshold t_{conv}.

current neighborhood. The number of neighbors σ_i within this neighborhood is queried by first performing a distribution pass to the neighbors to obtain the corresponding feedback radius r_{f_i}. A following gathering pass with r_{f_i}-sized feedback splats then accumulates a counter from each neighbor, yielding the current neighbor count σ_i. Instead of taking just one neighbor-count function sample σ_i at b, multiple samples $\{s_j | j = 1..m\}$ are taken at m regular steps between a and b. The total number of neighbors σ_i is therefore represented by s_m. To query multiple samples, m feedback counters are accumulated in each gathering pass, stored in several target texture channels in the radius textures (we use $m = 4$ in our implementation). Multisampling results in faster convergence since in each iteration it significantly raises the chance to find r_k and allows for a much tighter narrowing of the bounds. Based on the current bounds a_i, b_i and the counter samples $\{s_j\}$, the adapted bounds a_{i+1}, b_{i+1} for the next iteration are then computed. This adaptation occurs in two phases:

Expanding. As long as $\sigma_i < k$ holds, b is enlarged. The new upper radius bound b is chosen by extrapolating the current neighbor count σ_i to k assuming a constant two-dimensional point density. This linear relation between surface area and point count yields the radius increase factor

$$\alpha = \sqrt{\frac{k}{\sigma_i}}.$$

Bracketing. For a point count $\sigma_i > k$, a and b are iteratively narrowed until a radius with corresponding neighbor count sample $s_j = k$ is found. Bracketing is necessary to ensure a view-independent reconstruction. Although a faster approach would be an approximate nearest neighbor search that stops after the expanding stage and uses the resulting neighbor count $\kappa \geq k$ for reconstruction, the view-dependent estimation of $\tilde{r}_0$ based on an image-space point distribution would lead to a different κ and thus a different reconstruction under different views, resulting in temporal flickering artifacts under camera movement.

Implementation Details. We use a single feedback texture T_F and two radius textures $T_{R1,2}$ to store all required data. Besides the texture data shown in Figure 4.6, additional data is stored and ping-ponged between T_F and T_R, e.g., the current radius bounds a and b. Using separate blend functions for the RGB and the alpha channel, we achieve accumulating $\{s_j\}$, MAX-blending r_f, and passing along the additional data at the same time. To reduce fragment writes to a minimum, we do not actually feed back the counters from every neighbor within b, but only from those located within the delta region between a and b. To obtain the required counter samples $\{s_j\}$, we also store the last neighbor count at a and add it to the accumulated counters. If a point is already converged at the beginning of a further distribution pass, a 1-pixel sized splat is drawn to pass along the converged k-radius result at its own texture location until iteration is finished. Similarly, if a point that accumulated a zero r_{f_i} is about to feed back, a 1-pixel splat is required to pass along the point's data. The number of converged points is efficiently counted using GPU occlusion queries by looking up each point's current data in the radius textures and emitting a fragment if it is found to be converged.

4.4.3 Robustness

This section discusses some robustness issues that can arise due to noise, outliers and the information loss that occurs when projecting the data to a reduced set of points on the screen.

Outliers. Points in the framebuffer that have no neighbors in their immediate neighborhood can appear due to outliers in the point data set or because their neighbors are occluded by closer points in the depth buffer. In the expanding phase of the iterative search for r_k (Section 4.4.2), such points would continuously increase their search radius $\tilde{r}$ without finding any neighbor. This would lead to huge screen splats when projecting the search sphere onto the framebuffer, which could significantly impact performance. In our approach, the classification of such points is generally undecidable, since during iterative search we cannot distinguish between outliers with no real neighbors and points belonging to a coarsely sampled surface whose neighbors have just not been found yet. To reduce visual artifacts produced by outliers, we discard points with no neighbors after a certain number e_0 of expanding iterations. In our test scenes we found that $e_0 = 3 \sim 4$ is sufficient for a good reconstruction.

Small Point Groups. Outlying point groups containing $\kappa < k$ points represent a similar problem as single outlier points. To prevent our system from expanding the search radii up to k neighbors by bridging large gaps with no points, we further constrain the radius expansion. In each search iteration, the distance d_{max} of the furthest current neighbor is tracked. If the circular area defined by the expanding search radius $\tilde{r}_i$ grows by a certain factor λ without finding a new neighbor, expansion is aborted and the radius r_k is clamped to the reduced κ-neighborhood $r_\kappa = d_{max}$. In our scenes we used $\lambda = 4$ to cover surfaces of moderately irregular point distribution while avoiding too large gap-bridging splats.

4.5 Surface Fitting

The supporting plane of the splat attached to a point P is computed by fitting a plane π to the set of points $S = \{x_i | i = 1 \dots k\}$ in a local neighborhood of P by using linear regression [HDD+92]. A common method to find the parameters of π in the form $\pi : \ n \cdot (x - \bar{x}) = 0$ is to compute $\bar{x}$ as the mean

$$\bar{x} = \frac{1}{k} \sum_i x_i \tag{4.1}$$

of the point set S, and n as the eigenvector to the lowest eigenvalue of the scaled covariance matrix

$$cov(S) = \sum_i (x_i - \bar{x})(x_i - \bar{x})^T. \tag{4.2}$$

With the KNN radius r_k at hand, this computation can be carried out in three steps in our system (see Figure 4.3):

Mean Accumulation. First we perform a *gathering pass* that accumulates the mean $\bar{x}$ of the points in the neighborhood of P by ADD-blending each neighbor's world-space position as well as the counter value 1 for counting the number of accumulated values (Equation 4.1). The latter is necessary since we cannot be completely sure that each point has found exactly k neighbors in the KNN-radius computation pass before (see also Section 4.4.3).

Covariance Accumulation. In a second gathering pass, we accumulate the terms required for summing the covariance matrix from all neighbors (Equation 4.2). Each neighbor contributes the symmetric matrix $(x-\bar{x})(x-\bar{x})^T$, where the mean $\bar{x}$ is calculated by dividing the accumulated position by the counter value obtained in the previous pass. Since the covariance matrix is symmetric, it is sufficient to accumulate only the 6 values of its upper triangle matrix, which can be stored compactly within two render-target textures.

Eigensolving. A final per-pixel pass reads the covariance values and computes the eigenvector to the smallest eigenvalue using a standard eigensolving procedure [Ebe11] in a fragment shader program. This eigenvector defines the non-oriented normal of the supporting plane of the splat. Since we do not intend to consistently reconstruct the complete surface, but only the visible parts of the point cloud required for rendering, it suffices to orient the normals towards the camera. We can also render elliptical splats by additionally computing the remaining two eigenvectors, which represent the minor and major axes of the ellipse. The root of the fraction of their two eigenvalues is used as the proportion of their respective lengths [Paj03].

To determine the splat radius, we use a quick estimator based on the average area coverage of the points in the local neighborhood. At k neighbors, the radius of a circle enclosing the average area covered by each neighbor is defined by $\overline{r} = \sqrt{\frac{{r_k}^2}{k}}$. We choose the splat radius to be the average distance between neighboring points, which we approximate by $r_{splat} = 2\overline{r}$. For elliptical splats, this value is used for the length of the semi-minor axis.

4.6 Auto Splat Rendering

After we have computed the necessary per-point normals and splat radii, we use them to perform high-quality surface splatting [BHZK05] to render the final surface. This method employs three passes: An initial *visibility* pass producing a depth map, an *attribute* pass using this depth map for proper occlusion culling and front-surface attribute accumulation, and a final *normalization and shading* pass applying deferred shading. In the following, we describe two extensions to the technique which improve the quality and speed of rendering reconstructed splats.

4.6.1 Depth Refinement

Our reconstruction started with drawing all points as one-pixel sized point primitives. The resulting image can contain regions of a dense distribution of points sampling a foreground surface, while still exhibiting some holes where pixels from background surfaces are visible. In such cases, most of the actual nearest neighbors of these background pixels will be occluded by foreground pixels, leading to large neighborhoods during KNN estimation, and thus to large reconstructed splats. This can lead to artifacts when they extend beyond the silhouette of the foreground surface in the final rendering. Because these points appear mostly in back-facing or occluded regions, we can get rid of most of these artifacts by using vertex-based occlusion culling against the initial depth map. However, since the depth map from the first visibility pass is rendered *with* those incorrect splats, the visibility information in the depth map might be corrupt and we would miss a number of surface points for rendering.

Therefore, we extend the usual surface splatting pipeline by an additional depth buffer *refinement* stage that produces an improved depth map without artifacts. This is done by two additional depth passes. First, all points in the screen are culled against the

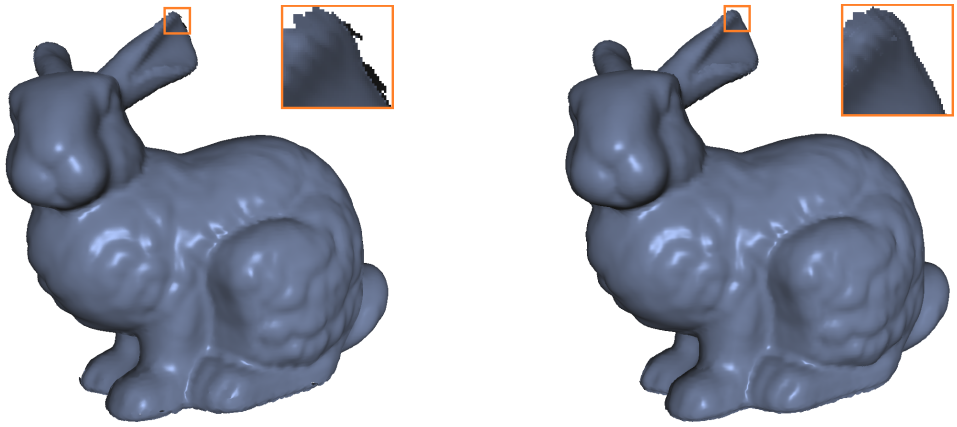


Figure 4.7: Comparison of a bunny model rendered with Auto Splats (left) and with precomputed normals using the same normal and radius estimation procedure (right).

initial coarse depth map to render a new, mostly artifact-free depth map. Then, we cull the points against this second map to remove possible holes and obtain a final refined depth map, which is used as input for the attribute pass.

4.6.2 Grid Culling

In large scenes of high depth complexity we often face the situation that due to perspective projection, the screen contains only a sparsely distributed set of visible foreground surface points while being densely covered by points from an actually occluded background. These cases can become very inefficient for our method, as we would spend most of the frame time on reconstructing background splats that will not be visible anyway. To improve efficiency in such large point clouds, we use an optional strategy similar to depth peeling. We first apply an approximate culling technique based on the low-resolution grid we have used to accumulate point densities for the initial KNN radius estimation in Section 4.4.1. While accumulating point counts in this grid, we also determine the depth d_i of the nearest point per grid cell i. For each cell, a culling plane is then defined at depth $d'_i = d_i + s_i$ to set all cell points occluded by this plane, i.e., with view-space depth $z_P > d'_i$, to *passive* state. Here, s_i represents the unprojected side length of screen cell i in world space at depth d_i. Splat reconstruction (KNN search, fitting, visibility) is then performed only for the remaining *active* points. However, the passive point set is still used for communication with the active points, i.e., they are still involved in the gathering passes of the KNN search and splat fitting to maintain a correct splat reconstruction of the active set.

This approach provides a selective splat reconstruction of the majority of the visible points on the screen, while skipping the unnecessary reconstruction of the bulk of actually occluded points in large scenes of dense screen coverage, which can significantly reduce the computation time. However, this approximate grid-culling technique will generally

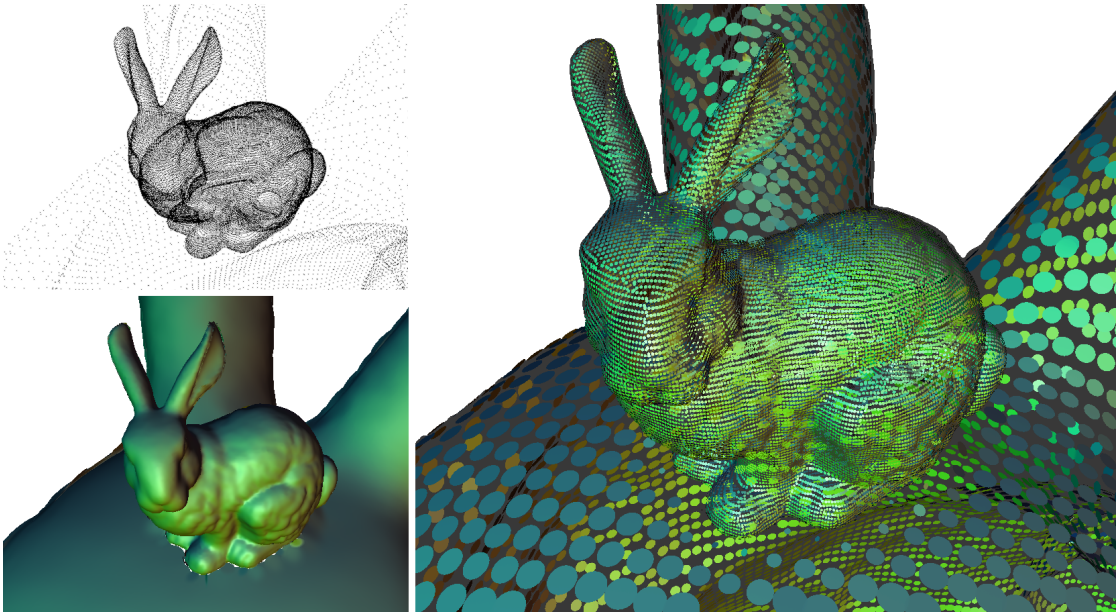


Figure 4.8: Reconstruction quality in a scene exhibiting large differences in point density, here showing a Stanford Bunny sitting on the head of another, larger Bunny (left top). Surface-aligned splats can be computed for both the large-scale and the small-scale model (left bottom). In the right image, splats were scaled down for better visibility.

classify a minor amount of visible points as culled, especially near silhouettes, where surfaces are viewed at grazing angles. Therefore, we need to apply a second reconstruction phase per frame to compute the splats for the remaining points. To this end, the depth map obtained from the reconstruction of the initial active set is used to perform a much more accurate culling of passive points, allowing us to quickly reconstruct the splats of the remaining active points. For very large point clouds, the minor overhead incurred by this additional reconstruction stage is easily outweighed by the reduced workload in the first pass. We have observed a speed-up of up to 60% for our larger test scenes.

4.7 Results and Discussion

4.7.1 Reconstruction Quality

Figure 4.7 compares the reconstruction quality of Auto Splats with preprocessed normals that were computed using the same neighborhood size and normal and radius estimation procedure as used in our algorithm. Despite some minor artifacts due to information loss at silhouettes, we observe a similar visualization. Since we are especially interested in the algorithm's behavior in scenes with point sets of strongly varying sample densities, we placed two bunny models with a large difference in scale into the same scene, shown in

Figure 4.9: A huge laser scan of a cathedral (470M points) rendered by an out-of-core point renderer with Auto Splats. The renderer streams an amount of ∼10M points to the GPU each frame.

Figure 4.8. Note that we reduced the splat radii in this image for a better visualization of the reconstructed splats. Our algorithm correctly adapts the splat sizes to the local point densities in world space, without relying on user input as in the method of Zhou et al. [ZHWG08], where an accurate KNN computation and thus an artifact-free rendering can only be performed if the user-specified initial radius estimator $\tilde{r}_0$ is chosen conservatively large enough for the entire scene, which is very inefficient for regions of high point densities.

4.7.2 Performance

The performance of our system has been profiled for four different scenes (Figures 4.7, 4.13, 4.1 and 4.9) exhibiting different characteristics in the geometry and density of the points. All measurements have been taken at a resolution of 1760x900 using a GeForce GTX580 Graphics Card with 1536MB VRAM and an Intel i7-930 CPU with 2.8 GHz. Note that the Imperia statue shown in Figure 4.13 was rendered in portrait format containing the full model. Figure 4.10 shows the required rendering times for the test scenes at different neighbor counts and decomposes the frame computation times according to the amount of time spent in each pass. The main part of the computation time is spent in the KNN search stage. For the shown test scenes, our KNN search requires about 7–10 iterations to converge for 99,9% of the points at $k = 10$. Moreover, as depicted in Figure 4.11, we observe a characteristic convergence graph that is common to all our test scenes, independent of the number of points. It can be seeen that in general, 6–7 iterations already provide a good quality.

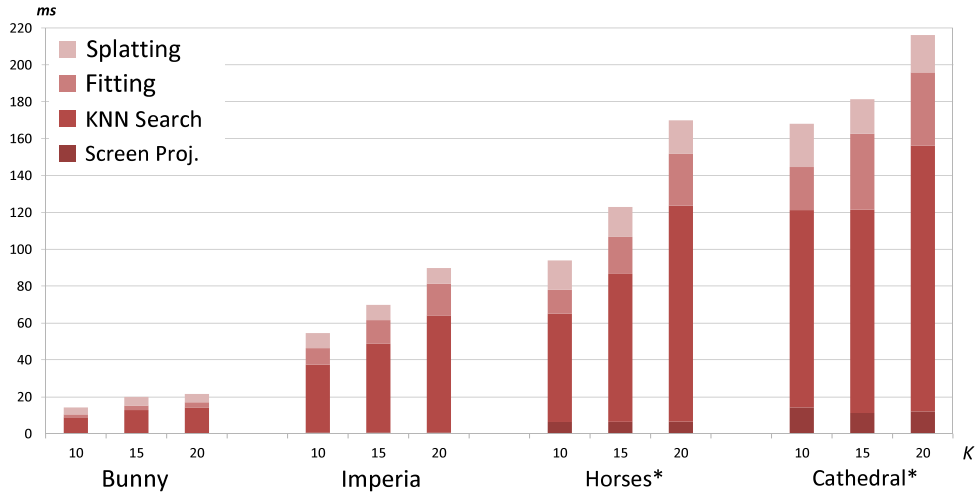


Figure 4.10: Performance decomposition of the computation pipeline for different test scenes and neighborhood sizes K at 99.9% convergence. The majority of the frame time is consumed by the KNN search. Plane fitting takes about the same time as the final splatting stage. The horses and the cathedral scene were drawn using grid culling.

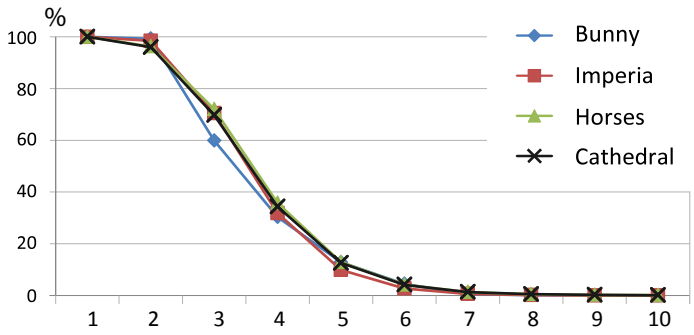


Figure 4.11: Convergence behaviour of the k-radius search. The y-axis denotes the percentage of unconverged points before each iteration.

We also compare the KNN-search performance of our method to using a dynamically constructed GPU kd-tree [ZHWG08], which also allows reconstructing point-based models on-the-fly. We reimplemented this method and analyzed its performance characteristics. Since tree build-up parameters have not been reported by the authors, we provide a best-case comparison for this alternative method. We profiled the tree build-up and k-radius search using several parameter combinations (including the user-specified initial range $\tilde{r}_0$) for our test scenes. As suggested by the authors, we used a histogram resolution of $n_{hist} = 32$ and $n_{iter} = 2$ range search iterations. We picked the lowest achieved timings

	Points	Pixels	kdTree	AS	Speedup	
Scene	#	#	ms	ms	p.pt.	**ALL**
Bunny	36K	34K	14	9	1,47	**1,56**
Imperia	546K	291K	95	37	1,37	**2,57**
Horses	1M	690K	169	117	1,00	**1,44**
Horses*	1M	690K	169	71	1,64	**2,38**
Cathedral	10M	1.3M	-	264	-	-
Cathedral*	10M	1.3M	-	123	-	-

Table 4.1: k-radius search times in ms at $k = 10$ achieved with Auto Splats compared to a GPU kd-tree [ZHWG08] with supposed preknown ideal parameters. The horse and the cathedral scene were measured both with* and without grid culling. The kd-tree times represent time for tree build-up plus k-radius search. Speedups are listed per element (p.pt), i.e., per point or pixel, and overall (all), i.e., taking into account the savings of handling only visible points in Auto Splats.

that came reasonably close to the k-radius accuracy of 99,9% we use in our scenes and compared them to the KNN-search timings of our Auto Splatting system (Table 4.1). Note that since we assume dynamic scenes, the kd-tree timings include the time spent on the tree build-up in each frame. In all scenes, our system outperforms the GPU kd-tree variant. On the one hand, this is explained by the fact that our system performs the search only on the reduced set of visible points. However, even when assuming the same number of points by comparing the average time spent on the KNN-query per point, Auto Splats provide a speedup over using GPU kd-trees (see the "p.pt." column in the table). Furthermore, note that the GPU kd-tree algorithm was not able to handle larger scenes like the Cathedral.

4.7.3 Applications

Real-Time Scan Visualization. One application where our technique is especially useful is the setup of real-world scanning sessions of static and dynamic content, possibly using multiple scanners. We simulated such a setup, as depicted in Figure 4.12. A dynamic object is scanned by up to four scanners, each providing a registered 3D point cloud (assuming mutual scanner registration). In addition, each scanner simulates a different amount of Gaussian noise. The auto-splatting technique allows us to display the scanned surfaces with high quality in real time. This enables positioning the scanners for optimizing surface coverage and minimizing occlusions in a scene, for example, during a film sequence. Furthermore, we can apply helpful visualization techniques, like the size of the K-neighborhood as shown in Figure 4.12, where the red end of the false-color spectrum suggests too sparse sampling. Note that for such scenarios, we can turn off the outlier removal of our system to not distort the result. In the future, real-time scanning devices can be of increasing importance for obtaining fast but high-quality previews for film, VR and AR applications.

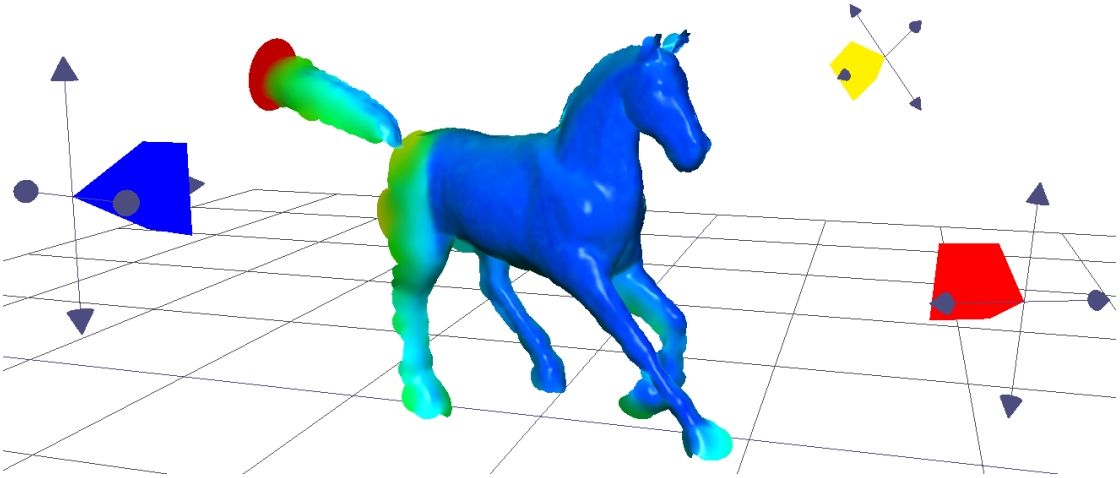


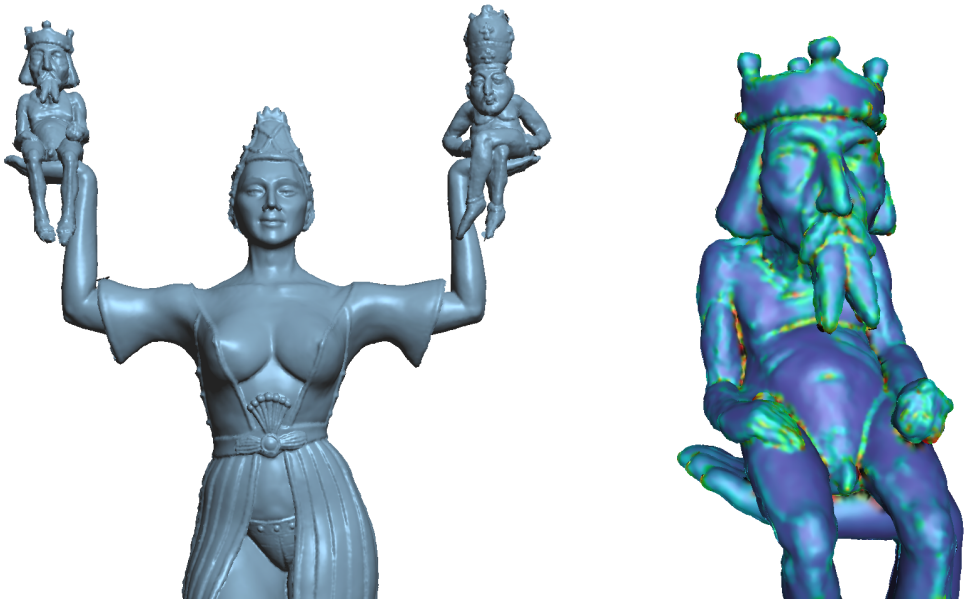
Figure 4.12: A dynamic object scanned by three scanners.

Normal and Curvature Estimation Preview. Offline algorithms performing normal or curvature estimation on massive point datasets can require up to several hours of processing time. However, the reconstruction quality of the hole dataset is not known in advance and non-optimally chosen parameters that could require a recomputation are often only recognized after the processing is finished. Our algorithm can be used to provide an instant preview of the reconstruction of different parts of a data set by an interactive walk-through. A user might wish to test different parameters for the k neighborhood to find a smooth but still feature-preserving optimum, or wants to analyze whether a certain parameter choice leads to uneven quality among the points. See Figure 4.13b for an example of an instant visualization of curvatures.

Modeling Applications. Applications that allow the user to modify a point cloud cannot rely on a lengthy preprocessing phase for normal-vector estimation. An example is an application that allows archaeologists to modify a scene to experiment with different reconstructions of an archaeological site.

4.7.4 Discussion and Limitations

The described method to locally reconstruct surfaces based on the points sampled to the screen has many advantages. For example, there are practically no parameters that influence the reconstruction quality (except for the neighborhood k), making the system readily useful for many applications. On the other hand, because each pixel only stores the front-most point, a number of points get lost. Generally, this is intentional, especially in large or dense point clouds with high pixel overdraw where we only want to perform computation on the the potentially small fraction of visible points in the front. However, not storing a point can become a problem if either its splat would still contribute to



(a) (b)

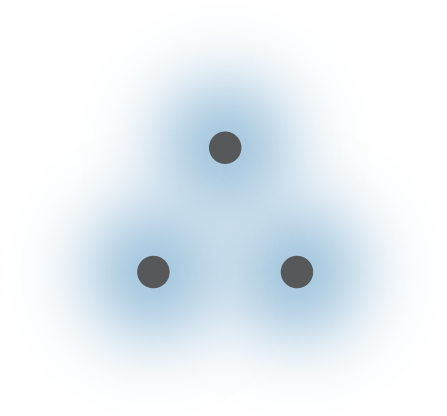
Figure 4.13: (a) Auto-Splatted image of a range scan of Imperia. (b) Closeup on a part of the statue, visualizing the curvature estimates for the points. Like normals and splat radii, curvature is dynamically computed from the KNNs provided by our algorithmn.

the final rendering, or it would contribute to the normal direction of a visible splat in its neighborhood. This can happen, for example, at object silhouettes, where spatially neighboring points can get rasterized to the same image pixel, leading to misaligned splats. Another restriction is the inability of determining a correct splat orientation, as this would require sequential or global computations on the point set that are too costly for a real-time setup, for example, computing a minimum spanning tree in the weighted K-neighborhood graph [HDD+92]. This can lead to wrongly illuminated splats, mostly at silhouettes.

4.8 Summary

In this chapter, we have developed an improved screen-space algorithm for producing interactive high-quality visualizations from dynamic unstructured point clouds by performing an on-the-fly splat reconstruction that allows for rendering smooth surfaces. The method can be applied to point clouds from any source, including dynamic data streamed from real-time scanners or spatial nodes of point data streamed from an out-of-core rendering system. Our algorithm uses the frame buffer as search data structure for the necessary nearest-neighbor computations, which comes with the advantage that we already obtain a reduced subset of the possibly huge amount of points in the scene for reconstruction. In contrast to the screen-space triangulation approach presented in Chapter 3, this algo-

rithm provides much smoother surfaces, can work on arbitrary reconstruction bandwidths, performs an actual true KNN search and is thus view-independent. It further introduces several performance optimizations by utilizing the hardware capabilities and reducing the reconstruction workload to a necessary minimum. Nevertheless, since the true KNN search requires several range search iterations to converge, its performance statistics show that these quality improvements require a generally higher computational effort. Still, we have shown that in each of our tests cases, our parallel KNN search technique outperforms alternative dynamic KNN search techniques like the dynamically constructed GPU kd-tree, even if the parameters required for the kd-tree are chosen favorably. Our method, on the other hand, does not require manual parameter tuning, and even works for very large scenes.



Basically, Probability
says that fate's gonna side with me.
It's been so long
on my shelf.

— Sugababes

CHAPTER 5

Continuous Locally Optimal Projection

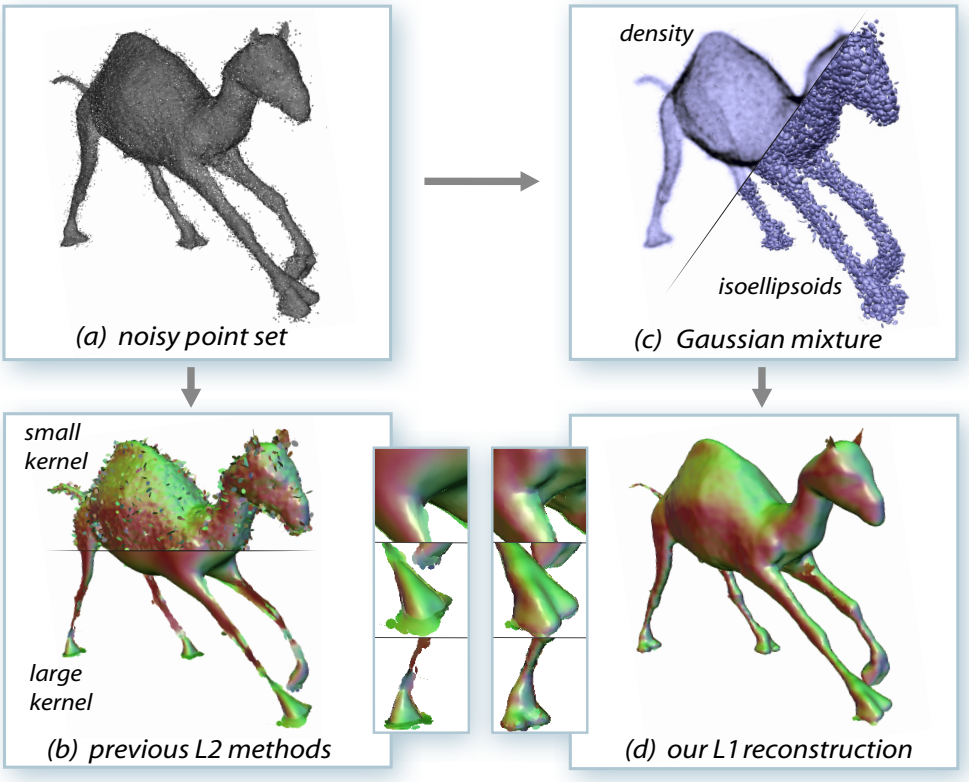


Figure 5.1: Given an input point set (a), L_2-based splat fitting (b) can produce noise or oversmoothing. We efficiently compute a sparse Gaussian mixture (c) and apply a robust *Continuous Projection* operator, which is up to 7 times faster than its discrete variant, thus allowing for an interactive L_1-based reconstruction (d) of unordered dynamic points.

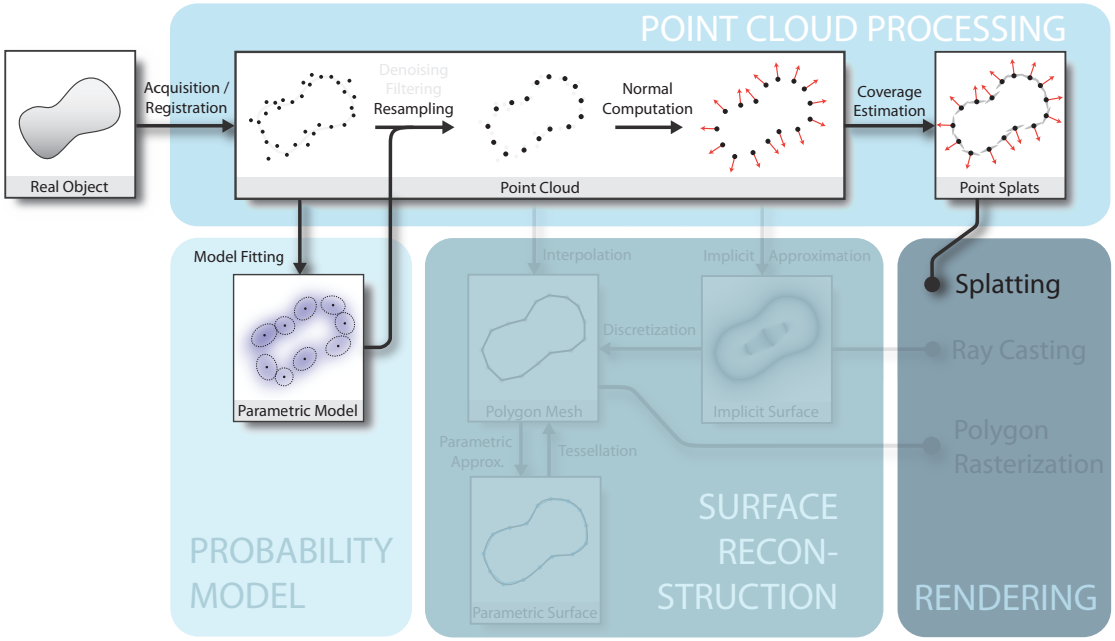


Figure 5.2: A specific kind of parametric model, i.e., a Gaussian mixture model, is fitted to the input point cloud. This *probabilistic* representation of the data allow accelerating a robust state-of-the-art L_1 resampling operator in a way that enables on-the-fly execution on dynamic point data.

5.1 Introduction

In the previous chapter, we have developed a screen-space technique for an on-the-fly reconstruction of splats from a dynamic stream of unordered points, which allows us to obtain an instant high-quality surface visualization. However, a major challenge that has not been addressed so far is the robust performance of such a dynamic reconstruction technique in the face of corrupt and noise-contaminated data, as is commonly given in real-world scanning scenarios due to limited sensor accuracy, occlusions, imperfect registration and other issues. Since standard splat fitting as discussed in Section 4.5 typically corresponds to a least-squares solution, it is sensitive to noise and outliers in the data, producing significant noise artifacts when the reconstruction kernel bandwidth, i.e., the neighborhood radius used for fitting a splat's local support plane, is too small. A straightforward way to address this problem is to increase this kernel bandwidth in order to achieve a stronger noise-filtering effect. However, due to the L_2-based nature of this splat fitting, this also introduces a smoothing of salient features, which can significantly degenerate the surface, as can be seen in the example of the thin limbs of the camel model in Figure 5.1b.

In recent years, more *robust* surface reconstruction techniques have been developed that can deal with defects like noise, outliers, holes or registration errors in the data, while

being much more feature preserving than simple L_2-based methods. They are commonly based on a robust L_1-optimization approach and are able to produce high-quality output despite strongly contaminated data. However, current L_1 methods are typically too expensive to achieve interactive reconstruction times for at least moderately sized point sets, even for parallel implementations. Hence, due to their nature, they are designed for quality rather than performance.

In this chapter, we introduce a highly efficient variant of the *locally optimal projection* (LOP) operator [LCOLTE07], a L_1-based resampling technique that robustly fits a set of *particles* to a noisy point cloud by iteratively applying a system of attractive forces defined by the input points. Characteristically, LOP requires high computational effort for the iterative evaluation of all mutual forces between the particles and the *discrete* set of points. Our approach computes a compact, continuous *probability model* describing the distribution of the discrete input points, and reformulates this LOP operator to be applicable to the resulting *continuous* representation of the point cloud's attractive potential field. This introduces an intermediate probabilistic stage to our dynamic reconstruction pipeline, which allows for an efficient robust denoising prior to the normal and splat computation stage (Figure 5.2). We use a *Gaussian mixture model* (GMM) to describe the point cloud's density in a geometry-preserving manner and show how to compute an *efficient analytic solution* to the integral forces exerted on the particles by each of its continuous components. By operating on a small set of Gaussians instead of the large input point cloud, *Continuous LOP* (CLOP) achieves speedups of up to a factor of 7 over a comparable LOP implementation on the GPU. This makes a robust reconstruction at interactive frame rates for moderately sized dynamic point sets possible (see Figure 5.1). We also demonstrate that the same continuous formulation can be directly applied to the spherical domain to efficiently compute *locally robust point normals* as well. Furthermore, our continuous representation allows for robust point-cloud upsampling. Our results show that despite its much faster computation, our continuous algorithm achieves better point regularity and equal or even higher reconstruction accuracy than its discrete counterpart and even high-quality variants like *Weighted LOP* [HLZ+09].

In the following, we will first review the discrete LOP operator, and then give an overview of the GMM computation from unordered points and show how to transfer this operator to the continuous probability density function of a Gaussian mixture.

5.2 A Review of the LOP Operator

The *Locally Optimal Projection* (LOP) Operator [LCOLTE07] fits a number of points $Q = \{q_i\}_{i \in I}$ (denoted as *particles*) into local medians of a point set $P = \{p_j\}_{j \in J}$, I and J being the respective index sets. The algorithm performs a localized version of Weiszfeld's algorithm for finding the spatial median $q = \underset{\mathbf{x}}{\operatorname{argmin}}\{\Sigma_{j \in J}\|\mathbf{x} - \mathbf{p}_j\|\}$ using a steepest descent on the sum of Euclidean distances from all points p_j. To extend the Weiszfeld algorithm to multiple particles, LOP uses an isotropic, fast decaying localization kernel $\theta(r) = e^{-r^2/(h/4)^2}$ around each particle, which concentrates its influence onto its support

radius h. Starting with an arbitrary initial particle set $Q^{(0)}$, LOP computes the target particle positions Q by performing a fixed-point iteration

$$Q^{(k+1)} = \underset{X=\{x_i\}_{i\in I}}{\operatorname{argmin}} \{E_1(X, P, Q^{(k)}) + E_2(X, Q^{(k)})\} \tag{5.1}$$

where

$$E_1(X, P, Q^{(k)}) = \sum_{i\in I}\sum_{j\in J}\|x_i - p_j\|\theta(\|q_i - p_j\|),$$
$$E_2(X, Q^{(k)}) = \sum_{i'\in I}\lambda_{i'}\sum_{i\in I\setminus\{i'\}}\eta(\|x_{i'} - q_i\|)\theta(\|q_{i'} - q_i\|).$$

Here, E_1 is an energy term attracting Q towards the local medians of P, while E_2 defines a repulsive energy between the particles that strives for an equal distribution of the q_i over the approximated surface. θ denotes the localization kernel constraining both terms to a finite influence radius, $\{\lambda_i\}_{i\in I}$ are weights balancing the particles' attractive and repulsive forces, and η is a repulsion function determining a distance-based repulsion strength (we use $\eta(r) = -r$ as suggested by Huang et al. [HLZ+09]). Eq. (5.1) leads to the following formulation for the updates of each particle $q_i^{(k)} \in Q^{(k)}$ in iteration k. The first iteration acts as an L_2 initializer for Q,

$$q_i^{(1)} = \frac{\sum_{j\in J} p_j\theta(\|p_j - q_i^{(0)}\|)}{\sum_{j\in J}\theta(\|p_j - q_i^{(0)}\|)}, \quad i \in I, \tag{5.2}$$

followed by the fixed-point iteration updates

$$q_i^{(k+1)} = F_1(q_i^{(k)}, P) + \mu\ F_2(q_i^{(k)}, Q_i'^{(k)}) \tag{5.3}$$
$$F_1(q, P) = \sum_{j\in J} p_j \frac{\alpha_j}{\sum_{j'\in J}\alpha_{j'}} \tag{5.4}$$
$$F_2(q, Q_i') = \sum_{i'\in I\setminus\{i\}} (q - q_{i'}) \frac{\beta_{i'}}{\sum_{i''\in I\setminus\{i\}}\beta_{i''}} \tag{5.5}$$

where Q_i' denotes the set of complementary particles $Q\setminus\{q_i\}$. The repulsion parameter $\mu \in [0, 0.5)$ controls the balancing between the attractive forces F_1 of the points p_j and the repulsive forces F_2 from the neighboring particles q_i. Both forces are defined as convex sums over their respective neighbors, with pairwise weights

$$\alpha_j = \frac{\theta(\|p_j - q\|)}{\|p_j - q\|}, \quad \beta_{i'} = \frac{\theta(\|q - q_{i'}\|)}{\|q - q_{i'}\|}\left|\frac{\partial\eta}{\partial r}(\|q - q_{i'}\|)\right|. \tag{5.6}$$

Huang et al. [HLZ+09] proposed an improved, *weighted* version of LOP, referred to as WLOP, that introduces additional balancing of these weights, allowing for a more uniform distribution of the particles in regions of varying point density. We show in Section 5.6 that this additional balancing can be natively integrated into our analytic approach, enabling us to actually perform analytic WLOP. However, for simplicity, we will refer to our method as *Continuous LOP* (CLOP).

5.3 Motivation and Overview

The formulation of Eq. (5.1) can be interpreted as a particle simulation of a set of repulsive particles Q on an attractive background potential field Π, which is represented by a discrete set of samples P. The computational effort of LOP scales with the number $|P|$ of points and the number $|Q|$ of resampling particles to be processed in order to evaluate all mutual forces in the system. As typically $|Q| \ll |P|$, the majority of the time will be spent on the evaluation of the attractive forces from all points p_j, which can be seen as the carriers of the energy potential of Π. We therefore propose to reduce Π to a more compact, yet still accurate representation, which allows evaluating the attraction term much more efficiently.

In our method, we use a mixture $\mathcal{M}$ of anisotropic Gaussians to represent the density of the input points, where $|\mathcal{M}| \ll |P|$ (Section 5.4). We then derive an analytic solution for the continuous attraction forces exerted by each individual Gaussian (Section 5.5). The mixture is efficiently computed from the input points P by a constrained hierarchical expectation-maximization procedure. This one-time effort easily pays off, considering the reduced amount of density-representing entities to process during the following LOP iterations (typically 10–20). In Section 5.6, we show how to extend our approach to WLOP without additional cost. In Section 5.7, we exploit an inherent coherency in the repulsive moments of the particles to also accelerate the evaluation of the repulsion term over all LOP iterations. Finally, we extend our continuous formulation to the robust estimation of normals (Section 5.8).

5.4 Gaussian Mixture Density Computation

In this section, we efficiently reduce the set P of unordered input points to a much more compact mixture of Gaussians $\mathcal{M} = \{w_s, \Theta_s\}$ that reflects the density distribution of the points. That is, $\mathcal{M}$ defines a probability density function (pdf) as a weighted sum of $|\mathcal{M}|$ Gaussian components

$$f(\mathbf{x}|\mathcal{M}) = \sum_s w_s g(\mathbf{x}|\Theta_s), \tag{5.7}$$

where the $\Theta_s = (\mu_s, \Sigma_s)$ are the Gaussian parameters, w_s their corresponding convex weights, and g denotes the d-variate Gaussian pdf with

$$g(\mathbf{x}|\mu, \Sigma) = |2\pi\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu)}. \tag{5.8}$$

We demand $\mathcal{M}$ to be efficiently computable in parallel, and to ideally reflect the density of P while minimizing the smoothing of the signal which LOP tries to reconstruct. To this end, we use a constrained variant of *hierarchical expectation maximization* [Vas98], which aims at optimizing $\mathcal{M}$ in the maximum-likelihood sense while trying not to destroy characteristic information about the underlying geometry. Next, we will shortly review the EM and hierarchical EM (HEM) algorithms, and then present a modified, constrained variant of HEM to compute an accurate density estimate of the point cloud.

5.4.1 Expectation Maximization in GMMs

An ideal density estimate $\mathcal{M}$ of an input point set $\{\mathbf{p}_i\}$ is defined in a way such that it maximizes the likelihood $\mathcal{L}(\mathcal{M}) = \prod_i f(\mathbf{p}_i|\mathcal{M})$ of producing the set P under $\mathcal{M}$. Starting with an initial guess $\mathcal{M}^{(0)}$, Expectation Maximization [DLR77] computes such a *maximum likelihood estimate* (MLE) for mixture models by iteratively optimizing an estimator of the parameters of $\mathcal{M}$ until a local maximum of the objective log-likelihood function $\mathcal{L}_{log} = log\ \mathcal{L}(\mathcal{M})$ is found. It thereby uses a discrete distribution of posterior responsibility probabilities r_{is} for a fuzzy assignment of each point to each component, and optimizes them along with the model parameters. This is done in an alternating two-step procedure, which eventually converges to a local maximum of $\mathcal{L}_{log}$:

E-Step: Given the current model parameters $\mathcal{M}$, compute the expected responsibilities

$$r_{is} = \frac{\mathcal{L}(\Theta_s|\mathbf{p}_i)w_s}{\sum_{s'} \mathcal{L}(\Theta_{s'}|\mathbf{p}_i)w_{s'}}, \quad \mathcal{L}(\Theta_s|\mathbf{p}_i) = g(\mathbf{p}_i|\Theta_s) \tag{5.9}$$

M-Step: Based on the new responsibilities, update the model parameters $\mathcal{M}'$. For Gaussian components, these are the points' weighted means μ_s and weighted covariances Σ_s with convex weights $r_{is}/\sum_{i'} r_{i's}$, and mixture coefficients $w_s = \sum_i r_{is}/|P|$.

5.4.2 Hierarchical EM

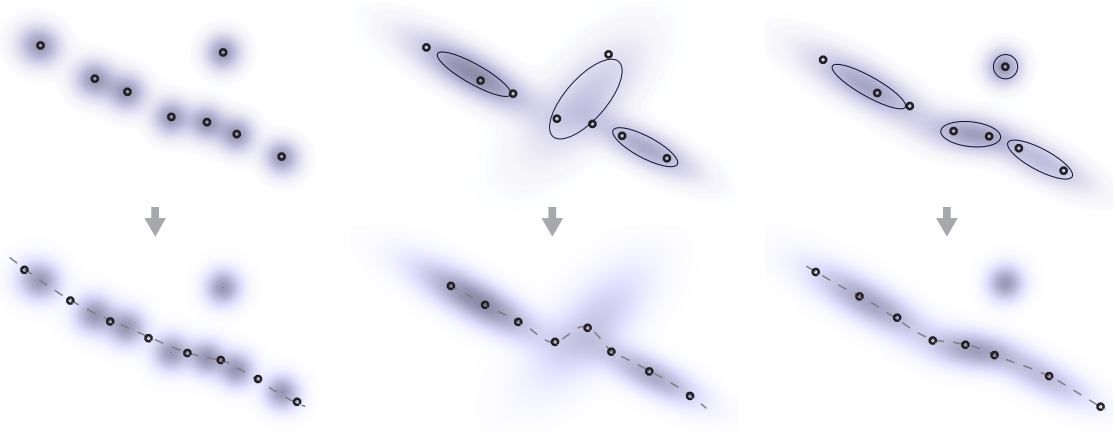
In contrast to classic EM, hierarchical EM performs only one initial EM iteration on the complete input data, and then successively reduces the mixture by hierarchically applying EM on Gaussians instead of points. HEM equips each input point $\mathbf{p}_i$ with an initial low-variance Gaussian $\Theta_i^{(0)}$, which results in an initial mixture $\mathcal{M}^{(0)} = \{w_s^{(0)}, \Theta_s^{(0)}\}$ with initially equal component weights $w_s^{(0)} = 1/|P|$. The component parameters $\Theta_s^{(l+1)}$ of the next level are then estimated based on those of the current level l by a modified EM step. Since each Gaussian Θ_i represents $\overline{w}_i = w_i|P|$ points, HEM alters the likelihood function $\mathcal{L}$ employed in the E-Step in Eq. (5.9) to incorporate $\overline{w}_i$ representative "virtual samples":

$$\mathcal{L}(\Theta_s^{(l+1)}|\Theta_i^{(l)}) = \left[g(\mu_i^{(l)}|\Theta_s^{(l+1)})\ e^{-\frac{1}{2} tr([\Sigma_s^{(l+1)}]^{-1}\Sigma_i^{(l)})} \right]^{\overline{w}_i} \tag{5.10}$$

Given the responsibilities r_{is} and the mixture $M^{(l)}$ of the current level, the model parameters of the next higher level are again maximized by convex sums

$$\begin{aligned} w_s^{(l+1)} &= \sum_i r_{is}\ w_i & \mu_s^{(l+1)} &= \sum_i \omega_{is}\mu_i \\ \Sigma_s^{(l+1)} &= \sum_i \omega_{is} \left(\Sigma_i^{(l)} + (\mu_i^{(l)} - \mu_s^{(l+1)})(\mu_i^{(l)} - \mu_s^{(l+1)})^T \right) \end{aligned} \tag{5.11}$$

with convex weights $\omega_{is} = r_{is}w_i/\sum_{i'} r_{i's}w_{i'}$. To initialize the mixture $\mathcal{M}^{(l+1)}$ of each next higher level before the hierarchical EM-step, we randomly subsample the set $\mathcal{M}^{(l)}$ (usually $\sim 33\%$).



(a) (b) (c)

Figure 5.3: Gaussian Mixture on a signal with an outlier (top) and its LOP reconstruction (bottom). Ellipses denote Gaussians' one-σ-isodistances. (a) Initial mixture. (b) Level $\mathcal{M}^{(1)}$ of unconstrained HEM. (c) $\mathcal{M}^{(1)}$ with constrained clustering radius. Standard HEM tends to smooth the signal, while regularized HEM is more feature preserving, at the cost of less component reduction.

5.4.3 Geometrically Regularized HEM

Since the maximum likelihood estimate of a Gaussian is a least-squares solution, thus non-robust, an ordinary MLE of a Gaussian Mixture is inherently prone to bias the input signal in a way that obliterates any subsequent robust reconstruction. Thus, without any further consideration, a mere *statistically* optimal fit of $\mathcal{M}$ could place a Gaussian component in a way that blurs the information of outliers against which we want to robustly reconstruct, as illustrated in Figure 5.3a and 5.3b. While there are alternative distributions that provide a robust MLE, like the Laplace distribution, these cannot be expressed in closed form and would thus require an expensive iterative approximation. Instead, we improve the robustness of the Gaussian mixture by adopting a *geometric* regularization to Hierarchical EM, which stems from the idea of agglomerative hierarchical clustering to merge only those clusters which are closest under a given distance measure. Restricting the set of neighboring components that are considered for merging to a *clustering kernel* of finite radius ρ allows merging the energy mass of close-by Gaussians, while leaving more distant clusters untouched (Figure 5.3c). This results in a *regularized hierarchical EM* procedure, which strives for a maximum likelihood estimate under a reinforced similarity constraint.

Dissimilarity Measure. To measure the distance between two Gaussians Θ_t and Θ_s in $\mathbb{R}^3$, we use their Kullback-Leibler divergence

$$D_{KL}(\Theta_t \| \Theta_s) = \tfrac{1}{2}\left(d_M(\mu_t, \Theta_s)^2 + tr(\Sigma_s^{-1}\Sigma_t) - 3 - \ln \tfrac{|\Sigma_t|}{|\Sigma_s|}\right) \tag{5.12}$$

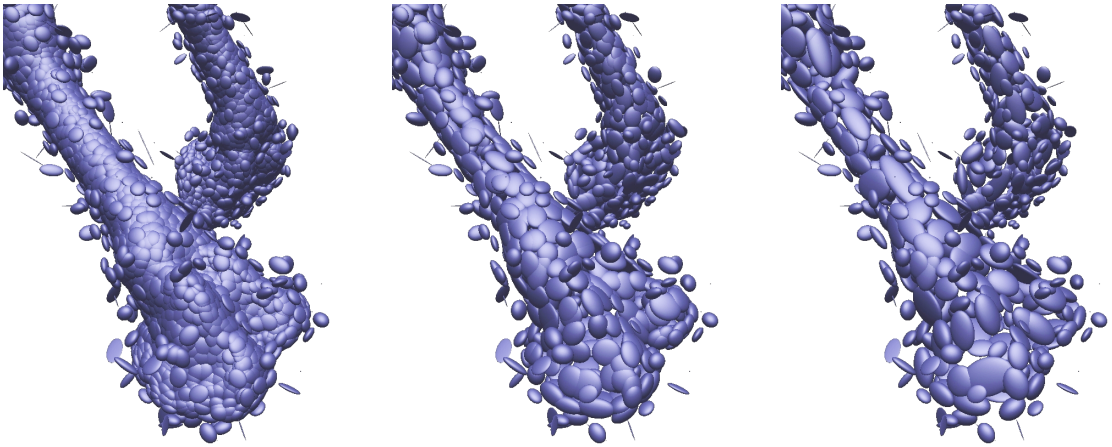
and define ρ to be the maximum distance $D_{KL}(\Theta_t \| \Theta_s)_{max}$ within which Θ_s is allowed to merge other components Θ_t. Although D_{KL} is a measure of relative entropy, it has an intuitive geometric interpretation, as it accounts for both the scale-invariant Mahalanobis distance d_M between their centers as well as the deviation of their principal component directions. Thus, D_{KL} lets large anisotropic Gaussians continue clustering in the direction of their largest variance, while smaller Gaussians, possibly representing outlier points, are restricted to a small clustering radius.

Clustering Kernel Size. In contrast to previous authors [JRJ11, WBKP08], who choose ρ to be the n-th globally smallest occurring distance between Gaussians, we try to avoid such a global computation, but rather choose ρ to be a good compromise between clustering efficiency (large, relaxed ρ), and geometric accuracy (small, restrictive ρ). To provide an intuitive control over ρ, we suggest a free parameter α, so that $\rho = \alpha^2/2$, which has a simple interpretation: If two Gaussians have equal covariances, thus presumably representing similarly oriented geometry, Eq. (5.12) reduces α to a simple threshold of the Mahalanobis distance of their centers. On the other hand, assuming the Gaussians have coinciding centers, differently oriented covariances suggest a change in orientation of the underlying surface, which α will segregate even more. In our experiments, $\alpha \approx 2$ has proven to give a good balance between clustering efficiency and accuracy.

Mixture Initialization. The initial mixture $\mathcal{M}^{(0)}$ needs to be defined in a way that allows α to provide a similar regularization behavior throughout all levels of the hierarchy. Placing an initial Gaussian at each point ($\mu_i^{(0)} = p_i$) creates a simple kernel density estimate of $|P|$, whose kernel bandwidth defines the extent of the covariances $\Sigma_i^{(0)}$ [Vas98]. A too small bandwidth requires a large α to allow any clustering at all, but also diminishes the regularization effect in subsequent levels. On the other hand, a too large bandwidth smooths the signal in advance, thus again increasing the reconstruction bias. To produce a suitable initial density estimate, we first use a conservative kernel radius r (usually $2 \sim 3$ times a point's nearest neighbor distance) to compute for each point an initial anisotropic covariance matrix Σ_r, whose shape already reflects the local distribution of the n points within the kernel. In a second step, Σ_r is scaled down such that its unit-σ-ellipsoid fits its local nearest neighbor distances. This gives an initial covariance

$$\Sigma^{(0)} = \Sigma_r \frac{r}{\sigma_{max} \sqrt[3]{n}} + c\, I, \tag{5.13}$$

where σ_{max} is the square root of the largest eigenvalue of Σ_r, I is the identity matrix, and c is a small trace bias giving $\Sigma^{(0)}$ a minimum extent. Figure 5.4 shows the feature-preserving reduction of the mixture from Figure 5.1 over different hierarchy levels of regularized HEM.



(a) $|\mathcal{M}^{(0)}| = 64K$ (b) $|\mathcal{M}^{(2)}| = 13K$ (c) $|\mathcal{M}^{(4)}| = 8400$

Figure 5.4: Unit-σ-isosurfaces of the mixture Gaussians at the camel model's front hooves at different hierarchy levels for $\alpha = 2.1$. Note how with successive levels, the main signal components are merged, while the Gaussians modeling outliers remain unchanged.

5.5 Continuous LOP in Gaussian Mixtures

In this section we show how to apply the robust LOP operator to a mixture of Gaussians. By reformulating the attractive force F_1, we obtain our accelerated CLOP algorithm.

5.5.1 Reformulation of the Attraction Force

Eq. (5.4) concentrates the attractive energies of the potential field Π in singular points P, and defines F_1 as a convex weighted sum over all points p_j with corresponding weights α_j. As $\mathcal{M}$ now continuously distributes these energies according to its density function (5.7), we define a corresponding *continuous* force $\mathcal{F}_1$ by the convex sum over the integral attraction of each single Gaussian, with convex weights w_s accounting for the Gaussians' relative point mass:

$$\mathcal{F}_1(q, \mathcal{M}) = \sum_s w_s \int_{\mathbb{R}^3} \frac{\mathbf{x}\ g(\mathbf{x}|\Theta_s)\alpha(\mathbf{x})}{\sum_{s'} w_{s'} \int_{\mathbb{R}^3} g(\mathbf{x}'|\Theta_{s'})\alpha(\mathbf{x}')\ \mathrm{d}\mathbf{x}'}\ \mathrm{d}\mathbf{x} \tag{5.14}$$

where similar to Eq. (5.6), we define the weight $\alpha(\mathbf{x}) = \theta(\delta)/\delta$, with $\delta = \|\mathbf{x} - q\|$. Additionally, each point $\mathbf{x} \in \mathbb{R}^3$ is now weighted by the Gaussian density g of the corresponding component Θ_s. As before, the integral over all weighted contributions is normalized by the integral over all weights. Figure 5.5a illustrates the spatial weights induced on the domain $\mathbb{R}^3$ by an anisotropic Gaussian Θ_s and a radial kernel α centered at a particle q. Multiplying all occurring weights into a combined weight function

$$\Omega_s(\mathbf{x}) = w_s g(\mathbf{x}|\Theta_s)\alpha(\mathbf{x}) \tag{5.15}$$

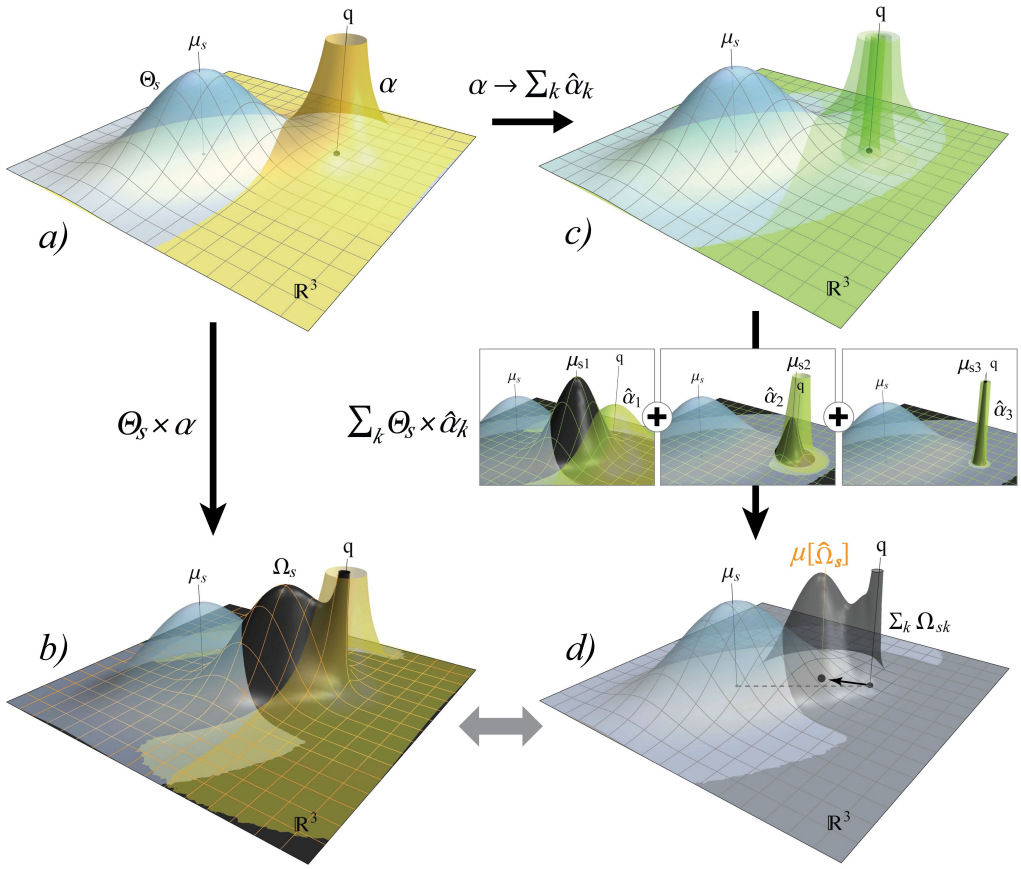
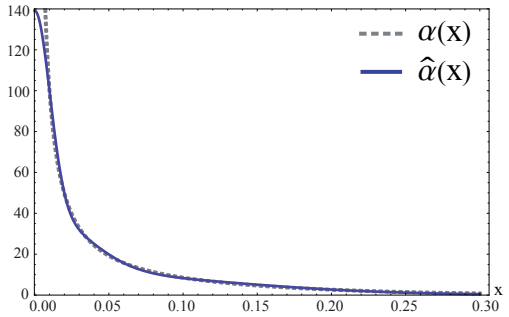


Figure 5.5: Continuous attraction of a particle q from a Gaussian Θ_s. (a) Density of Θ_s (blue) and kernel α (yellow) centered in q lead to (b) a product weight Ω_s (black) with infinite integral. (c) Approximation of α by a sum of 3 Gaussians (green) divides this integral into 3 finite product Gaussians Ω_{sk} (black). The sum of their means, convexly weighted by their integrals, yields (d) the estimated mean of Ω_s, which is the destination point of q given by the integral attraction of Θ_s. Note the good approximation quality of $\hat{\Omega}_s$ (d) compared to Ω_s (b) with only 3 Gaussian summands.

defines the attraction step $\mathcal{F}_1$ as

$$\mathcal{F}_1(q, \mathcal{M}) = \frac{\sum_s \int_{\mathbb{R}^3} \mathbf{x}\, \Omega_s(\mathbf{x})\, \mathrm{d}\mathbf{x}}{\sum_s \int_{\mathbb{R}^3} \Omega_s(\mathbf{x})\, \mathrm{d}\mathbf{x}}. \tag{5.16}$$

Figure 5.5b shows the form of the combined weight function Ω_s for the generating Gaussian Θ_s and kernel α. Note that the integral in Eq. (5.16) is not finite due to a singularity at $\delta = 0$ produced by its factor δ^{-1}. However, in the basic Weiszfeld's algorithm as well as in LOP, a particle spatially coinciding with an input point (i.e., $\delta = 0$) represents a singularity anyway, which is typically accounted for by removing the point in question, or biasing the denominator δ^{-1} to some $(\epsilon + \delta)^{-1}$ to clamp the point's energy at $\delta = 0$ to a finite peak. Following the same reasoning, we circumvent the infinite



(a)

k	$\hat{w}_k$	$\hat{\sigma}_k$
1	11.453	0.11772
2	29.886	0.03287
3	97.761	0.01010

(b)

Figure 5.6: (a) Plot of the original function α (dashed) and its approximation by a sum of 3 Gaussians (solid) using the coefficients listed in (b). Note the finite peak of $\hat{\alpha}$.

integral by approximating the original weight function α by a sum of K Gaussians

$$\hat{\alpha}(\mathbf{x}) = \sum_{k=1}^{K} \hat{\alpha}_k(\mathbf{x}) = \sum_{k=1}^{K} \hat{w}_k \hat{c}_k \; g(\mathbf{x}|q, \hat{\Sigma}_k) \tag{5.17}$$

(Figure 5.5c), which provides an integrable finite peak at $\delta = 0$ while still exhibiting the characteristic weight falloff of α. Eq. (5.17) gives the general d-dimensional formulation for $\hat{\alpha}_k$, where $\hat{\Sigma}_k = \hat{\sigma}_k^2 h^2 I$ denotes its covariance, $\hat{w}_k$ the (dimension-invariant) weight, and $\hat{c}_k = |2\pi\hat{\Sigma}_k|^{\frac{1}{2}}$ compensates for the dimension-dependent normalization factor of the pdf g. Since the Gaussian kernel is normalized by the LOP support radius h, the coefficients $\hat{w}_k$ and $\hat{\sigma}_k$ of the model function (5.17) can be fitted by setting $h = 1$ and considering only the normalized range $\delta \in [0, 1]$. In our experiments, we observed that a sum of $K = 3$ Gaussians provides a sufficient approximation, as shown in Figure 5.6a. The coefficients obtained using Levenberg-Marquardt optimization are listed in Figure 5.6b. Replacing α by $\hat{\alpha}$ gives an estimator $\hat{\Omega}$ approximating the combined weight function (5.15) by

$$\hat{\Omega}_s(\mathbf{x}) = w_s g(\mathbf{x}|\Theta_s) \sum_{k=1}^{K} \hat{\alpha}_k(\mathbf{x}) = \sum_{k=1}^{K} \hat{\Omega}_{sk}(\mathbf{x}). \tag{5.18}$$

This comes with a convenient property: Since the product of two Gaussians is again a Gaussian, Eq. (5.18) reduces the complete weight function $\hat{\Omega}_s$ to a sum of K product Gaussians $\hat{\Omega}_{sk}$, which we can again interpret as weighted Gaussian pdfs, with weights ω_{sk} and means μ_{sk}. Therefore, Equation (5.16) can now be expressed in **closed form** by

$$\mathcal{F}_1(q, \mathcal{M}) = \frac{\sum_s \sum_k \int_{\mathbb{R}^3} \mathbf{x} \; \hat{\Omega}_{sk}(\mathbf{x}) \; \mathrm{d}\mathbf{x}}{\sum_s \sum_k \int_{\mathbb{R}^3} \hat{\Omega}_{sk}(\mathbf{x}) \; \mathrm{d}\mathbf{x}} = \frac{\sum_{s,k} \omega_{sk} \mu_{sk}}{\sum_{s,k} \omega_{sk}}, \tag{5.19}$$

which in the same way that Eq. (5.4) is a convex sum of 3D points p_j, now becomes a convex combination of the product Gaussians' means μ_{sk} with weights ω_{sk}. By applying the identities for the integral and the expectation of a Gaussian product [PP12], we derive these quantities as follows.

Weight ω_{sk}. Using Eq. (5.18), we obtain ω_{sk} for the general d-dimensional case as

$$\begin{aligned}\omega_{sk} &= \int_{\mathbb{R}^d} \hat{\Omega}_{sk}(\mathbf{x})\mathrm{d}\mathbf{x} = w_s\hat{w}_k\hat{c}_k \int_{\mathbb{R}^d} g(\mathbf{x}|\Theta_s)\; g(\mathbf{x}|\hat{\Theta}_k)\mathrm{d}\mathbf{x} \\ &= w_s\,\hat{w}_k\,\hat{c}_k\,g(\mu_s|q,\Lambda_{sk}) \\ &= w_s\,\hat{w}_k\,\hat{\sigma}_k^d\,h^d\,|\Lambda_{sk}|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mu_s-q)^T\Lambda_{sk}^{-1}(\mu_s-q)}\end{aligned} \tag{5.20}$$

where we have introduced the covariance sum $\Lambda_{sk} = \Sigma_s + \hat{\Sigma}_k$.

Mean μ_{sk}. Evaluating the weighted mean in the numerator of Eq. (5.19) gives

$$\begin{aligned}\omega_{sk}\mu_{sk} &= \int_{\mathbb{R}^d} \mathbf{x}\;\hat{\Omega}_{sk}(\mathbf{x})\mathrm{d}\mathbf{x} = w_s\hat{w}_k\hat{c}_k \int_{\mathbb{R}^d} \mathbf{x}\; g(\mathbf{x}|\Theta_s)\; g(\mathbf{x}|\hat{\Theta}_k)\mathrm{d}\mathbf{x} \\ &= w_s\,\hat{w}_k\,\hat{c}_k\,g(\mu_s|q,\Lambda_{sk})(\Sigma_s^{-1}+\hat{\Sigma}_k^{-1})^{-1}(\Sigma_s^{-1}\mu_s+\hat{\Sigma}_k^{-1}q) \\ &= \omega_{sk}(\Sigma_s^{-1}+\hat{\Sigma}_k^{-1})^{-1}(\Sigma_s^{-1}\mu_s+\hat{\Sigma}_k^{-1}q).\end{aligned} \tag{5.21}$$

Due to the expensive inversions in this formulation, we centralize the coordinate frame in q to further simplify the mean

$$\begin{aligned}\mu_{sk} &= (\Sigma_s^{-1}+\hat{\Sigma}_k^{-1})^{-1}(\Sigma_s^{-1}(\mu_s-q)+\hat{\Sigma}_k^{-1}(q-q))+q \\ &= \hat{\Sigma}_k(\Sigma_s+\hat{\Sigma}_k)^{-1}(\mu_s-q)+q \\ &= \hat{\sigma}_k^2\,h^2\Lambda_{sk}^{-1}(\mu_s-q)+q.\end{aligned} \tag{5.22}$$

This way, the evaluation of both quantities requires only one matrix inversion of Λ_{sk}, which already produces the term $|\Lambda_{sk}|^{-1}$ required in ω_{sk} as side product. The final complete continuous attraction step is thus given by

$$\mathcal{F}_1(q,\mathcal{M}) = q + \sum_{s,k} \hat{\sigma}_k^2\,h^2\,\Lambda_{sk}^{-1}(\mu_s-q)\frac{\omega_{sk}}{\sum_{s',k'}\omega_{s'k'}}. \tag{5.23}$$

5.5.2 Initial Iteration

As shown in Eq. (5.2), LOP initializes its particle positions with the weighted mean of the input points using the weight kernel θ. Its continuous variant

$$\mathcal{F}_1^{(1)}(q,\mathcal{M}) = \frac{\sum_s \int_{\mathbb{R}^3} \mathbf{x}\,w_s g(\mathbf{x}|\Theta_s)\theta(\delta)\,\mathrm{d}\mathbf{x}}{\sum_s \int_{\mathbb{R}^3} w_s g(\mathbf{x}|\Theta_s)\theta(\delta)\,\mathrm{d}\mathbf{x}} = \frac{\sum_s \omega_s^{(0)}\mu_s^{(0)}}{\sum_s \omega_s^{(0)}} \tag{5.24}$$

is similar to Eq. (5.14), except that it omits the term δ^{-1} and can thus be evaluated only by the weight θ instead of K summands $\hat{\alpha}_k$. Expressing $\theta(\delta)$ by a scaled Gaussian pdf $c_\theta\,g(\mathbf{x}|q,\Sigma_\theta)$, with $\Sigma_\theta = (h^2/32)I$ and $c_\theta = |2\pi\Sigma_\theta|^{\frac{1}{2}}$, gives the quantities for the initial weight and mean as

$$\omega_s^{(0)} = w_s\,c_\theta\,g(\mu_s|q,\Lambda_{s\theta}), \quad \mu_s^{(0)} = \frac{1}{32}h^2\Lambda_{s\theta}^{-1}(\mu_s-q)+q, \tag{5.25}$$

with covariance sum $\Lambda_{s\theta} = \Sigma_s + \Sigma_\theta$.

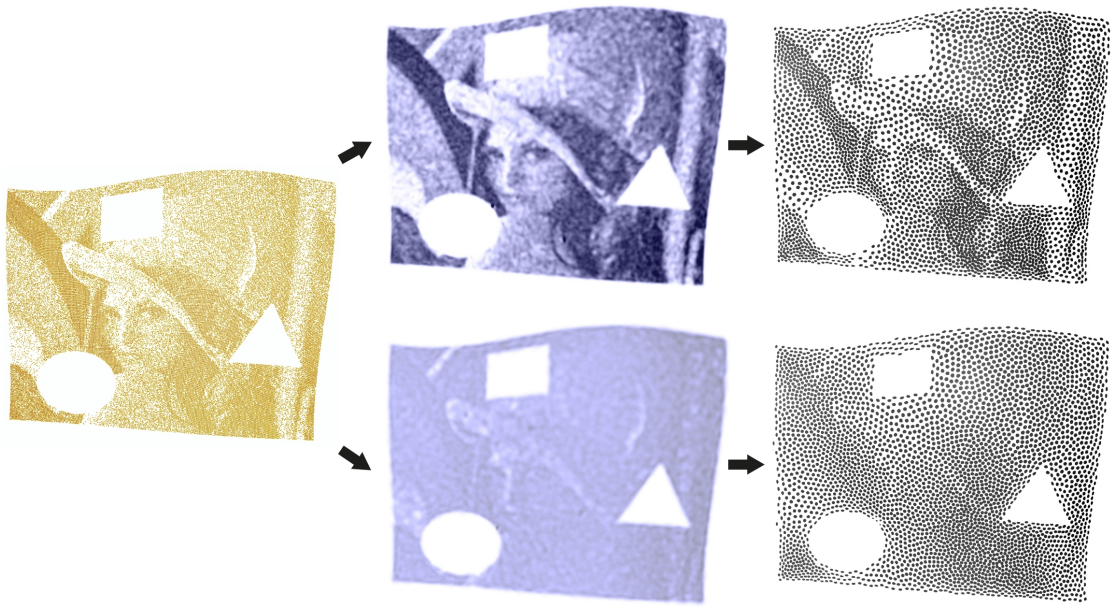


Figure 5.7: Point sampling of Lena with density inverse proportional to image intensity (74K points, left), its corresponding mixture $\mathcal{M}^{(4)}$ (5K Gaussians, middle) and CLOP resampling (3700 particles, right). The top row shows the unweighted mixture, resulting in an unevenly distributed resampling, while the bottom shows the desired, balanced particle distribution built on initially weighted Gaussians.

5.6 Weighted CLOP

As the original LOP operator is very sensitive to regions of varying point densities, Huang et al. [HLZ+09] proposed a weighted LOP operator (WLOP), which normalizes the attractive force (5.4) over differently dense regions by adding for each point p_j a density-dependent weight $v_j = 1 + \sum_{j' \in J \setminus \{j\}} \theta(\|p_j - p_{j'}\|)$, so that

$$F_1(q, P) = \sum_{j \in J} p_j \frac{\alpha_j / v_j}{\sum_{j' \in J} \alpha_{j'} / v_{j'}}. \tag{5.26}$$

This additional weighting can be easily adopted in CLOP, without even changing its integral formulations in Section 5.5. Since a Gaussian's attractive potential is defined by its weight w_s, we can encode the balancing weights v_j directly in the Gaussians representing the p_j in the initial mixture $\mathcal{M}^{(0)}$, by altering their initial weights to

$$w_j^{(0)} = (v_j |P|)^{-1}. \tag{5.27}$$

This way, the attraction-determining weights w_j of Gaussians that cluster points in dense regions (large v_j) will be relaxed more strongly than weights in regions of lower density. Applying CLOP to such a *weighted* mixture thus results in a continuous equivalent of

the weighted attraction in WLOP. As we can directly accumulate the sum (5.26) along with the points' covariances in the initial kernel pass (Section 5.4.3), this weighting can be achieved in CLOP without any additional effort. Figure 5.7 recreates the Lena demo from Huang et al. [HLZ+09], demonstrating the improved performance of CLOP when using such weighted mixtures.

5.7 Accelerating Repulsion

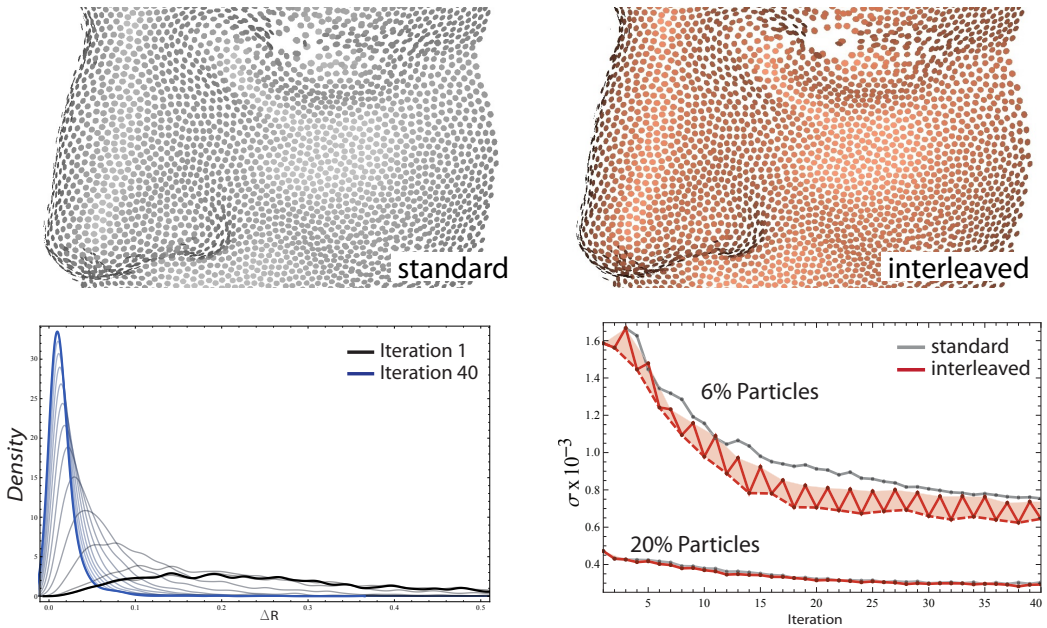
The reformulation and continuous evaluation of F_1 shown in the previous sections accelerates the major part of the computational workload in a LOP iteration (5.1). As a result, when using a larger number of particles, the discrete computation of the repulsion forces becomes the bottleneck. In this section, we address this problem by two strategies.

5.7.1 Kernel Cutoff

A simple way to accelerate repulsion is to skip the evaluation of F_2 (5.5) between particles where the relative repulsive influence is very low. For higher particle counts, this applies to all particles q with distance $\gtrsim h/2$ to a repulsing particle q', due to the Gaussian kernel falloff weighting this repulsion exponentially lower than those from particles closer to q. We have observed that simply cutting off the repulsion kernel at about half its radius reduces the repulsion computation effort by $\sim 75\%$, while having a negligible effect on the regularity of the final particle distribution. Note, however, that such a cutoff is not applicable to the attraction force, as there it is crucial for the kernel to bridge the gap between an outlier and the surface it should be projected to.

5.7.2 Repulsion Coherence

Another optimization exploits a coherence in the particles' repulsive moments $F_2(q, Q')$ in Eq. (5.5), which we will here denote as $\dot{R}$. We have observed that although the moment $\dot{R}$ of each individual particle q is highly dynamic in both direction and magnitude, the relative change in the overall system is generally low. To measure a particle's coherence of $\dot{R}$ between two iterations, we examine its relative change of magnitude $\Delta\dot{R}^{(k)} = \frac{\|\dot{R}^{(k)} - \dot{R}^{(k-1)}\|}{\|\dot{R}^{(k-1)}\|}$, which can also be thought of as a scale-invariant error measure when using $\dot{R}^{(k-1)}$ to approximate $\dot{R}^{(k)}$. Figure 5.8a shows the distribution of $\Delta\dot{R}$ for the face data set as it develops over different numbers of CLOP iterations. The graphs indicate that the overall error $\Delta\dot{R}$ is bounded and progressively reduces as CLOP converges. After the first iteration, the repulsive moments of most particles do not deviate more than $\sim 40\%$, and with successive iterations, hardly more than 10%. This observation suggests that, under acceptance of the discussed error, a given $\dot{R}^{(k)}$ can be used as an estimator for $\dot{R}^{(k+1)}$ in the next iteration. Therefore, we are able to reduce the computation effort by another 50% alone by reusing the repulsion vectors in every second CLOP iteration. We propose to still perform an actual repulsion computation in the final iteration in case of an odd number of iterations.



(a) (b)

Figure 5.8: (a) Development of the distribution of $\Delta\dot{R}$ for the Face model, from first (black) to the last (blue) CLOP iteration. (b) Convergence of nearest-neighbor variances σ for unoptimized (gray) and interleaved repulsion (red). The closeup visually compares the resulting particle distributions.

Figure 5.8b plots the variance σ of nearest neighbor distances, measuring the regularity of the point distribution [HLZ+09] for both unoptimized and interleaved repulsion and different amounts of resampling particles. We observe that with a larger relative number of particles (20%), an interleaved repulsion update hardly affects the convergence behavior of σ. On the other hand, a lower number of particles (6%) allows them to move more freely, leading to an oscillation of σ when correcting the repulsive moment only each second iteration. However, the band in which it oscillates generally appears to drop faster, which shows that in addition to a performance improvement, interleaved repulsion actually leads to a potentially faster convergence in point regularity.

5.8 Robust Normal Computation in Mixtures

Having derived a robust projection operator for spatial data to accelerate the L_1 point resampling, we are interested in a similar speedup for locally robust normal reconstruction methods [OGG09, ZSW+10]. In this section we show that our derivation of the continuous attraction in Section 5.5 can be directly applied to the domain of unit normals to quickly compute L_1-aligned, unoriented normals for the particles obtained by CLOP.

5.8.1 Spherical Weiszfeld for Normal Axes

The basic idea of our local L_1-based normal alignment is to find the robust median within a set of unoriented normals *(normal axes)* $\mathbf{m}_j^* \in \mathbb{S}^2$ of spatially neighboring points $\mathbf{p}_j$, which can be roughly estimated using standard PCA. Similar to how Weiszfeld's algorithm iteratively approximates the spatial median of noisy points in $\mathbb{R}^3$, we can use a spherical equivalent to find a *spherical median* $\mathbf{n}_{opt} = \underset{\mathbf{n}\in\mathbb{S}^2}{\operatorname{argmin}}\{\sum_{j\in J} d_g(\mathbf{m}_j, \mathbf{n})\}$ of these noisy estimated point normal axes, which minimizes the sum of geodesic distances $d_g(\mathbf{m}_j, \mathbf{n}) = cos^{-1}\langle \mathbf{m}_j, \mathbf{n}\rangle_{max}$ [BDGS05]. Here, $\mathbf{m}_j$ represents the unit vector parallel to the (bipolar) axis $\mathbf{m}_j^*$ that minimizes the geodesic distance $d_g(\mathbf{m}_j, \mathbf{n})$. Based on the above definition, we can define a spherical Weiszfeld iteration that moves an initial estimator of a particle normal $\mathbf{n}$ towards the median of neighboring point normal axes $\mathbf{m}_j^*$ by

$$\mathbf{n}' = \frac{\sum_{j\in J} \mathbf{m}_j\, \alpha_j}{\|\sum_{j\in J} \mathbf{m}_j\, \alpha_j\|}, \quad \alpha_j = \frac{\theta(\|p_j - q\|)}{d_g(\mathbf{m}_j, \mathbf{n})} \tag{5.28}$$

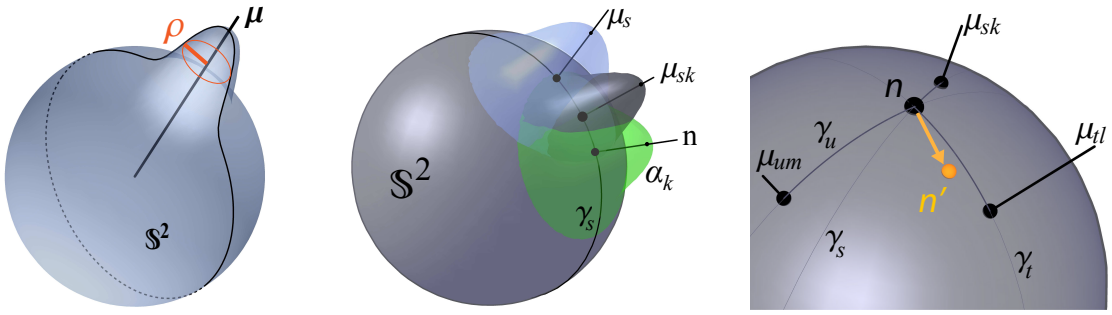
where θ localizes the median projection to point neighbors within a compact range as before. Projecting the normal of each particle into the median of a set of point normals $\mathbf{m}_j$ produces the same computational effort as the LOP attraction term in Eq. (5.4). We will therefore now introduce a fast continuous variant of the spherical Weiszfeld algorithm, which corresponds to CLOP and operates on a spherical mixture distribution of the unoriented point normals $\mathbf{m}_j^*$.

5.8.2 Spherical Mixture Distribution

Similar to how HEM reduces the input points to a mixture of continuous spatial distributions, we reduce the set of estimated point normals $\mathbf{m}_j$ to a set of *wrapped normal* (WN) distributions, which can be thought of as normal distributions endlessly wrapped around the unit circle [MJ09]. They give an approximate description of the Mises-Fisher (vMF) distribution, which is a well established and exact model for a random variate on $\mathbb{S}^2$. Nevertheless, considering a hierarchical clustering and doing calculus to obtain a continuous formulation of a spherical Weiszfeld iteration, vMFs are hard to handle directly. However, we can sufficiently simulate a vMF by rotating a one-dimensional wrapped normal distribution $\Phi = (\mu, \rho)$ around its mean μ on the unit sphere (Figure 5.9a). The concentration parameter ρ defines the dispersion of the distribution and measures the *mean resultant length* $\rho = \|\sum_j \mathbf{m}_j\|/n$ of a set of n unit vectors $\mathbf{m}_j$, increasing in value as the dispersion decreases. ρ also relates to the variance of the standard normal distribution by $\sigma^2 = -2\ log\ \rho$. In the following, we will use both ρ and σ^2 in our derivations. The pdf of Φ is given by

$$g_w(x|\Phi) = \frac{1}{\sigma\sqrt{2\pi}} \sum_{k=-\infty}^{\infty} e^{-\frac{1}{2}(\frac{x-\mu+2\pi k}{\sigma})^2}. \tag{5.29}$$

For reasonably concentrated distributions (variance bounded by 2π), the sum representing the infinite wrapping of the distribution can be sufficiently approximated by the term



(a) (b) (c)

Figure 5.9: (a) Univariate WN distribution on the $\mathbb{S}^2$, creating a spherical normal distribution by rotation. (b) Since both α_k and μ_s are isotropic, their product μ_{sk} lies on their common geodesic γ_s. (c) Weiszfeld step for $\mathbf{n}_j$ defined by the weighted mean of all μ_{sk}.

$k = 0$ ([MJ09] p. 50), which gives a standard normal distribution. Note that in Eq. (5.29), μ and x represent angles on a great circle.

An ordinary HEM-like maximum-likelihood estimate of the set of Φ_s on the spherical domain [BDGS05] is not sufficient for our needs, as it neglects the association of normals $\mathbf{m}_j$ to points $\mathbf{p}_j$ required for the spatial localization kernel θ in Eq. (5.28). Therefore, instead of computing a spherical mixture independent from the Gaussian point distributions $\mathcal{M}$, we assign a distinct WN Φ_s to each Gaussian $\Theta_s \in \mathcal{M}$, and cluster them along with the Gaussians during the HEM procedure described in Section 5.4, i.e., the normals do not influence the computed responsibilities which determine the clustering. This leads to a simple extension of the HEM clustering algorithm:

1. For each point p_j, extend the initial mixture $\mathcal{M}^{(0)}$ by an initial WN distribution $\Phi_j^{(0)} = (\mathbf{m}_j, \rho^{(0)})$, with $\rho^{(0)} = 1$.
2. In the M-Step, update the MLE of Φ_s using the same spatial weights r_{is} and $\bar{w}_s$ that cluster Θ_s.

We define the MLE updates for a next level WN $\Phi_s^{(l+1)}$ by

$$\mu_s^{(l+1)} = \frac{\sum_i \mu_i^{(l)}\ r_{is}\overline{w}_i}{\|\sum_i \mu_i^{(l)}\ r_{is}\overline{w}_i\|} \tag{5.30}$$

$$[\sigma_s^2]^{(l+1)} = \frac{\sum_i \sigma_i^2\ r_{is}\overline{w}_i}{\sum_i r_{is}\overline{w}_i} - 2\ log\left(\left\|\frac{\sum_i \mu_i^{(l)}\ r_{is}\overline{w}_i}{\sum_i r_{is}\overline{w}_i}\right\|\right), \tag{5.31}$$

which is the spherically wrapped isotropic equivalent to the clustering of Gaussians in Eq. (5.11). Since the *log*-argument in Eq. (5.31) is the mean resultant length ρ of WN

means $\mu_i^{(l)}$, the complete second term gives the variance of these $\mu_i^{(l)}$. Thus, according to Eq. (5.11), Eq. (5.31) defines the MLE of the variance $[\sigma_s^2]^{(l+1)}$ by the weighted sum of level-l variances and the variance of the level-l means.

5.8.3 Continuous Spherical Weiszfeld in Mixtures

We will now show that the results for the continuous attraction $\mathcal{F}_1$ in Section 5.5 can be directly applied to formulate a continuous spherical Weiszfeld (CSW) step. CLOP defines the target position of an iteration step by a convex combination of expectations $E[\hat{\Omega}_{sk}]$, where $\hat{\Omega}_{sk}$ are Gaussian weights defined in Eq. (5.18). On the unit sphere, those weights are now accordingly defined by

$$\hat{\Omega}_{sk} = w_s \; g_w(x|\Phi_s) \; \hat{\alpha}_k(x). \tag{5.32}$$

where w_s are the mixture coefficients of $\mathcal{M}$ as before, g_w is the (one-dimensional) pdf of the WN distribution, and $\hat{\alpha}_k(x)$ is defined as in Eq. (5.17). The sought quantities μ_{sk} and ω_{sk} for computing $E[\hat{\Omega}_{sk}]$ can be obtained by wrapping the Euclidean arrangement of the involved weights (Figure 5.5) onto the unit sphere (Figure 5.9). Although we operate on the 2-dimensional domain $\mathbb{S}^2$, it is sufficient to evaluate these expectations only along 1-dimensional geodesics, on which the wrapped normal distributions Φ_s are defined. Due to the isotropic symmetry of the weighted Gaussian components g_w and $\hat{\alpha}_k$ in Eq. (5.32), the sought mean μ_{sk} always lies on the geodesic γ_s through the particle normal $\mathbf{n}$ and the mean normal μ_s of the s-th mixture component (Figure 5.9b). This allows us to evaluate μ_{sk} and ω_{sk} using an angular parametrization on γ_s. This way, the derived formulations for the mean and weight in Section 5.5 can be directly applied to the 2-dimensional unit sphere setting $d = 2$:

Weight ω_{sk}. The covariance sum Λ_{sk} defined in Eq. (5.20) now simplifies to an isotropic bivariate matrix $\Lambda_{sk} = \lambda_{sk} I_2$ with diagonal entries $\lambda_{sk} = \sigma^2 + \hat{\sigma}_k^2 \, h^2$, and ω_{sk} becomes

$$\omega_{sk} = w_s \, \hat{w}_k \, \hat{\sigma}_k^2 \, h^2 \, \lambda_{sk}^{-1} \, e^{-\frac{1}{2} \frac{d_g(\mu_s, \mathbf{n})^2}{\lambda_{sk}}}. \tag{5.33}$$

Mean μ_{sk}. Similar to the Euclidean case (5.22), we centralize the angular parametrization in the particle normal $\mathbf{n}$. Then the relative mean of the product function $\hat{\Omega}_{sk}$ on γ_s is defined by

$$d_g(\mu_{sk}, \mathbf{n}) = \hat{\sigma}_k^2 \, h^2 \, \lambda_{sk}^{-1} \, d_g(\mu_s, \mathbf{n}). \tag{5.34}$$

Here we can use the same coefficients $\hat{w}_k$ and $\hat{\sigma}_k$ as in CLOP (Figure 5.6b). Since it is not necessary to localize the median seeking of $\mathbf{n}$ on the unit sphere like LOP does in Euclidean space, h can be safely relaxed to a conservative radius $h = \pi$. The actual mean μ_{sk} can now be obtained by interpolating between μ_s and $\mathbf{n}$ on γ_s, i.e.,

$$\mu_{sk} = \mu_s t + \mathbf{n}(1 - t), \tag{5.35}$$

where Eq. (5.34) gives the interpolation factor

$$t = \frac{d_g(\mu_{sk}, \mathbf{n})}{d_g(\mu_s, \mathbf{n})} = \frac{\hat{\sigma}_k^2\, h^2}{\sigma^2 + \hat{\sigma}_k^2\, h^2}. \tag{5.36}$$

Finally and analogously to the Euclidean case (5.19), the Weiszfeld iteration step is given by the weighted sum of the resulting mean normals μ_{sk}, as illustrated in Figure 5.9c,

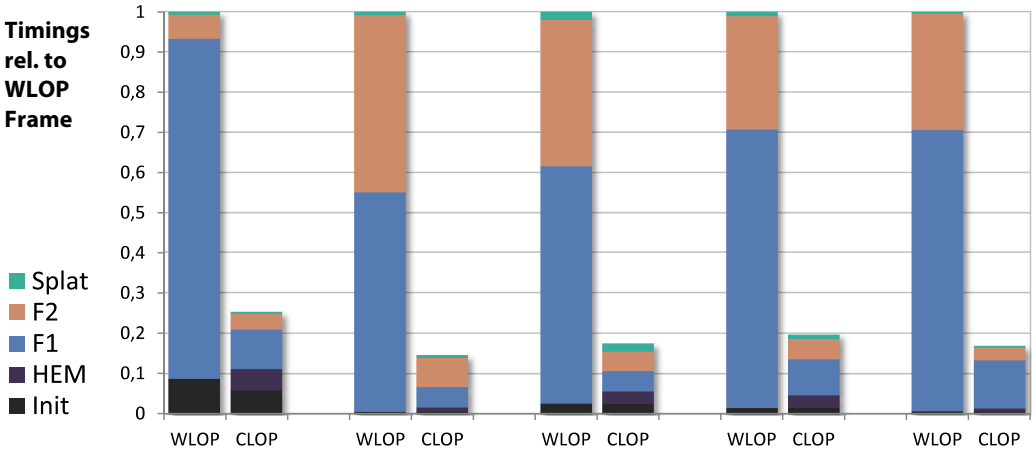
$$\mathbf{n}' = \frac{\sum_s \sum_k \omega_{sk}\, \mu_{sk}}{\left\| \sum_s \sum_k \omega_{sk}\, \mu_{sk} \right\|}. \tag{5.37}$$

5.9 Implementation

In order to achieve interactive reconstruction performance, we have used a screen-space reconstruction technique performing rasterization-based GPU neighbor queries similar to the methods developed throughout Chapter 3 and 4. All local kernel operations in the main stages of the algorithm are executed by quad rasterization on the projected images of the input point set P, the Gaussian component locations of $\mathcal{M}$, and the particle set Q, which are all stored in screen-sized textures. The Gaussian mixture computation, CLOP iterations, optional consecutive CSW (normal estimation) and final rendering are performed in each individual frame based on the input points P projected to these textures.

To obtain a more complete snapshot of the data for more thorough LOP computations, we additionally extend the screen-space reconstruction approach from the previous chapters by a straightforward A-buffering technique, which allows for a more memory-efficient storage of the projected points in the screen. Instead of storing the front-most point mapping to each pixel in one screen-sized texture, we subdivide the viewport into $n \times n$ tiles of dimension $w/n \times h/n$, where each tile represents a depth layer of reduced resolution. When projecting the points to this texture in the initial step, we use the atomic-function capabilities of current graphics hardware to concurrently fill the consecutive layers with the points mapping to the same tile fragment. Since points that would be occluded in a 1×1 layout are now able to switch to another free pixel location, this allows storing a higher number of points at the same texture space. In order to maintain a correct addressing of all neighbors within a given radius of a point in this tiled layout, we modify the query-splat rasterization such that one splat is emitted at the corresponding location for each tile. We have found that in general, the reduced size of the tiles and the projected query splats makes up in performance for the increased number of splats, and have used tilings of $n = 2 - 3$ in all our scenes.

For a full assessment of the performance of our system, we do not currently exploit any frame-to-frame coherence. However, common temporal-coherence approaches could accelerate our system even further [LXJF13]. Note that after normal estimation, we simply orient the normals towards the camera when rendering the reconstructed point cloud. Obtaining globally consistent normals would require global computations like orientation propagation [HLZ+09], which are too expensive for an interactive setting.



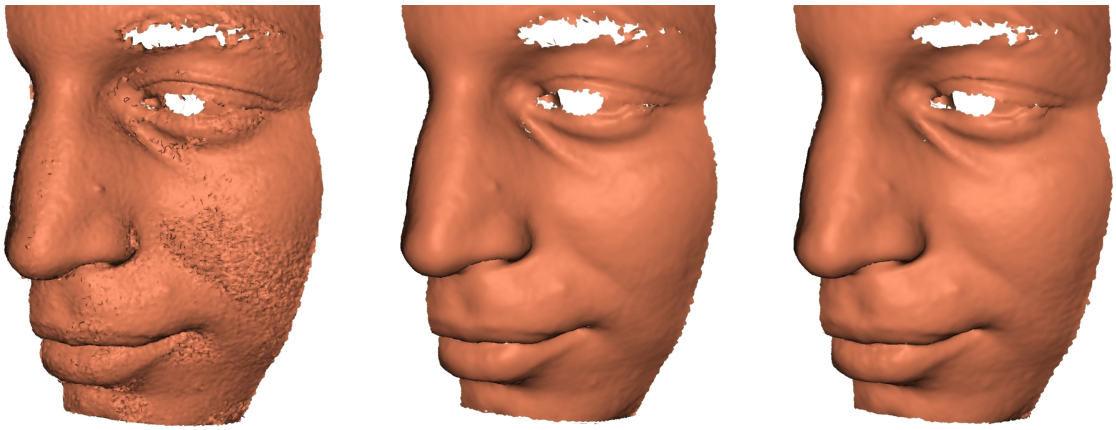
Model	Lena		Face		Camel		Garg. sm.		Gargoyle	
$\|P\|$	74K (74K)		84K (84K)		87K (87K)		77K (78K)		175K (302K)	
$\|\mathcal{M}\|$	5100		8100		7850		10K		32K	
$\|Q\|$	3700		84K		72K		38K		107K	
Iters	50		16		10		20		10	
ms	WLOP	CLOP	WLOP	CLOP	WLOP	CLOP	WLOP	CLOP	WLOP	CLOP
Init	17	11	9	9	17	17	10	10	9	9
HEM		10		18		20		18		11
F_1	161	19	837	77	379	33	423	54	981	167
F_2	11	7	677	110	234	30	172	31	405	42
Total	189	47	1523	214	630	100	605	113	1395	229
SU F_1	5.55		8.81		7.15		5.88		5.51	
SU F_2	1.57		6.15		7.80		5.55		9.64	
SU Tot.	**4.02**		**7.12**		**6.30**		**5.35**		**6.09**	

Figure 5.10: Model statistics and individual timings in ms. Speedups (SU) are given for attraction (F_1) and repulsion (F_2) separately, as well as for the whole CLOP operator compared to a corresponding WLOP GPU implementation. The top graphs give individual timings of each processing stage normalized by the total WLOP time.

5.10 Evaluation and Results

5.10.1 Performance

All results in this chapter were produced on a PC with an Intel i7 4470K 3.5 GHz CPU and NVIDIA GeForce GTX TITAN GPU. We used a framebuffer resolution of 1700×880 in all our performance tests. Fig. 5.10 summarizes statistics and performance measures for the 5 tested models (Lena, Face, Camel, Gargoyle and small Gargoyle) and plots the relative speedup of CLOP over WLOP. The given point set cardinalities $|P|$ denote the



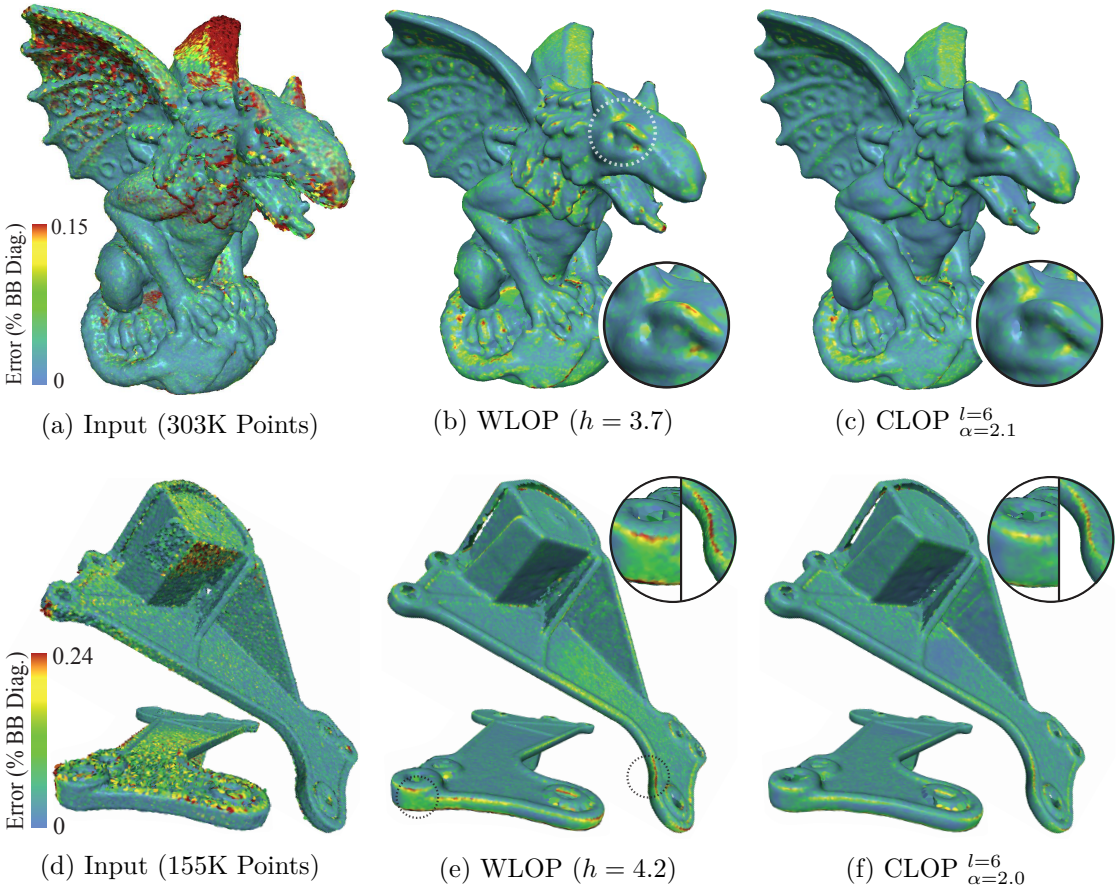
(a) Input (b) WLOP (1523 ms) (c) CLOP (214 ms)

Figure 5.11: (a) Small-kernel splat reconstruction on the Face model showing heavy registration errors. (b) After WLOP (16 iterations). (c) After CLOP with 10% Gaussian components.

points left to operate on *after* A-buffer projection. The *original* number of input points is given in parentheses. Note that since particle positions change over the CLOP iterations, possible A-buffer overflows commonly slightly reduce the number of total particles that finally remain for rendering. The listed particle counts $|Q|$ therefore always denote the average amount over all iterations. The performance numbers include individual timings for creating the mixture model (HEM) and evaluating the continuous attraction compared to the discrete attraction (F_1), as well as the accelerated repulsion compared to the full repulsion computation (F_2). The Gargoyle and Camel point sets were generated with a virtual scanning framework [BLN+13], using 16 to 18 individual scans. Its parameters have been set to generate a realistic but relatively high level of noise and outliers. Details on the used virtual-scanning parameters are given in Appendix A. All results were produced with weighting enabled (WLOP vs. weighted CLOP) since it doesn't incur additional costs on either side. We observe an overall speedup of up to 7 times the WLOP performance, while producing a practically indistinguishable reconstruction. Fig. 5.11 gives a visual comparison for the noisy face model, where CLOP outperforms WLOP by a factor of 7. The results show that even when reconstructing the model with a large number of particles (about the same as the input model), only a low number of Gaussians ($\sim 10\%$) are required to represent the input point cloud, leading to significant speedups in the attraction evaluation (up to a factor of 9).

5.10.2 Reconstruction Quality

In this section, we analyze the reconstruction quality of our method in depth and evaluate its accuracy against the original WLOP algorithm. To allow for an accurate evaluation that is not biased due to the A-buffer based particle loss described above, we use an exact reference implementation for all accuracy measurements. We will show that although our



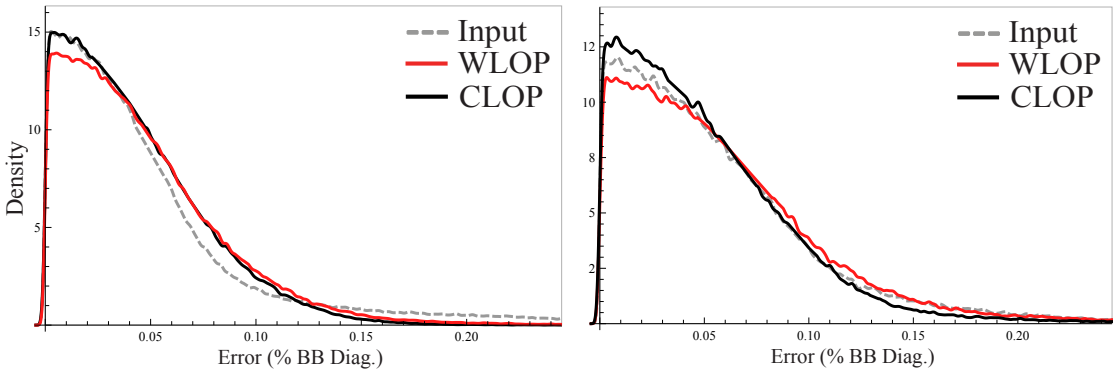
(a) Input (303K Points) (b) WLOP ($h = 3.7$) (c) CLOP $^{l=6}_{\alpha=2.1}$

(d) Input (155K Points) (e) WLOP ($h = 4.2$) (f) CLOP $^{l=6}_{\alpha=2.0}$

Figure 5.12: Reconstruction error on two noisy registered scans (a),(d). Heatmaps compare the surface error between (b),(e) the WLOP and (c),(f) the CLOP reconstruction (#particles = #points, 20 iterations). Kernel sizes h are given in % of the bounding-box diagonal. For both cases, the detail lenses and the respective error distribution functions (g),(h) demonstrate the superior accuracy of our method.

approach runs several times faster than its discrete counterpart, its continuous nature is able to produce a reconstruction of comparable or even better accuracy.

Accuracy. We study the reconstruction error of CLOP vs. WLOP on two models exhibiting different characteristics (Fig. 5.12). To provide exact reference models for measurement, we used the virtual scanning framework by Berger et al. [BLN+13]. Both models were resampled by 16 virtual scans exhibiting a moderate amount of additive Gaussian noise. These were registered using locally weighted ICP [BR07] using a realistic amount of rotational misalignment, which is a common source of outliers. Note that we generated an additional smaller version of the Gargoyle for the performance tests in Section 5.10.1 using the same scanner parameters, but with lower scanner resolution. The left column in Figure 5.12 shows the resulting noisy input point clouds, the middle



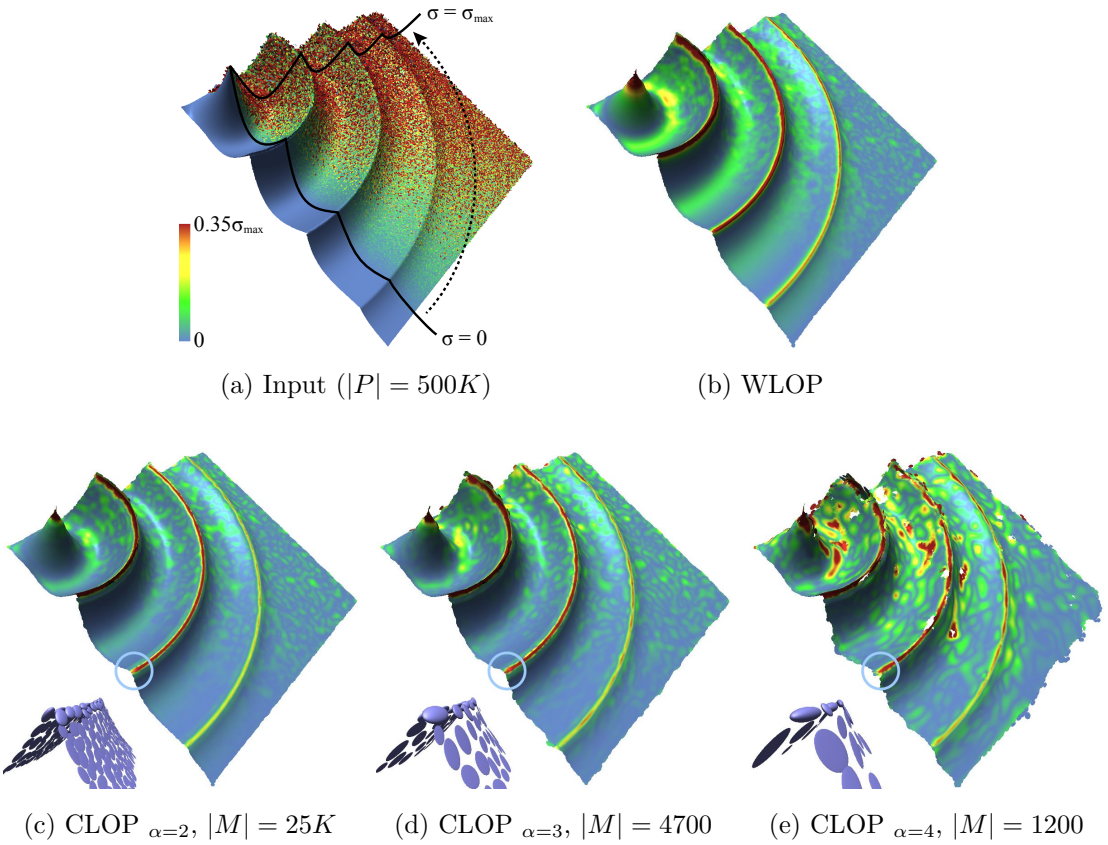
(a) Gargoyle error density

(b) Daratech error density

Figure 5.13: WLOP and CLOP error distribution functions for the models in Fig. 5.12.

and right columns give the WLOP and CLOP reconstructions. Splat colors indicate particle errors $E(q) = \langle q - p, N_p \rangle$, measuring the distance of q to the tangent plane in the nearest reference surface point p, with N_p being its normal. Interestingly, the error heat maps and detail lenses indicate a generally superior behavior of our method over WLOP, especially at regions of high curvature. Only in few isolated regions like at the Gargoyle's ear, the clustering between very close-by misaligned scans leads to a slightly higher error than using WLOP. Fig. 5.13 plots the error density functions (particle error on abscissa vs. density on ordinate) of CLOP against WLOP and the input data for both models. Both graphs show that CLOP produces more low-error particles and less high-error particles than WLOP, thus providing an overall better reconstruction quality.

A detailed error analysis for different geometric cases is given in Figure 5.14. Here we use a synthetic data set (Fig. 5.14a) designed to show varying levels of noise as well as both sharp and smooth features. Figure 5.14b shows the WLOP reconstruction after 20 iterations. As expected, the error is maximal at the sharp edges. The presence of noise in the data leads to a more noisy particle alignment, although only at a very subtle level. However, even where the input data is noise-free, WLOP produces homogeneous regions of error in the curved trenches of the function. This error is explained by particles being repulsed in tangential direction instead of along the curvature of the surface, and is thus less visible as the surface gets more planar. Figure 5.14c shows the corresponding CLOP result at reasonable parameters. Compared to WLOP, we observe a clear reduction of the error regions in both the flat trenches and the sharp edges, which is especially apparent at the conic apex of the function. We suspect that CLOP's overall better quality can be attributed to the continuous attractive energies, which provide a smoother and thus more robust description of the geometry than the singular attractive points used by WLOP. This might positively affect the stability of the attractive particle movements against the perturbing repulsion forces, and thus allow for a more controlled and overall better optimization.

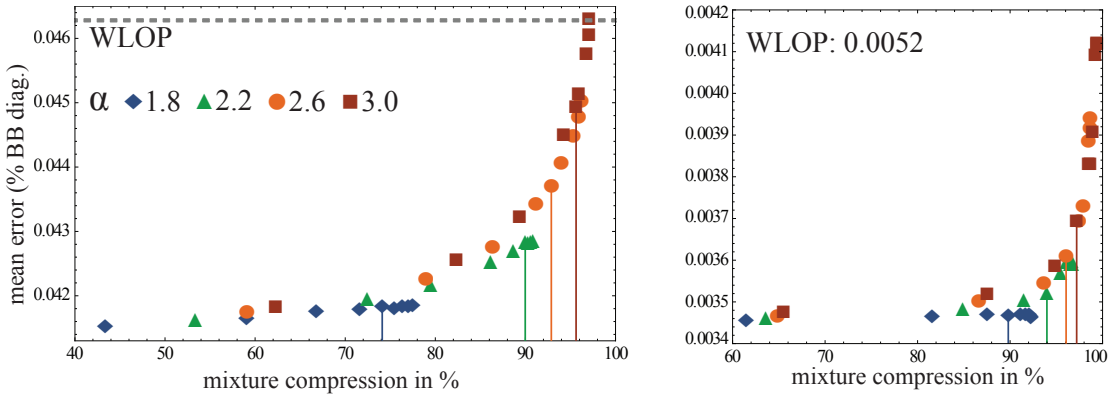


(a) Input ($|P| = 500K$) (b) WLOP

(c) CLOP $_{\alpha=2}$, $|M| = 25K$ (d) CLOP $_{\alpha=3}$, $|M| = 4700$ (e) CLOP $_{\alpha=4}$, $|M| = 1200$

Figure 5.14: (a) Point sampling of a radially symmetric ripple function, containing sharp edges of various angles, and exhibiting Gaussian noise increasing from 0 to σ_{max} with the angle of rotation. We compare the reconstruction error of (b) WLOP and (c)-(e) CLOP at various α and mixture reductions. Heat map colors relate to error amplitudes. Note that with a suggested $\alpha = 2$, we are able to achieve an overall lower error than WLOP.

Effect of Clustering. As our method relies on a reduced representation of the input data, i.e., a Gaussian mixture, we are interested in the effect of this reduction on reconstruction quality, especially in the presence of high frequencies. We thus investigate the reconstruction error for different levels of compression. Figure 5.14 (c) to (e) show CLOP reconstructions at same kernel size and iteration count for increasing levels of mixture compression (increasing α at 8 HEM levels). While for $\alpha = 2$ we have observed an improved accuracy over WLOP, the quality drops with successive levels of compression. Figure 5.14e shows that choosing an extreme mixture compression still achieves moderate reconstruction quality in noise-free regions, but breaks down for stronger noise levels, leading to a corrupt reconstruction with large errors and irregularities in the particle distribution (holes). The insets depict the Gaussians at a sharp edge and show that a sufficiently strict HEM regularization (i.e., small α) produces almost no signal blurring.



(a) Gargoyle (Fig. 5.12)

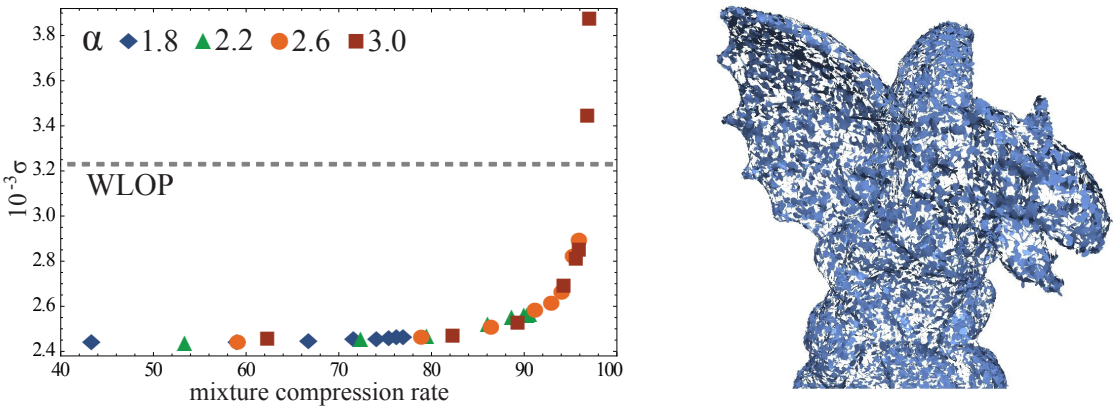
(b) Ripple function (Fig. 5.14)

Figure 5.15: Mean error development with increasing mixture compression, given by various values of α and $1-9$ levels. For each α, the marker lines indicate the level where the additional compression falls below 2.5%. The dashed line gives the corresponding WLOP error (above plot range in (b)).

The reduction of the Gaussian mixture is controlled by the regularization α and the number of clustering levels. Figure 5.15 plots the actual compression rates (abscissa) for the Gargoyle and the Ripple data set against the mean reconstruction error (ordinate, in % of bounding box diagonal) for different values of α and HEM levels. The plots show that in general, the mean error lies clearly below the WLOP error level for reasonable compression rates, and only starts to increase at strong compressions, depending on the complexity of the model: For the Gargoyle, this happens at 80%, while the ripple model, containing more smooth and flat regions, can be compressed up to 94% without significantly increasing the reconstruction error. We also see that for a given compression rate, lower values for α require more HEM clustering levels to achieve the same compression, but for $\alpha \lesssim 2.2$, also bound the compression such that the region of rapidly increasing error is avoided. A lower α also produces a lower error at a given mixture size due to a stronger regularization. For these reasons, we recommend to use an $\alpha \approx 2.0$.

Number of HEM Levels. For a given α, we want to use enough HEM levels to achieve sufficient compression, but also not waste performance on additional levels that do not substantially reduce the mixture further. Figure 5.15 shows that different input models exhibit different compression potential. To take this into account, we abort clustering when the additional compression afforded by a level falls below a given threshold. The markers at the abscissas indicate the final mixture compression for a threshold of 2.5% additional compression.

Point Regularity. In all our experiments, the continuous formulation of the attractive energies has shown to provide an improved sampling regularity of the resulting particles,

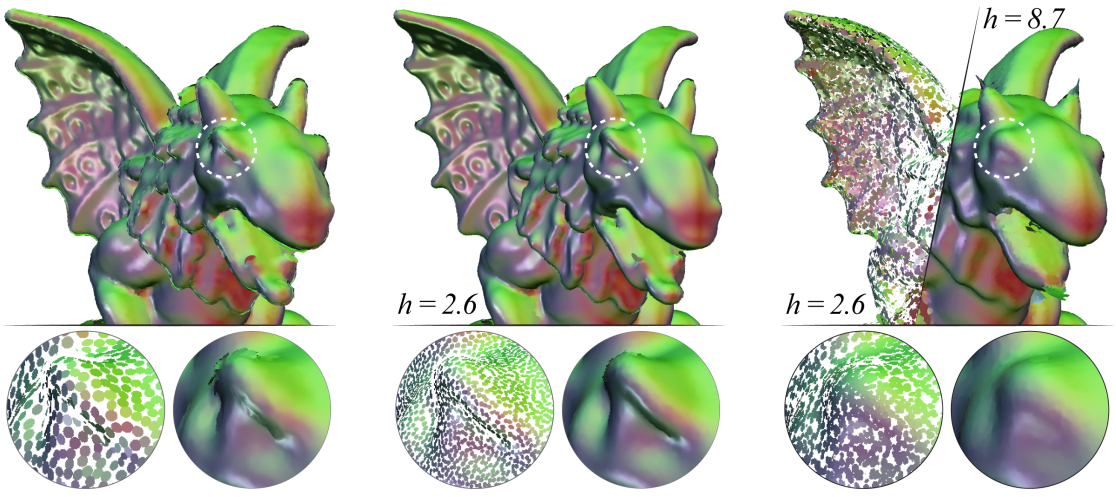


(a) (b)

Figure 5.16: (a) Sampling regularity σ of the Gargoyle model with increasing levels of compression. (b) WLOP on a sub-sampled Gargoyle with same parameters and reduction rate as in Fig. 5.12c.

which we measure by the variance σ of nearest neighbor distances [HLZ+09]. Figure 5.16a plots σ for different α and $1-8$ HEM levels for the Gargoyle from Figure 5.12c. In contrast to the regularity achieved by WLOP (dashed line), the continuous attractive energies allow for smoother particle movements, resulting in a notably lower σ up to a critical point of compression ($\alpha = 3$), where the smooth energy distribution cannot be sufficiently described by the remaining Gaussians anymore. Note that this regularity improvement is independent of the one achieved by interleaved repulsion (Section 5.7.2), which we have activated for both the CLOP and WLOP evaluation.

Usable Amount of Particles. As shown by Lipman et al. [LCOLTE07], the original LOP operator is problematic when using more particles for resampling than there are points in the model. If a given kernel contains too few attractive points to sufficiently describe a smooth energy density, particles tend to collapse into irregular clusters. A larger kernel is able to dampen this effect, but also comes with a much stronger smoothing of the resulting geometry. Therefore, the idea of accelerating WLOP by subsampling the input point set also reduces the usable amount of particles. Fig. 5.12c shows the CLOP reconstruction of the Gargoyle ($303K$ input points, $303K$ particles) using $35K$ *Gaussians*. Fig. 5.16b gives the WLOP result on the model after subsampling to *35K points*. Using the same particle count and kernel size as CLOP, the figure shows that WLOP fails to obtain a sufficient sampling regularity for a faithful reconstruction. In contrast, CLOP even allows for using much more particles than input points, and can therefore also be used to *upsample* a point cloud. For example, based on a sparse sampling of the Gargoyle model with $39.5K$ points (Fig. 5.17a), we used CLOP to project $198K$ particles (500%) into its Gaussian mixture with $22K$ components, using a sufficiently small kernel bandwidth (Fig. 5.17b). As shown in Fig. 5.17c, a similar upsampling with

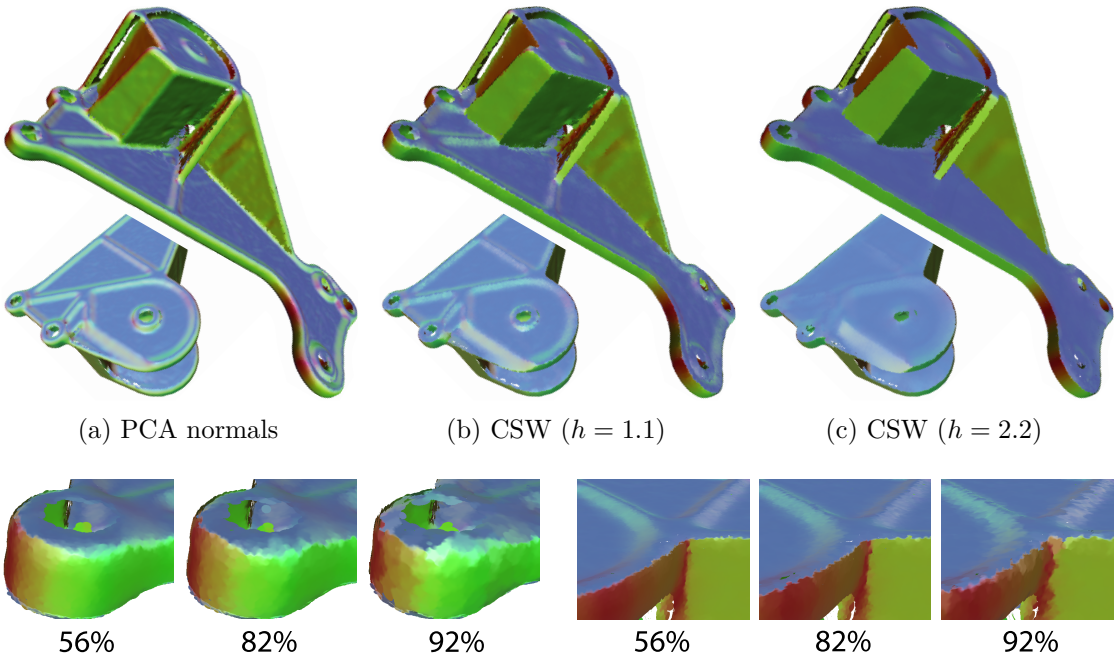


(a) Input ($39.5K$) (b) CLOP ($198K$) (c) WLOP ($198K$)

Figure 5.17: Sparse Gargoyle (a) upsampled to 500% of the input point count, (b) using CLOP ($h = 2.6$), (c) using WLOP ($h = 2.6$ and 8.7). Lenses show splat distributions and surface details for the input, CLOP upsampling, and large-kernel WLOP upsampling. Kernel sizes h are given in % of the bounding box diagonal.

low bandwidth using WLOP leads to particle collapses (left side), while a sufficiently larger bandwidth destroys salient features (right side).

Robust Normals. Finally, we investigate the effect of different CSW kernel sizes on the resulting alignment of the splat normals (Section 5.8.3). A common normal computation method is local tangent plane fitting using PCA [HDD+92], which is quick enough to be suitable for online reconstruction, as shown in Chapter 4, but suffers from typical smoothing artifacts. Figure 5.18a shows the CLOP result of the Daratech data set from Figure 5.12f with PCA normals, where the surface colors encode corresponding splat normals. Note the rounding of the model's sharp edges. Figure 5.18b illustrates the result after applying 13 continuous spherical Weiszfeld iterations on the normals associated with the points resulting from CLOP, where $|\mathcal{M}| = 17.8\%$ of the input point cloud. We used a manually tuned kernel of 1.1% of the BB diagonal, which optimally reconstructs the fine bracings at the model's front and backside. In contrast, a kernel of twice this size flattens subtle features, but also robustly aligns splats on more significant edges more faithfully (Figure 5.18c). Figure 5.18d shows details of the model for various levels of compression ($\alpha = \{1.5, 2.0, 2.5\}$, 5 levels). As in spatial Continuous LOP, an increasing mixture compression leads to a successive reduction in quality. An optimal screen-space reconstruction of the Daratech model with $155K$ points ($135K$ after screen-space projection) requires 124 ms for CLOP + 56 ms for CSW (with kernel size $h = 1.1\%$ for both passes). Figure 5.19 shows a robust online normal alignment of a Kinect stream using CSW.



(a) PCA normals (b) CSW ($h = 1.1$) (c) CSW ($h = 2.2$)

(d) Quality of normals with increasing mixture compression ($h = 1.1$)

Figure 5.18: CSW on the Daratech model. (a) Splat normals using local PCA, (b) and (c) CSW normals with different kernel sizes. (d) Quality reduction of the normals using increasing compression of the spherical mixture (given in % of input model size).

5.11 Limitations

Like its discrete variants, CLOP is a kernel-based operator, and therefore inherits their kernel-related characteristics: larger bandwidths are required in cases of stronger noise, which then leads to stronger smoothing. Since CLOP exploits the fact that dense regions of low curvature can be compressed to very few Gaussians without significant loss of geometric information, the compression potential, and thus the achievable speedup, is always bound by the geometrical complexity of the input data (see Figure 5.15). As discussed in Section 5.4.3, the choice of the regularization term α is a trade-off between clustering efficiency and reconstruction quality. In order to maintain high accuracy, there is a bound on suitable values for the regularization term α, and thus also on the achievable clustering speed. However, the resulting quality also depends on a proper mixture initialization bandwidth as mentioned in Section 5.4.3. The initializing kernel radius r has to be large enough (above noise level) to give the initial Gaussians proper orientation, but should be small enough to prevent from blurring the data in the first place. Currently, we use a density-adaptive radius r, which showed satisfactory results. However, in input data of strongly varying noise, a more flexible, locally noise-aware initialization could probably produce an even better reconstruction quality.

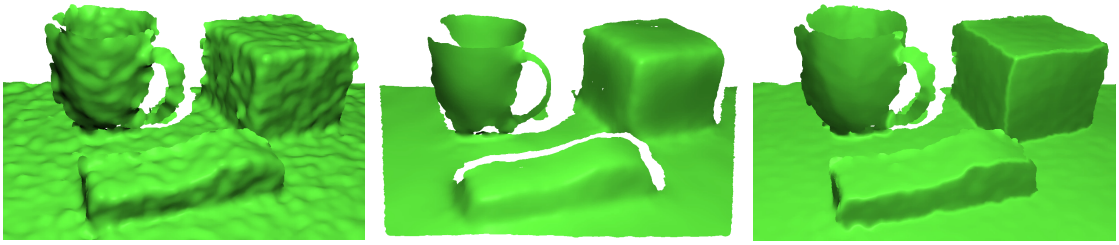
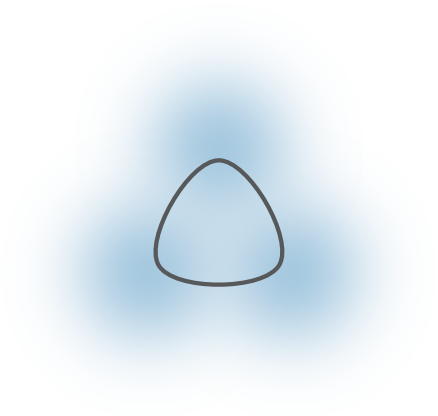


Figure 5.19: Screen capture during online reconstruction of a Kinect stream (left). L_2-based splat reconstruction (middle) results in typical smoothing artifacts, while our robust CSW normal reconstruction (right) faithfully reconstructs sharp features.

5.12 Summary

We have introduced a robust probabilistic point-set resampling and denoising technique that applies the robust Weighted LOP operator to a continuous representation of a point cloud. We showed how to regularize the hierarchical EM algorithm to cluster a point set to a Gaussian mixture in a geometry-preserving manner, and derived an analytic formulation of LOP's attraction forces for this much more compact representation. This way, our method runs several time faster than the original discrete variant, while being able to produce comparable or even superior accuracy for reasonable regularization terms ($\alpha \approx 2$). Furthermore, CLOP provides an overall better sampling regularity, does not put constraints on the number of resampling particles (unlike subsampling approaches based on discrete LOP), and is even capable of point-cloud upsampling. Finally, we have also shown that our continuous formulation can be applied to the spherical domain for efficient robust feature-preserving normal estimation.



And so it is
Just like you said it should be.

— Damien Rice

CHAPTER 6

Gaussian Kernel-Product Surfaces

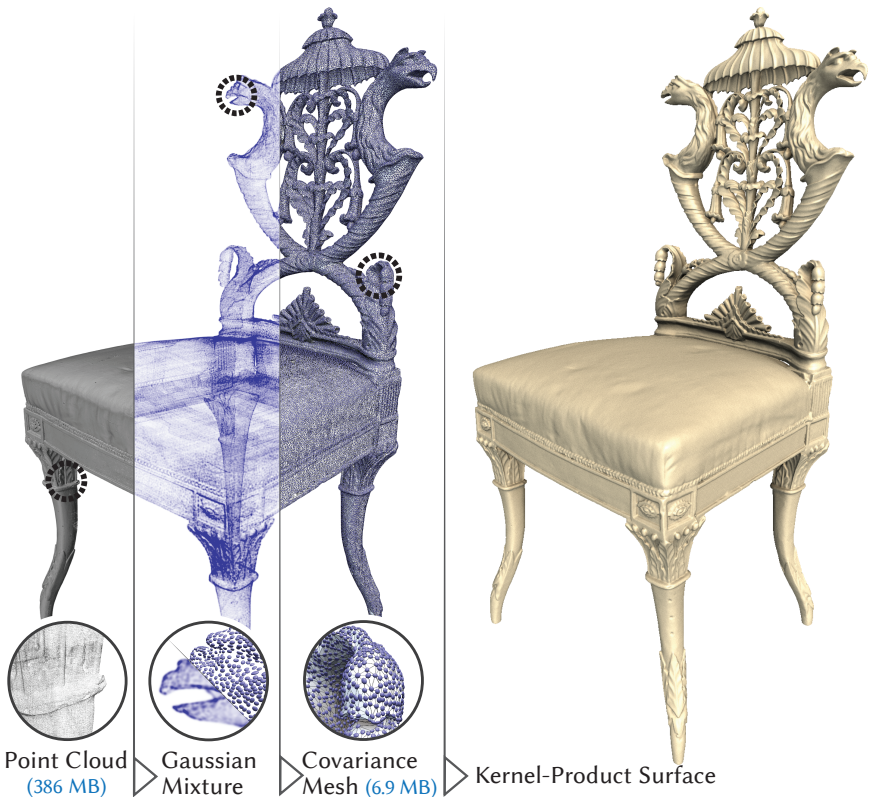


Figure 6.1: A probabilistic surface reconstruction pipeline. Based on a sparse Gaussian mixture modeling the input points' probability distribution, we define a C^∞-smooth surface using a novel interpolation strategy of its individual triangulated Gaussians.

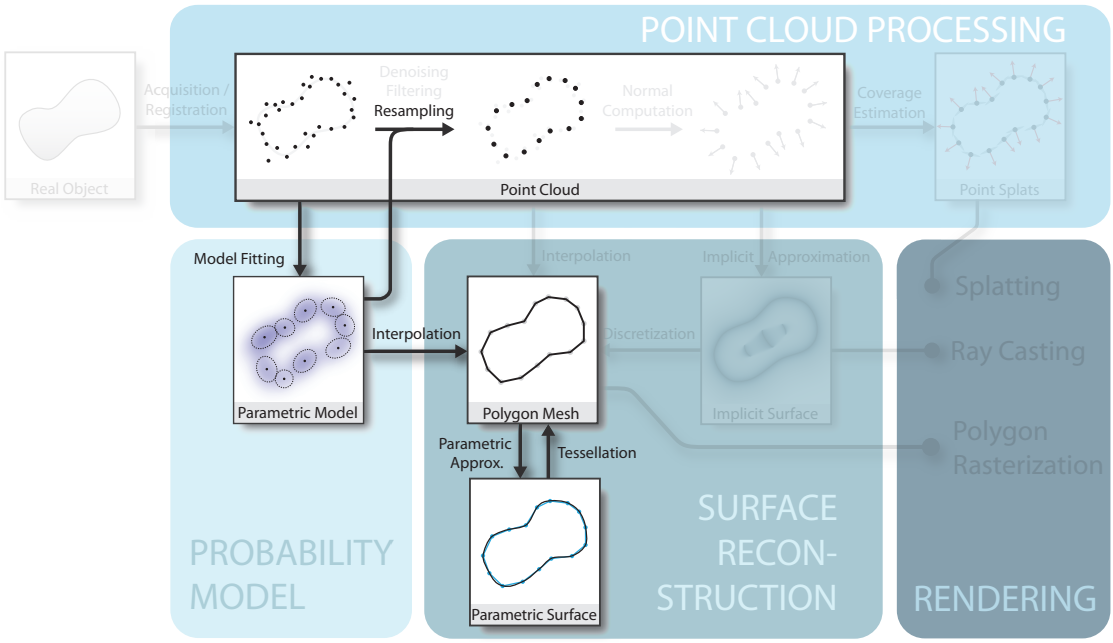


Figure 6.2: We introduce a new processing path for surface reconstruction, that first interpolates the components of a probabilistic model of the data to reconstruct its topological structure as a mesh, and then defines a parametric surface using a new interpolation method that incorporates the probabilistic information given by the model.

6.1 Introduction

In the previous chapter, we have seen that sparse probabilistic models like Gaussian mixtures offer many computational advantages for the processing of 3d point data: They are able to describe the spatial distribution of a point set in a compact, highly memory-efficient way, can naturally encode the uncertainty that is inherent to the data due to noise and imperfect sampling, and raise the potential to improve both the efficiency and accuracy of point-based operators when transferred to the continuous domain. However, up to the present, sparse Gaussian mixture models have mostly served as intermediate representations during the reconstruction process and have not been further used in the representation of the final surface. In this chapter, we therefore investigate methods for defining a smooth surface, accurately representing the shape given by the original point samples, solely based on such a sparse Gaussian mixture. To this end, we develop a new probabilistic technique that allows for efficiently reconstructing and representing large models using only a sparse set of Gaussians, which significantly reduces the required memory footprint and reconstruction time.

Previous probabilistic methods mostly tried to extract the extremal surface that is given by the ridge contour of the pdf of a dense mixture. These methods, however, break down at increasing mixture compression rates, where this ridge degenerates, and are thus unfit

to exploit the simplification power of Gaussian mixture models for defining continuous surfaces. In contrast, we obtain a parametric approximation of these probability ridges using a new *kernel-product interpolation* that is defined over a direct triangulation of the individual Gaussians components. This constitutes a new probabilistic reconstruction pipeline as shown in Figure 6.2. The resulting surface closely resembles the probability density ridge of its mixture pdf, while at the same time ensuring continuous surfaces even for strongly simplified mixtures. This allows for an efficient surface reconstruction from strongly compressed mixtures, reducing the memory storage footprint of a model by over 98% while achieving accuracy levels comparable to Screened Poisson surfaces at significantly lower reconstruction times. Moreover, in contrast to many existing methods, this technique does not depend on the availability of consistently oriented normals, thus providing increased robustness for non-orientable, noisy or deficiently sampled models.

The main technical contributions of this chapter are

- a new interpolation method for anisotropic Gaussians, using interpolation coefficients that act on the *powers* of their pdfs and thereby define a surface along their *Gaussian product means*,
- a new *probabilistic triangulation* of Gaussians that respects the anisotropy of their kernels and outputs a *covariance mesh*, and
- an approach to implement this interpolation in the construction of a *parametric pseudo-manifold* that is based on this covariance mesh, resulting in a consistent probabilistic surface.

6.2 Motivation and Overview

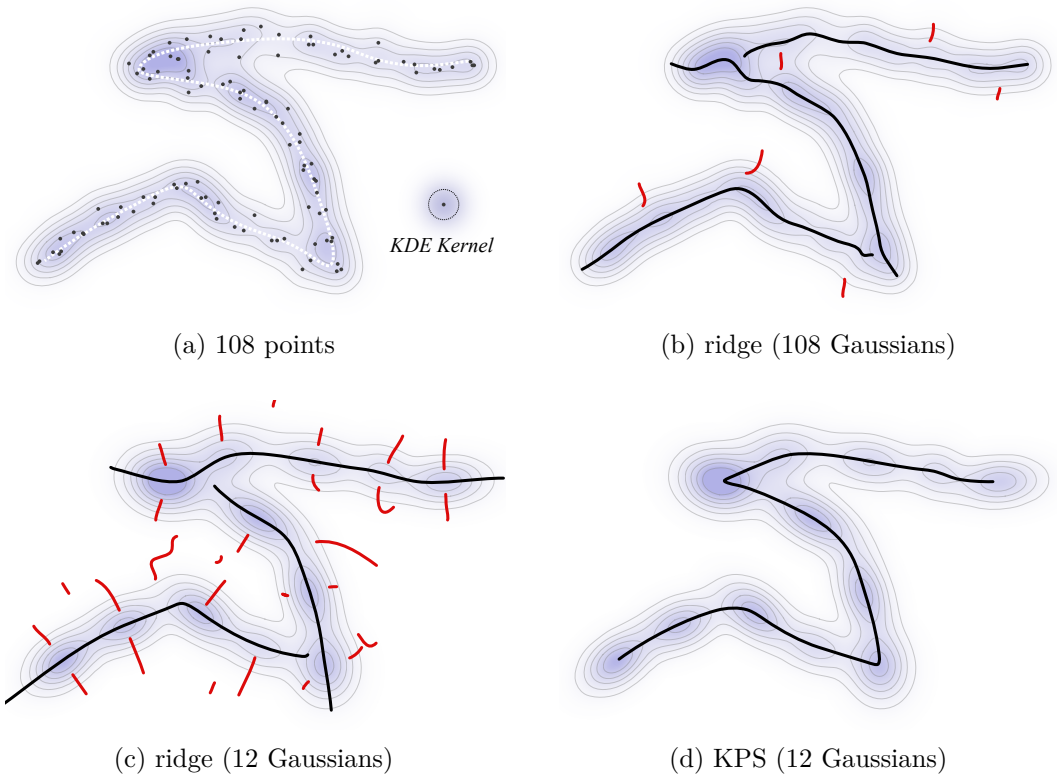
Let P be a set of discrete points sampled from a surface S^* (potentially with noise), and $\mathcal{M} = \{\Theta_i\}$ a mixture of weighted, multivariate anisotropic Gaussians $\Theta_i = (w_i, \mu_i, \Sigma_i)$ computed from P and modeling the probability distribution of its points by an associated probability density function (pdf)

$$f_{\mathcal{M}}(\mathbf{x}) = \sum_i w_i f(\mathbf{x}|\Theta_i), \tag{6.1}$$

with μ_i denoting the mean, Σ_i the covariance, w_i the weight, and

$$f(\mathbf{x}|\Theta_i) = |2\pi\Sigma_i|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{x}-\mu_i)^T\Sigma_i^{-1}(\mathbf{x}-\mu_i)} \tag{6.2}$$

the Gaussian pdf of the i-th mixture component. Our principal aim is to find a *continuous, smooth surface* S that faithfully resembles the shape of the original surface S^* solely based on the probabilistic representation $\mathcal{M}$. An optimal reconstruction with respect to the probability density of $\mathcal{M}$ is one that places the surface S along the *ridge* of the pdf landscape $f_{\mathcal{M}}$, as depicted in Figure 6.3a. Various methods exist that extract



(a) 108 points

(b) ridge (108 Gaussians)

(c) ridge (12 Gaussians)

(d) KPS (12 Gaussians)

Figure 6.3: (a) Surface contour (dashed white) sampled with normal distributed noise ($\sigma = 1.6\%$ of the shape diagonal). Convolving the Dirac distribution of the samples with a Gaussian kernel ($\sigma = 3.2\%$) results in a kernel density estimate (KDE) of their pdf (blue height field). Note that the ground-truth contour runs closely along the ridge of this pdf, where the density is maximial. (b) The actual ridge contour contains spurious ridges (red), and discontinuities at highly curved features (black). (c) A maximum-likelihood simplification of the pdf to fewer anisotropic Gaussians generally smoothes the ridge contour, but reinforces discontinuities and the appearance of spurious ridges. (d) A topology-aware kernel-product interpolation of the simplified Gaussians in (c).

this ridge by finding local maxima in the pdf along trajectories that are given by the smallest negative eigenvector of its Hessian [OE11, SG07, LLP+10]. However, as shown in Figure 6.3b, general Gaussian mixture pdfs can exhibit discontinuities in the smallest Hessian eigenvector field, resulting in discontinuous ridge structures (black lines) and the occurrence of secondary, *spurious* ridges (red lines), which have to be dealt with. Moreover, a maximum likelihood simplification of a dense input mixture to fewer, larger Gaussians of higher anisotropy expectedly smooths the ridge contour, but increases the appearance of discontinuities and spurious ridges (Figure 6.3c). This behavior opposes our intention to exploit the power of Gaussian Mixtures to model the distribution of a large number of input samples in a sparse, efficient way at minimal information loss.

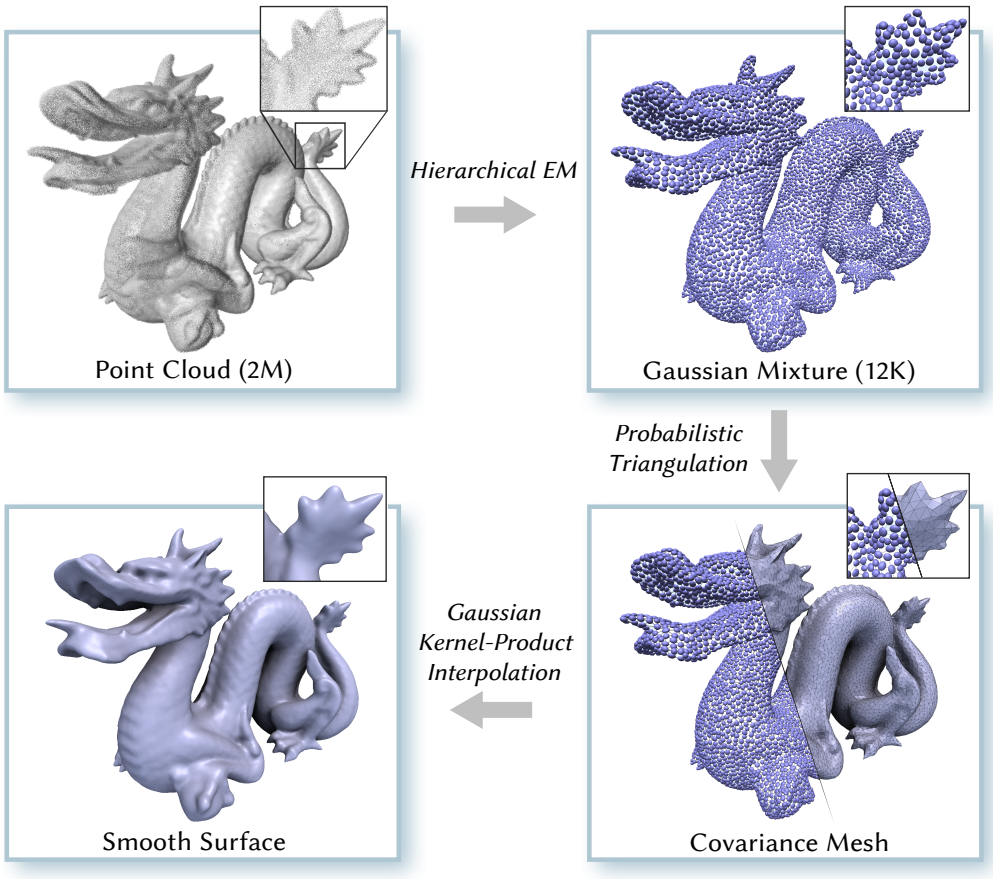
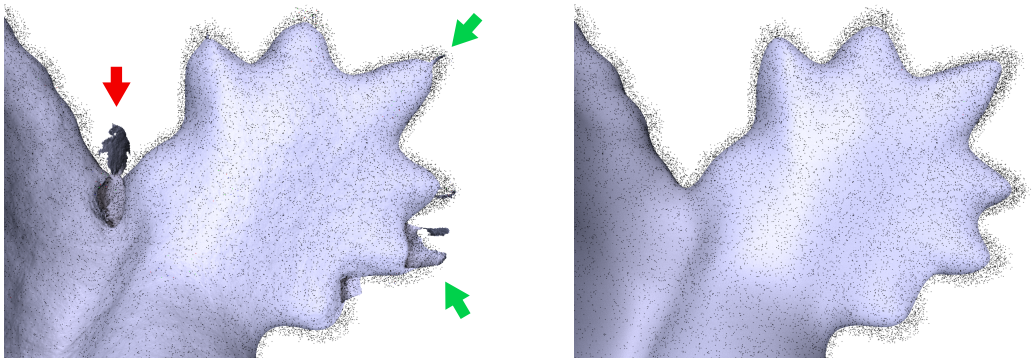


Figure 6.4: Overview of our probabilistic reconstruction pipeline. A noisy point cloud is first converted into a sparse Gaussian mixture. Its anisotropic Gaussians are then triangulated based on a probabilistic similarity measure. From the resulting *covariance mesh*, a smooth surface can be constructed using a new Gaussian interpolation scheme.

In this chapter, we therefore introduce a probabilistic approach, outlined in Figure 6.4, that works on already compressed, sparse mixtures. As described in the previous chapter, these can be quickly computed from given input point sets using hierarchical EM clustering (Section 6.3). Instead of trying to reconstruct an implicit or extremal surface directly from the mixture pdf, we first explicitly determine its topological structure by computing a *triangulation* of the individual Gaussian components, driven by a probabilistic measure of closeness (Section 6.4). This results in a *covariance mesh*, allowing us to construct a smooth surface by applying a novel interpolation of neighboring Gaussian kernels (Section 6.5). We thereby shift the task of finding a continuous surface structure in the pdf landscape to the much more tractable problem of finding a manifold triangulation of sparse anisotropic Gaussians. The resulting *kernel-product surface* (KPS) mimics the shape of the probability ridges while ensuring continuous and artifact-free contours (see Figure 6.3d and 6.5). Our evaluations show that our method competes in accuracy



(a) Ridge Surface [SG07] (b) Kernel-Product Surface

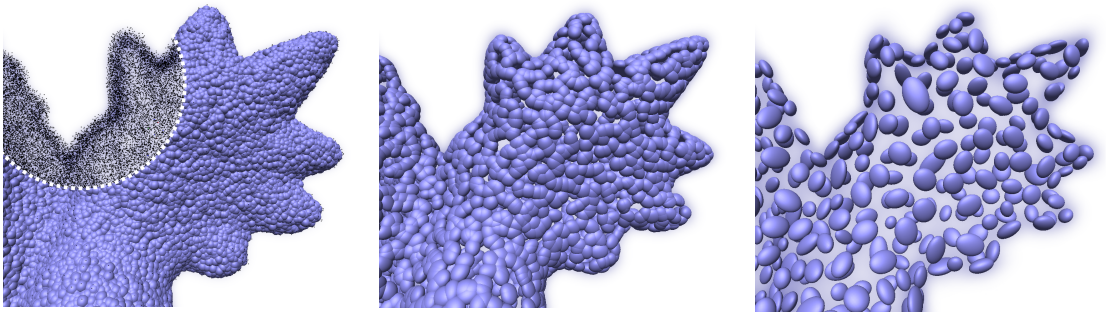
Figure 6.5: Defects of ridge surfaces (a) in high-curvature regions, where a tip collapses to one sheet (green) or a spurious ridge merges with the main ridge (red). (b) Robust surface definition provided by our method.

with non-probabilistic reconstruction methods, while both significantly speeding up the reconstruction process and drastically reducing the memory footprint for representing surfaces (Section 6.6).

6.3 Gaussian Mixture Computation

The input to our reconstruction pipeline is an unordered noisy point cloud $P = \{p_i\}$. It is important to note that our approach does *not* require the availability or explicit precomputation of per-point normals. In the first step, P is converted to a sparse mixture of anisotropic Gaussians $\mathcal{M} = \{\Theta_s\}$ using the geometrically regularized variant of *hierarchical expectation maximization (HEM)* that has been developed in Chapter 5. HEM starts with a dense base mixture (Figure 6.6a), containing one initial Gaussian component per input point, and then hierarchically reduces the number of components by merging close Gaussians in the maximum-likelihood sense. With each consecutive clustering step, the Gaussians' covariance matrices thereby become increasingly anisotropic, such that the plane spanned by their two larger eigenvectors more and more approximates the local balancing plane of the surface (Figure 6.6b and 6.6c). To avoid a too strong degeneration of geometric features modeled by the mixture, we use the same regularization threshold α as introduced in Section 5.4.3, which prevents the clustering of Gaussians that exceed a certain Kullback-Leibler divergence. A larger α allows merging more distant and dissimilarly oriented Gaussians and thus achieves a stronger compression of the mixture. For the models in this chapter, we used values between 1.5 and 2.5. See Section 5.4 and Appendix B for a detailed review of the α-regularized HEM algorithm.

Except for strongly displaced outliers, any Gaussians representing noisy points should be merged during the clustering process in order to obtain a noise-free mixture for a subsequent robust triangulation. However, enlarging α in this regard can have the side



(a) base mixture (2M) (b) HEM level 3 (74K) (c) HEM level 10 (8K)

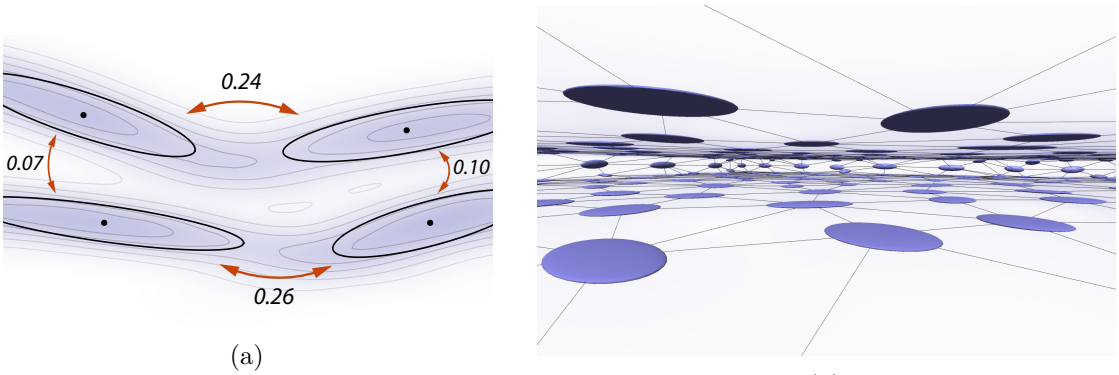
Figure 6.6: Hierarchical EM on the noisy dragon point cloud. (a) The points (detail lens) are converted into small isotropic Gaussians, which are iteratively clustered in the maximum-likelihood sense, thereby successively gaining anisotropy (b and c).

effect of producing a too strong compression and global degeneration of features. Therefore, we instead adapt the initial standard deviation σ_0 of the base mixture Gaussians to the local noise level (denoted by σ_n) in order to provide sufficiently overlapping kernels that ensure merging even for small, constrained α thresholds (Fig. 6.6a). In Figure 6.3a we chose $\sigma_0 = 2\,\sigma_n$ for initializing the 108 Gaussian kernels of the base mixture. After HEM compression, we obtain a noise-free chain of only 12 anisotropic Gaussians, which allows a faithful connectivity determination and subsequent curve reconstruction (Figure 6.3d). In case of strong noise, prefiltering the points additionally reduces the Gaussians' initial dislocation, which allows for even smaller σ_0 and consequently produces a sharper reconstruction. For example, the Gaussian means of the dragon tail in Figure 6.6a have been initialized to the weighted mean of surrounding points using a Gaussian weight kernel of $1.6\,\sigma_n$, allowing for initial kernels of $\sigma_0 = 0.8\,\sigma_n$ to be faithfully merged in the following HEM levels (Fig. 6.6b and 6.6c).

6.4 Probabilistic Triangulation of Gaussians

Our surface is defined over a manifold triangulation of the individual components of a Gaussian mixture $\mathcal{M}$. In this section, we therefore introduce a simple yet efficient way to produce such a triangulation by respecting the probabilistic properties of the Gaussian components. We call the resulting triangulation a *covariance mesh*, as it stores both the means μ_i *and* the covariances Σ_i of the mixture in its vertices.

An ordinary Delaunay-based triangulation, as used for simple point sets, is obviously unsuitable for our purpose, as the anisotropic extent of the individual Gaussian kernels calls for different metrics for the assessment of local distance relations. One way is to approach this problem as a general Delaunay triangulation on a Riemannian manifold, whose local metric is stretched according to the Gaussians' covariance tensors [RLWB16, BLdG+16]. However, such strategies lead to volume meshes, from which the extraction



(a) (b)

Figure 6.7: (a) Thin sheets modeled by 4 anisotropic Gaussians. Black dots indicate their means, ellipses their unit-variance isocontours. Simple distance-based criteria fail to express their topological connectivity. We therefore measure the overlap of their probability extent (numbers over red arrows). (b) Faithful triangulation of a cuboid mixture, where the distribution of Gaussian means does not meet the sampling criterion.

of a surface structure relies on input sampling guarantees we cannot provide in practice, and ends up being even harder than the triangulation problem itself [ACK01]. Therefore, instead of resorting solely to such *geometric* interpretations of the mixture model, we use a triangulation approach that makes use of the *probabilistic* information encoded in the Gaussians.

Greedy triangulation. Our method is based on a simple greedy front-growing triangulation of point sets [CSD04], which starts with an initial seed triangle and then iteratively advances its edge front by adding the next connected triangle Δ_{ijk} that is optimal with respect to a certain plausibility grade $P(\Delta_{ijk})$. For point sets, Cohen-Steiner and Da [CSD04] measure this grade of a candidate triangle geometrically by the reciprocal radius of the smallest empty circumsphere. To avoid the appearance of slivers, where a triangle is added in a way that creates a fold, a negative plausibility grade is assigned if the dihedral angle β_{ijk} between an existing triangle and its connected candidate Δ_{ijk} falls below a certain threshold β_{min}.

Probabilistic plausibility grading. For triangulating a set of anisotropic Gaussians describing the probability distribution of surface points, a Delaunay-based criterion is unsuitable, as the mere Euclidean distance between Gaussians is inconclusive about their topological connectivity. Figure 6.7a demonstrates this on the example of two thin density sheets, represented by 4 Gaussians. While two horizontally aligned Gaussians can have a larger distance between their means, they are still more plausible to be connected, since their individual probability distributions partially model the same region of the surface, i.e., they *overlap* to a considerable degree. Our intuition therefore is to use a

grading that assesses exactly this overlap of their probability distributions. To this end, we use the Bhattacharyya coefficient [Kai67]

$$BC_{ij} = \int_{\mathbb{R}^3} \sqrt{f(\mathbf{x}|\Theta_i)\, f(\mathbf{x}|\Theta_j)}\, \mathrm{d}\mathbf{x} \tag{6.3}$$

between two Gaussians Θ_i and Θ_j as fundamental connectivity measure in our system. This coefficient quantifies the amount of overlap between two statistical populations, has a closed form expression for Gaussians,

$$BC_{ij} = |\widetilde{\Sigma}|^{-\frac{1}{2}} |\Sigma_i \Sigma_j|^{\frac{1}{4}}\, e^{-\frac{1}{8}(\mu_i-\mu_j)^T \widetilde{\Sigma}^{-1}(\mu_i-\mu_j)} \tag{6.4}$$

with $\widetilde{\Sigma} = (\Sigma_i + \Sigma_j)/2$, and has a tractable range $0 < BC \leq 1$, where 1 indicates maximal overlap in case of coinciding distributions. The Bhattacharyya coefficients shown for pairs of Gaussians in Figure 6.7a indicate that they provide a robust measure of topological closeness even for difficult configurations like these thin sheets.

To grade the plausibility of a triangle connecting three Gaussians Ω_i, Ω_j and Ω_k, we request each respective pair of Gaussians to provide sufficient mutual overlap. We thus measure their probabilistic plausibility by

$$P_{prob}(\Delta_{ijk}) = BC_{ij} \cdot BC_{jk} \cdot BC_{ik}. \tag{6.5}$$

However, in addition to this probabilistic grade, we still need to incorporate a geometric regularization to avoid the appearance of slivers in the resulting triangulation. Therefore, we adopt the dihedral angle threshold β_{min} from Cohen-Steiner and Da. Moreover, our intuition is that at similar probability overlaps, the candidate triangle with the larger dihedral angle is more plausible to provide a good triangulation. Therefore, we add a geometric weight

$$P_\beta(\Delta_{ijk}) = \beta_{ijk}/\pi \tag{6.6}$$

based on the smallest dihedral angle β_{ijk} between Δ_{ijk} and its neighboring triangles. Our final plausibility grade is thus given by

$$P(\Delta_{ijk}) = \begin{cases} P_{prob}(\Delta_{ijk}) \cdot P_\beta(\Delta_{ijk}) & \beta_{ijk} < \beta_{min} \\ -\beta_{ijk} & else \end{cases} \tag{6.7}$$

Figure 6.7b shows the inside of a triangulated mixture representing a thin cuboid, where the average distance between neighboring Gaussians lies above the distance between its upper and lower faces. Next, we will use the resulting triangulation of the Gaussian components to define a smooth surface based on a new interpolation formulation.

6.5 Kernel-Product Surfaces

6.5.1 Gaussian Kernel-Product Interpolation

We now develop the interpolation formulation that lies at the heart of our probabilistic surface definition. Let us first look at the basic example of two topologically connected Gaussians Θ_i and Θ_j, shown in black in Figure 6.8a. Our aim is to find a smooth contour connecting their means along a continuous path of highest possible probability density. While the ridge of the pdf (blue contour) would have maximum density along its path, it has problems to meet the continuity requirement, especially close to where the Gaussians overlap. Therefore, we need a curve formulation that closely approximates the ridge where possible, but provides a closed path, specifically in the Gaussian kernel overlap region, where the continuity of the ridge contour tends to break down. To find such a curve, we employ the Gaussian's *joint* distribution, which is given by the product of their individual probability distributions,

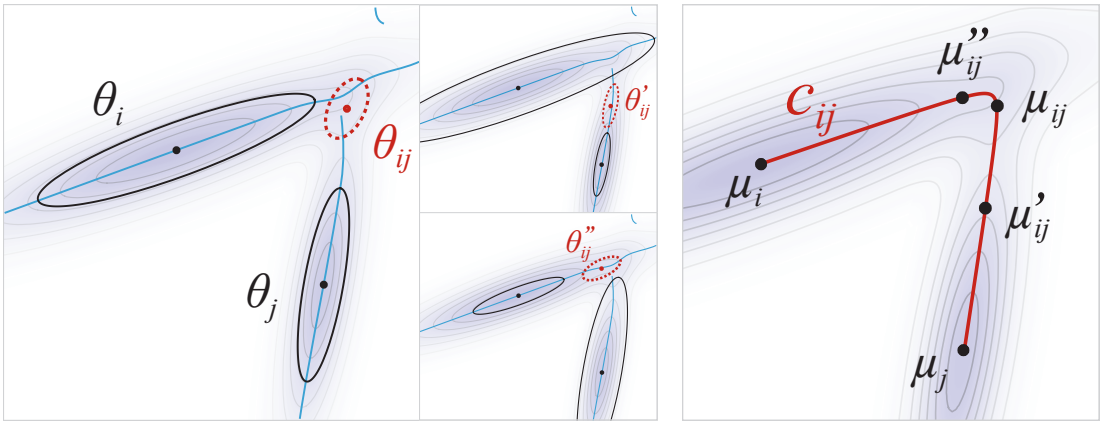
$$f(\mathbf{x}|\Theta_{ij}) = w_{ij}^{-1}\, f(\mathbf{x}|\Theta_i)\, f(\mathbf{x}|\Theta_j) \tag{6.8}$$

and results in the weighted pdf of another Gaussian Θ_{ij} (red dashed). Here, the weight factor $w_{ij} = \int_{\mathbb{R}^d} f(\mathbf{x}|\Theta_i) f(\mathbf{x}|\Theta_j)\, \mathrm{d}\mathbf{x}$ indicates that the product of the two pdfs is generally not a normalized pdf that integrates to 1. The expected value μ_{ij} of this product Gaussian represents the point that is most likely to be part of *both* distributions and can therefore be interpreted as the surface point upon which both Gaussians agree the most. As shown in Figure 6.8a, this product expectation gives a robust and suitable prediction for a surface point even if the ridge contour exhibits a discontinuity.

In order to exploit this quality of the *Gaussian kernel product* for obtaining a complete interpolating contour, we can now modify the statistical certainty of a Gaussian factor distribution by modifying the *power* of its generalized pdf $f(x|\Theta_i)^p$. Increasing p causes a downscaling of its covariance matrix Σ_i by $1/p$, which results in a tighter distribution, increasing the statistical certainty of the Gaussian. In contrast, reducing p upscales its covariance, resulting in a larger distribution of reduced certainty.

Figure 6.8b shows the same configuration of Gaussians as before, but with different powers applied to their pdfs, thus modifying the extent of their probability distributions. As demonstrated by the figure, the kernel product tends to shift towards the Gaussian with the smaller kernel (higher power), and away from the Gaussian with the larger kernel (lower power). In the extremal case, where the power of the first Gaussian approaches zero (infinite covariance), its resulting pdf is a flat function, and the product becomes equal to the second Gaussian. This behavior suggests that we can formulate a continuous interpolation of the Gaussian centroids by relaxing one kernel while tightening the other. To this end, we introduce an interpolation variable that modifies the power of the individual Gaussian pdfs:

$$f(\mathbf{x}|\Theta_{ij,t}) = w_{ij}^{-1}\, f(\mathbf{x}|\Theta_i)^{1-t}\, f(\mathbf{x}|\Theta_j)^t \qquad t \in [0,1]. \tag{6.9}$$



(a) (b) (c)

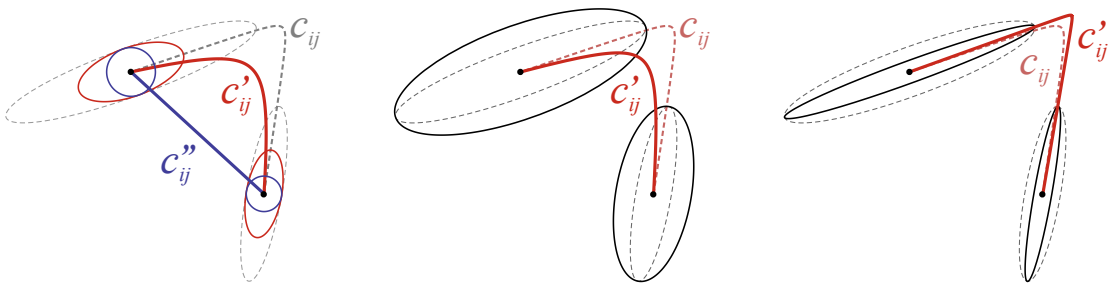
Figure 6.8: (a) Two Gaussians Θ_i and Θ_j and their common product Gaussian Θ_{ij}. Dots indicate their means, the ellipses the unit-variance isocontours of their kernels. The blue lines represent the ridge of the pdf of the mixture $\{\Theta_i, \Theta_j\}$, exhibiting a discontinuity near their pdf overlap. (b) Movement of the product at double (resp. half) the covariance matrix of Θ_i (resp. Θ_j) (upper image) and vice versa (lower image). (c) Resulting contour c_{ij} from continuous kernel-product interpolation.

The final interpolating contour is now obtained by tracing the expectation of the variable product Gaussian $\Theta_{ij,t}$, i.e., its mean

$$\begin{aligned} c_{ij}(t) = E[\mathbf{x}|\Theta_{ij,t}] &= \int_{\mathbb{R}^d} \mathbf{x}\ f(\mathbf{x}|\Theta_{ij,t})\ \mathrm{d}\mathbf{x} \\ &= (t\ \Sigma_j^{-1} + (1-t)\Sigma_i^{-1})^{-1}(t\ \Sigma_j^{-1}\mu_j + (1-t)\Sigma_i^{-1}\mu_i) \end{aligned} \tag{6.10}$$

which results in a rational parametric curve, shown in red in Figure 6.8c. A detailed derivation of a generalized form of Eq. (6.10), which we will use later in this chapter, is given in Appendix C. The resulting curve has several interesting properties, which we will discuss in the following.

Smoothness. In general, the smoothness of the curve is controlled by the ratio of the eigenvalues of the covariance matrices, i.e., their degree of anisotropy. Figure 6.9a illustrates this feature by successively reducing the anisotropic extent of the Gaussians from Figure 6.8a (dashed). With decreasing anisotropy of the kernels, the resulting contour c'_{ij} becomes smoother (red). In the extreme case, where both Gaussians are isotropic, the resulting curve c''_{ij} reduces to a straight line (blue). This behavior also lets the kernel-product contour intuitively reflect the level of uncertainty encoded in the Gaussian covariances. In a sparse mixture representing a noisy point set, stronger noise will result in a larger surface-orthogonal variance of the Gaussian kernels, resulting in reduced anisotropy and smoother contours (Figure 6.9b). In contrast, a low level of noise will create thinner, more anisotropic Gaussians, allowing for sharper features and an



(a) (b) (c)

Figure 6.9: Influence of the shape of Gaussian kernels on their kernel-product contour c_{ij}. (a) Successively reducing their tangential variance straightens out the curve up to the point where c_{ij} is a straight line. (b) Increasing the surface-orthogonal variance produces smoother contours as well, while (c) reducing the variance results in sharper features.

overall more precise reconstruction (Fig. 6.9c). These observations suggest that we can control the smoothness of the reconstruction by reducing the amount of anisotropy in the kernels, as can be achieved by convolving the mixture with an isotropic Gaussian smoothing kernel (Gaussian blurring). Finally, we note that applying a common scale factor to the kernels Σ_i and Σ_j will not affect the curve, as this factor will cancel out in Eq. (6.10).

Weightlessness. While the Gaussian weights w_i affect the additive definition of the mixture pdf (6.1) and are thus crucial for the HEM computation, the expectation in Eq. (6.10) and thus the resulting kernel-product contour is defined solely over Gaussian product pdfs $f(\mathbf{x}|\Theta_{ij,t})$ and is therefore agnostic to any component weight w_i, w_j, or product weight w_{ij}. For the purpose of defining a smooth kernel-product surface, covariance meshes therefore only have to store the Gaussian covariances in their vertices, while the original mixture weights can be dropped.

Continuity. The kernel-product contour defined in Eq. (6.10) interpolates the Gaussian means, i.e. $c_{ij}(0) = \mu_i$ and $c_{ij}(1) = \mu_j$, thus for two curves c_{ij} and c_{jk} meeting in a common Gaussian Θ_j, C^0-continuity is provided. However, they will generally not be tangent-continuous, which becomes more apparent after reformulating Eq. (6.10) to

$$c_{ij}(t) = \mu_i + \Lambda(t)^{-1}(\mu_j - \mu_i), \tag{6.11}$$

where $\Lambda(t) = I + (t^{-1} - 1)\Sigma_i^{-1}\Sigma_j$, and noting that its derivative dc_{ij}/dt always depends on both means μ_i and μ_j for $t \to 0$ and $t \to 1$. Therefore, in the next section we will show how to define C^∞-smooth surfaces by seamlessly blending elementary kernel-product surface patches.

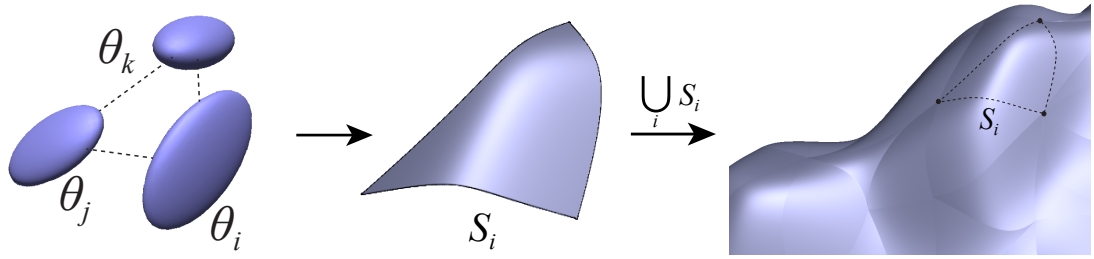


Figure 6.10: 3 Gaussians define a simple kernel-product surface patch S_i, which however only provides C^0 continuity to its neighboring patches.

6.5.2 Constructing Manifold Surfaces

The basic interpolation formulation developed in the previous section can now be extended to create smooth two-dimensional surfaces by incorporating 3 or more Gaussians for which a two-dimensional parametrization can be defined. Adding a third Gaussian factor to the kernel-product formulation in Eq. (6.9) already gives us a simple 3d product Gaussian with pdf

$$f(\mathbf{x}|\Theta_{ijk,s,t}) = w_{ijk}^{-1}\, f(\mathbf{x}|\Theta_i)^{(1-s-t)}\, f(\mathbf{x}|\Theta_j)^s\, f(\mathbf{x}|\Theta_k)^t, \tag{6.12}$$

whose expectation $E[\mathbf{x}|\Theta_{ijk,s,t}]$ forms a rational kernel-product surface patch parametrized by barycentric coordinates (s,t). However, due to the tangent discontinuity property discussed before, such a piecewise continuous parametrization of individual triangles only results in a piecewise smooth surface, as shown in Figure 6.10. On the other hand, a suitable globally continuous parametrization is hard to obtain for meshes of arbitrary topological complexity. Therefore, we construct a *parametric pseudo-manifold* (PPM) [GH95, SXG+09], which allows defining a C^∞-continuous two-dimensional surface by seamlessly blending individual overlapping kernel-product surface patches.

PPM construction. A blueprint for the construction of a PPM is shown in Figure 6.11a. For each vertex Θ_i of a covariance triangle mesh, we create a local planar parametrization $\Omega_i \subset \mathbb{R}^2$ (called *chart*) over its umbrella U_i of one-ring triangles. Each pair of neighboring charts Ω_i, Ω_j overlaps at their two common triangles $U_{ij} = U_i \cap U_j$ (shaded orange). In this overlap region, we establish a *transition map* τ_{ij} from points $x \in \Omega_{ij} \subset \Omega_i$ to points $x' \in \Omega_{ji} \subset \Omega_j$, which enables us to move continuously between the individual parametric spaces. If τ_{ij} satisfies $\tau_{ij} = \tau_{ji}^{-1}$ (bijectivity) and $x = (\tau_{ki} \circ \tau_{jk} \circ \tau_{ij})(x)$ (cocycle condition), Grimm and Hughes [GH95] show that the set of charts Ω_i and transition maps τ_{ij} alone are sufficient to define a manifold M.

Similar to Siqueira et al. [SXG+09], we set up the chart Ω_i of a vertex with valence m such that the parameter coordinates of its associated source vertex i lie in the origin, and its one-ring neighbors j, k, l ... form a regular m-polygon inscribed in the unit circle centered at i. The neighbors are thereby arranged in counterclockwise order starting at angle 0, i.e., the n-th neighbor is located at $(\cos\alpha_n, \sin\alpha_n)$, with angle $\alpha_n = 2\pi n/m$.

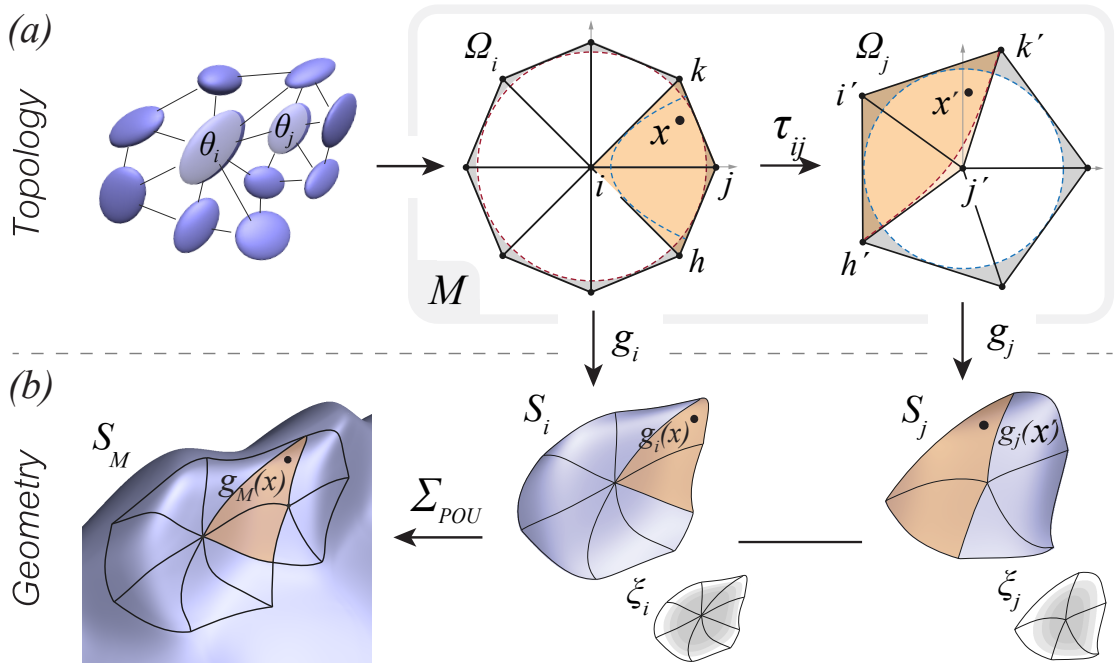


Figure 6.11: (a) A PPM M parametrizes the topological structure of a given covariance mesh using a set of local planar parametric charts Ω_i and a set of transition maps τ_{ij} allowing a continuous movement between the individual charts within their common domain (orange). (b) M is given continuous geometry by defining over each chart a local parametric surface patch S_i and a localizing weight ξ_i which compose the patches together to a complete surface S_M by partition of unity.

Since umbrellas of different valence disallow a simple congruent mapping of their common overlap region, the transition map τ_{ij} performs an angular distortion on both triangles $\Delta ijk \subset \Omega_i$ and $\Delta j'k'i' \subset \Omega_j$ to bring them into common equilateral form where the transition between coordinate frames can occur. To this end, τ_{ij} is composed of several atomic transformations:

$$\tau_{ij} = R_{ji}^{-1} \circ \varphi_j^{-1} \circ \rho \circ \varphi_i \circ R_{ij}. \tag{6.13}$$

First, a rotation R_{ij} is applied to the triangle $\Delta ijk \subset \Omega_i$, such that the neighbor vertex j associated with the transition target is aligned with the x-axis at angle 0. Given the polar form (θ, r) of any point $x \in \Delta ijk$, we can then perform an angular stretching

$$\varphi_i(\theta, r) = (\theta \cdot m/6, r \cdot \cos(\pi/6)/\cos(\pi/m)) \tag{6.14}$$

which transforms this triangle into equilateral form. Now we are able to apply a congruent map $\rho(x) = (1 - x_1, -x_2)$ to reflect the coordinates of $x = (x_1, x_2)$ to its corresponding equilateral location in the chart of the neighbor j. Finally, the transition triangle in Ω_j is transformed back to its original shape $\Delta j'k'i'$ using the inverse stretching φ_j^{-1}, and

rotated to its original angle using R_{ji}^{-1}. Note that since the cocycle condition requires the counterclockwise ordering of the neighbors k', i', h' in Ω_j to be inverse to the ordering of h, j, k in Ω_i, the actual change of coordinates by ρ represents both a horizontal reflection (from i to j along the x-axis) and a vertical reflection, flipping the overlapping double triangles upside down to be in correct order and position when rotated back in Ω_j.

Adding geometry. The PPM constructed above solely defines the topological structure of the manifold M, i.e., a mapping between the charts. To define the geometry of a surface parametrized by this PPM, we can now equip its individual charts with an associated geometry function $g_i : \Omega_i \mapsto \mathbb{R}^3$, defining a local surface patch S_i, and a locally supported weight function ξ_i, defining for each parametric point $x \in \Omega_i$ the contribution of $g_i(x)$ to the final manifold surface S_M. Any point x of the i-th chart can then be mapped to its final surface point $g_{M,i}(x)$ by collecting the contributions from overlapping neighboring charts Ω_j via the transition maps τ_{ij} and blending their surface patches together as partitions of unity (Fig. 6.11b). The final surface point is thus defined as

$$g_{M,i}(x) = \sum_{j \in T(x)} \omega_{ij}(x)\, g_j(\tau_{ij}(x)) \tag{6.15}$$

where ω_{ij} are convex combinations of the weight functions ξ_j with overlapping support,

$$\omega_{ij}(x) = \frac{\xi_j(\|\tau_{ij}(x)\|)}{\sum_{j' \in T(x)} \xi_{j'}(\|\tau_{ij'}(x)\|)}, \tag{6.16}$$

$T(x)$ is the index set of patches overlapping at x, i.e., the vertices of the surrounding triangle, and we presume that $\tau_{ii} = id$.

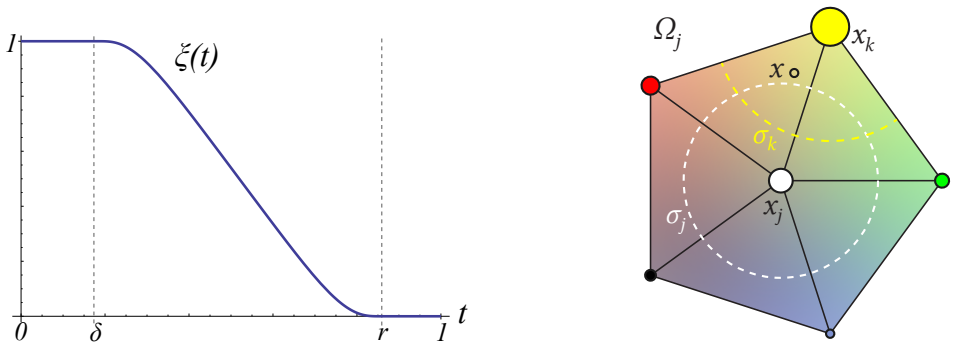
Weight function. The weight ξ_i is a compactly supported, radially symmetric and smoothly decaying function centered in the origin of the chart Ω_i. Note that in order to achieve a seamless blending of neighboring charts, its support must not exceed the radius $r = \cos(\pi/m)$ of the circle inscribed to the m-sided one-ring polygon (dashed circles in Fig. 6.11a). We adopt the weight from Siqueira et al. [SXG+09], which is given by

$$\xi(t) = \begin{cases} 1 & t \le \delta \\ 0 & t \ge r \\ 1/(1+e^{2\,s}) & \delta < t < r \end{cases} \tag{6.17}$$

where

$$s = (1-h)^{-\frac{1}{2}} - h^{-\frac{1}{2}} \qquad \text{and} \qquad h = (t-\delta)/(r-\delta).$$

This function is C^∞-smooth, thus allowing for a seamless blending of surface patches without constraining the continuity order of the resulting surface. The constant δ specifies the offset of the blending range and is set to $r/5$ in all our examples. Fig. 6.12a illustrates the curve and its ranges, where, by definition, $0 \le \delta \le r < 1$.



(a) (b)

Figure 6.12: (a) Plot of the weight function ξ. (b) Distance-based power RBF ψ of each Gaussian for a given parameter point $\mathbf{x}$. Dashed circles show the standard deviations of the RBFs.

Geometry function. The local geometry function g_i over each chart is defined using our new kernel-product surface formulation. Given a mesh umbrella U_i parameterized by a planar chart Ω_i, we need to set up a kernel product of all Gaussians $\Theta_{j\in J}$, $J = \{i \cup \mathcal{N}(i)\}$ of the umbrella:

$$f(\mathbf{x}|\Theta_{J,x}) = w_J^{-1} \prod_{j\in J} f(\mathbf{x}|\Theta_j)^{\psi_j(x)} \tag{6.18}$$

with weight w_J analogous to Eq. (6.8). Here, ψ_j represents a function $\Omega \mapsto \mathbb{R}$ that assigns a power to the j-th Gaussian based on the parametric point $x \in \Omega_i$. This power should be maximal at the chart coordinates x_j of Θ_j, and continuously fall off with increasing distance to x_j. In contrast to the case of 2 and 3 Gaussians, a generalized barycentric interpolation similar to Eq. (6.9) and (6.12) is infeasible for this function due to the non-cyclic layout of the vertices in the chart. Therefore, we model the power functions of Gaussians Θ_j using radial basis functions centered in their chart locations x_j. To ensure a continuous surface, the support of these RBFs has to cover its neighboring vertices in the umbrella, since a two-dimensional kernel-product surface requires everywhere at least 3 Gaussian factors with non-zero power. On the other hand, ψ_j should be small enough at its neighbors x_k to minimize smoothing due to too closely overlapping bases. We thus use a Gaussian RBF

$$\psi_j(x) = e^{-\|x-x_j\|^2/\sigma_j^2} \tag{6.19}$$

and adjust its bandwidth σ_j according to the parametric length s of the longest edge incident to x_j in the chart. This way, every umbrella Gaussian Θ_j obtains a power $\psi_j(x)$ depending on the parametric distance of their chart position x_j to the parameter point x, illustrated by the size of the colored disks in Figure 6.12b. Note that $s = 1$ for the center vertex of any chart and for all vertices in umbrellas with $m \geq 6$ neighbors, while in umbrellas with 3, 4 and 5 neighbors, it corresponds to the the side length of the boundary

polygon. We recommend to set $1/2\,s \leq \sigma_j \leq 3/4\,s$, and have used $\sigma_j = 0.6 \cdot s$ in all our examples.

With the interpolating power $\psi_j(x)$ at hand, the geometry function g_i associated with any parametric point $x \in \Omega_i$ is now given by the expectation of the kernel-product pdf, analogous to Eq. (6.10):

$$\begin{aligned} g_i(x) = E[\mathbf{x}|\Theta_{J,x}] &= \int_{\mathbb{R}^3} \mathbf{x}\ f(\mathbf{x}|\Theta_{J,x})\ \mathrm{d}\mathbf{x} \\ &= \Big(\sum_j \psi_j(x)\ \Sigma_j^{-1}\Big)^{-1}\Big(\sum_j \psi_j(x)\ \Sigma_j^{-1}\mu_j\Big). \end{aligned} \tag{6.20}$$

A detailed derivation of the resulting matrix form of Eq. (6.20) is given in Appendix C. Note that by caching the inverse covariances Σ_j^{-1} and matrix-vector products $\Sigma_j^{-1}\mu_j$ of each Gaussian, this function can be efficiently evaluated using linear combinations and only one matrix inversion per evaluation point x.

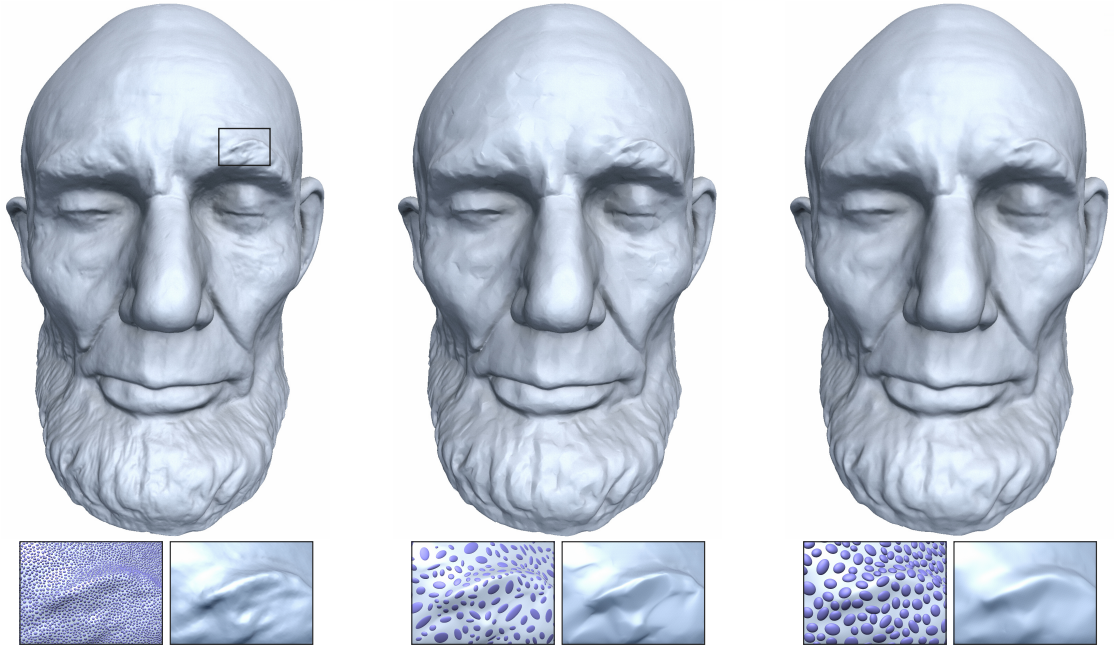
Boundaries. In case of bounded covariance meshes, boundary vertices V_i, which only contain partial mesh umbrellas, require special chart layouts. In order to ensure a sufficient parametric distance between the chart positions of its two neighboring boundary vertices, the chart of V_i is set up in such a way that the opening angle of its partial parametric fan of neighbors does not exceed a half circle. To this end, we simply modify the angular arrangement of its neighbors, such that the n-th neighbor in the chart is placed on the unit circle at angle $\alpha_n = \pi n/m$.

6.6 Results and Discussions

In this section we study the effect of different HEM parameters on the resulting surface, analyze time and memory consumptions of our method for different levels of compression, and assess its quality and robustness in the presence of noise. We will finish with a discussion of limitations and further applications of our probabilistic reconstruction technique. All reconstructions were produced on a PC with 32 GB RAM and Intel i7 3.5 GHz CPU. The kernel-product surfaces (KPS) shown in this section were rendered by tessellating their covariance-mesh triangles to screen resolution at render-time by sampling their geometry function (see Eq. (6.20)).

6.6.1 Reconstruction Performance

Effect of HEM parameters. As discussed in Section 6.3, the mixture compression strength achieved by the initial HEM procedure depends on both the initial kernel size σ_0 and the regularization threshold α, which controls the anisotropic merge range of Gaussians. A larger α allows clustering more dissimilar Gaussians with respect to their Kullback-Leibler divergence, while a larger σ_0 decreases this divergence and thus allows more Gaussians to be clustered as well. We are therefore interested in the effect of these parameters on the resulting surface quality.



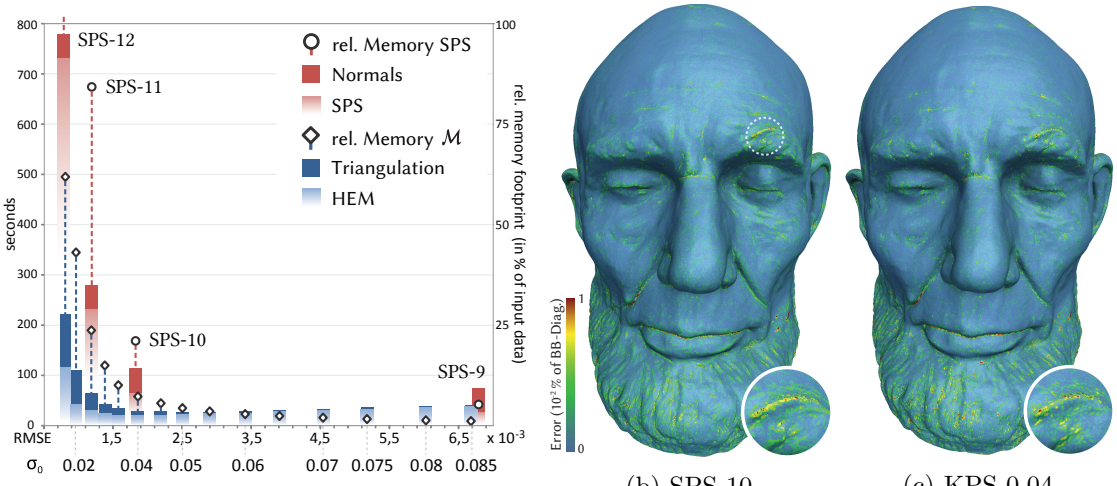
(a) $\sigma_0 = 0.02,\ \alpha = 2$
$|\mathcal{M}| = 1.4M$
compr = 43.1%

(b) $\sigma_0 = 0.02, \alpha = 2.8$
$|\mathcal{M}| = 32K$
compr = 1.0%

(c) $\sigma_0 = 0.085,\ \alpha = 2$
$|\mathcal{M}| = 38K$
compr = 1.2%

Figure 6.13: Effects of increasing the HEM parameters α and σ_0 on the resulting surface. (a) Small σ_0 and α allow only limited compression, but high detail preservation. (b) Increasing α produces a stronger compression, but also highly anisotropic Gaussians, resulting in flattening artifacts. (c) Increasing σ_0 increases the compression as well, but produces a smoother surface.

Fig. 6.13a shows a detailed KPS reconstruction of the Lincoln point set (9.8M points) with $\sigma_0 = 0.02$, which is about half the average nearest-neighbor distance in the input point cloud (all values of σ_0 are given in % of the model diameter). Using $\alpha = 2$, HEM clustering converges to a mixture of 1.4M Gaussians, which represents only a moderate level of compression, since one Gaussian requires 9 floats to store its mean and covariance, which equals the memory footprint of 3 points. The detail lens reveals the shape and distribution of Gaussians as well as the resulting surface at the eyebrow. Increasing α increases the compression, but since this allows more distant and dissimilar Gaussians to be merged during HEM clustering, the resulting Gaussians become highly anisotropic, exhibiting a flatness that corresponds to the small initial kernel size σ_0 (see Fig. 6.13b). However, as visible in the detail view, such highly anisotropic Gaussians can lead to flattening effects of the surface around their means and produce sharper creases due to the anisotropic interpolation behavior shown in Fig. 6.9. In contrast, Fig. 6.13c demonstrates that keeping α low and increasing σ_0 instead (here to about twice the average nearest-neighbor distance) also increases the level of compression, but results in more mollified Gaussians, and consequently a smoother surface. In the following, we



(a) Performance for varying σ_0

(b) SPS-10
113 s – 23.7 MB

(c) KPS 0.04
30 s – 8.2 MB

Figure 6.14: (a) KPS reconstruction performance for different compressions of the Lincoln model at $\alpha = 2$ (blue) compared to SPS reconstructions at different octree levels (red). Timings are plotted over the achieved RMS error. The dashed anchors indicate the memory reduction over the input data for the given covariance and SPS output meshes. Note that the relative memory factor for SPS-12 lies at 288%. (b) and (c) Error distribution over the surface for KPS and SPS at similar error levels.

will therefore only vary σ_0 to analyze the reconstruction quality of different compression rates. For the purpose of an unbiased analysis, we also do not apply a prefiltering of the initial Gaussian locations for the results shown in this section, except for a discussion of limitations (Section 6.6.3).

Reconstruction time, accuracy and compression. To assess the performance of our technique, we analyze the computation timings, reconstruction errors, and the achieved levels of compression over varying σ_0. We put them in context with comparable Screened Poisson Surfaces (SPS) [KH13], representing the state-of-the-art reference in high-accuracy reconstruction, where similar to our mixture compression, different octree levels result in different levels of smoothing and output mesh sizes. For the comparisons, we have used a multi-threaded C++ implementation of our method, and the multi-threaded reference implementation of SPS, using 8 threads for both methods. Fig. 6.14a plots the KPS reconstruction times of the Lincoln model for varying σ_0 (blue bars) and the SPS timings for octree depths 9 to 12 (red bars) over the RMS error to the input point set achieved by the respective reconstruction. To determine the SPS error, we measured the RMSE over all vertices of the resulting trimmed Poisson mesh. The KPS error was computed using 36 samples per parametric triangle. KPS timings are broken down to mixture computation (HEM) and triangulation times. SPS timings include the precomputation of oriented normals and the actual SPS reconstruction. The plots indicate that for a given level of accuracy, our method is generally much faster than a

comparable SPS reconstruction. Fig. 6.14b and 6.14c show the error distributions over the surface in false colors for two results of SPS and KPS at similar levels of accuracy, where our method is almost 4 times faster. Fig. 6.14a also plots the relative memory footprints of the resulting mixtures (in % of the input data size) over their reconstruction error. We compare them to the memory footprints of the vertices of the respective SPS output meshes, as these represent the minimum amount of information defining an SPS result. The plot shows that for a surface of similar accuracy, our representation requires only a fraction of the memory footprint of a corresponding SPS.

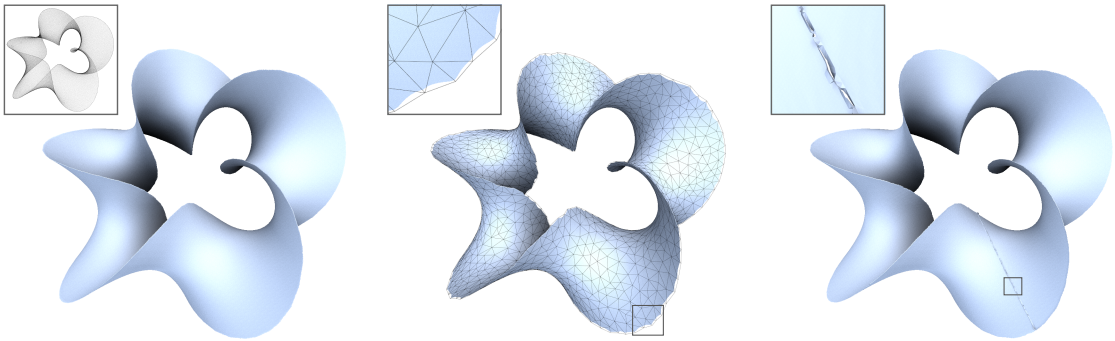
Measuring the reconstruction performance by $(time \times error)$, the optimal level of compression is achieved around $\sigma_0 = 0.035$, which is about the average nearest-neighbor spacing of the input points. For $\sigma_0 > 0.05$, the time required for triangulation becomes negligible, which is due to the number of Gaussians being already reduced to about 1% of the number of input points. At the same time, we observe a slight increase in the HEM computation timings, since during clustering, larger kernel sizes require larger radii for the spatial neighborhood queries, while at such high compression rates, further simplifications do not significantly reduce the clustering effort any more. Table 6.1 gives a complete list of the dataset sizes, HEM parameters, compression rates and reconstruction timings for the different models in this chapter. The parameter σ_0 was mostly chosen to fit the local point density and/or noise level, while α has been kept below 2.0. While these represent rather moderate parameter values, we have achieved memory compression rates down to under 2%.

	$\lvert P\rvert$	σ_0	α	$\lvert\mathcal{M}\rvert$	MB_P	$\text{MB}_\mathcal{M}$	compr.	HEM	triang	total
Chair	33.8 M	0.036	2.0	200 K	386.3	6.9	1.8 %	152	10	162
Castle	32.8 M	0.03	1.8	316 K	374.9	10.8	2.9 %	129	25	154
Lincoln	9.8 M	0.04	2.0	239 K	111.9	8.2	7.3 %	21	9	30
Dragon	2.0 M	0.3	1.3	12 K	22.9	0.4	1.8 %	14	2	16
Gargoyle	303 K	0.3	1.8	7.3 K	3.5	0.3	9.4 %	2	2	4
Moebius	141 K	0.5	1.6	1.3 K	1.6	0.04	2.7 %	0.6	0.1	0.7

Table 6.1: Dataset sizes and memory footprints (in MB) for the input point sets P and covariance meshes $\mathcal{M}$, HEM parameters (σ_0 in % of the model diameter), achieved compression rates and computation times (in seconds) for the models in this chapter.

6.6.2 Discussion of Quality and Robustness

Orientability. Our method can naturally handle non-orientable surfaces and models where noise and bad sampling conditions prevent a consistent orientation of normals. The inset in Fig. 6.15a shows a point set sampling a five-twist Moebius strip. Similar to Simple Point Set Surfaces (SPSS) [AA04], the KPS reconstruction is independent of the availability of orientable normals and can thus produce a continuous surface (Fig. 6.15b). Other PSS variants that rely on properly oriented normals [AA09, OGG09, GAB12,

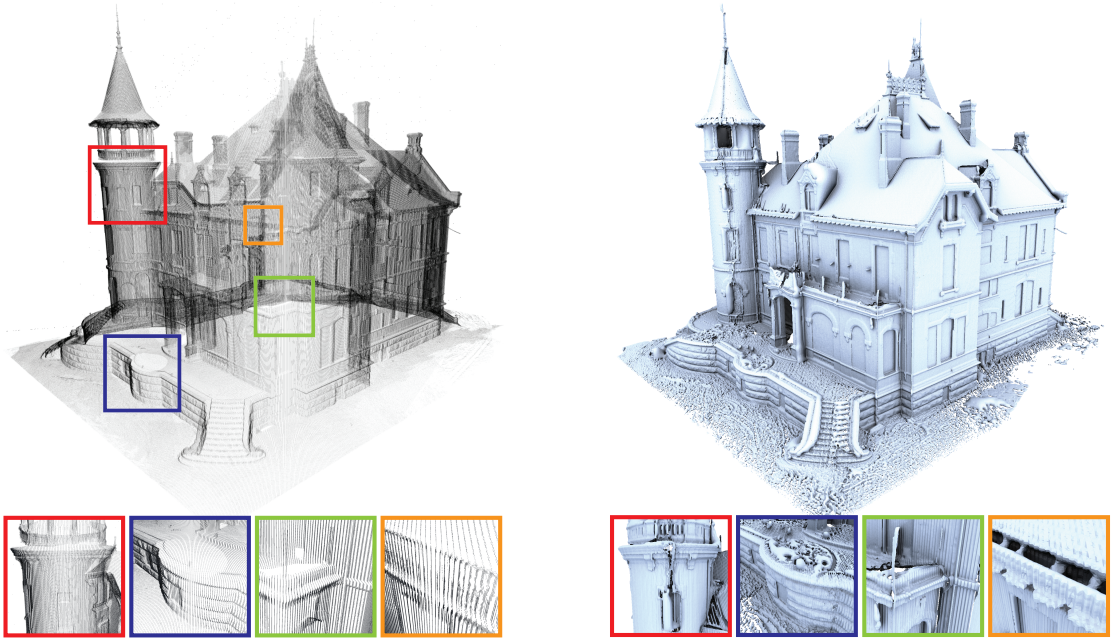


(a) SPSS (b) KPS (c) Screened Poisson

Figure 6.15: Comparison of different reconstructions of a point set sampling a non-orientable bounded surface (left inset).

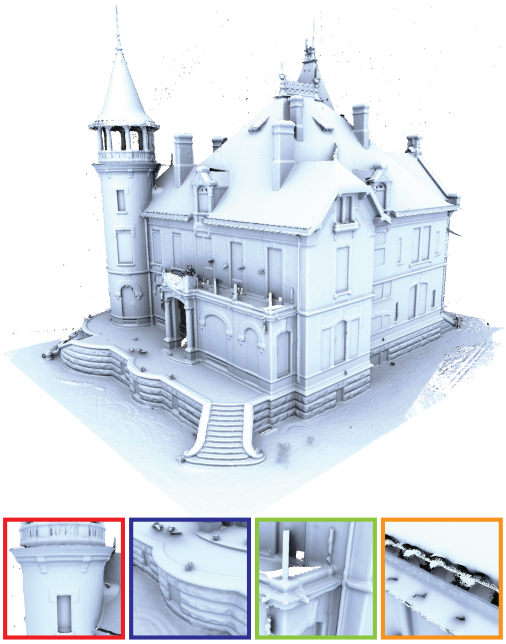
GG07] or methods constructing a consistent indicator function like Screened Poisson reconstruction tend to fail in the presence of normal discontinuities (Fig. 6.15c).

Deficient sampling. We evaluate the robustness of our method in the presence of noise and difficult sampling conditions on the example of a large LIDAR scan of a geometrically complex building, exhibiting thin features and sharp creases. Fig. 6.16a shows the input point cloud embedded in a number of outliers. The closeup views reveal a highly non-unform sampling pattern along vertical scan lines, which is reinforced by the registration of individual scans. The Screened Poisson reconstruction at octree depth 11 shown in Fig. 6.16b can crisply reconstruct small salient features, but is also oversensitive to the predominant sampling direction, which produces corresponding groove patterns. Furthermore, noise and sampling deficiencies prohibit a consistent orientation of normals, resulting in discontinuities and artifact surfaces along the tower (red), roof and chimney corners. Note that the resulting SPS mesh has been trimmed using a manually determined density threshold representing the best trade-off between artifact removal and model integrity. In contrast, an SPSS reconstruction produces a cleaner and more robust result (Fig. 6.16c). However, outliers and residual points result in artifact surfaces, as seen in the air and in front of the entrace (blue). Moreover, thin or undersampled features like the balcony rods (green) or fascia bearers (orange) degenerate to flat sheets. As with SPSS, the KPS reconstruction remains unaffected by the non-orientability of the input, producing a surface of generally similar quality (Fig. 6.16d). However, the closeup-views reveal differences in how our method responds to outliers and undersampled features. Due to their vanishing probability overlaps, Gaussians from outliers and residual points are mostly left out by the triangulation, which removes most artifacts visible in the SPSS reconstruction (air, entrance, terrace doors). Clear artifacts are visible at the eaves boards along the edge of the roof, where the sparse and deficient sampling produces an ambiguous Gaussian configuration that represents a break-down case for the probabilistic triangulation.



(a) input (374.9 MB)

(b) SPS-11 (88.6 MB)



(c) SPSS (374.9 MB)

(d) KPS (10.8 MB)

Figure 6.16: Reconstructions of a large, non-uniformly sampled point cloud with noise (a), where a consistent orientation of normals fails and thus produces notable seams in a Screened Poisson surface (b). Normal-independent methods like SPSS produce more robust results (c), but are typically still sensitive to outliers. Our method runs on compressed data (d), where the triangulation skips most outliers, but can still be confused by sparse sampling.

(a) SPSS (13.8 MB) (b) SPS-12 (7.4 MB) (c) KPS (2.1 MB)

Figure 6.17: Part of the chair model. (a) SPSS produces artifacts at outliers and concavities. The detail shows a flattening artifact puncturing the surface where its curvature lies below the reconstruction bandwidth. (b) High-resolution SPS result. (c) moderately compressed KPS, preserving high curvatures similar to SPS, but at a lower memory footprint.

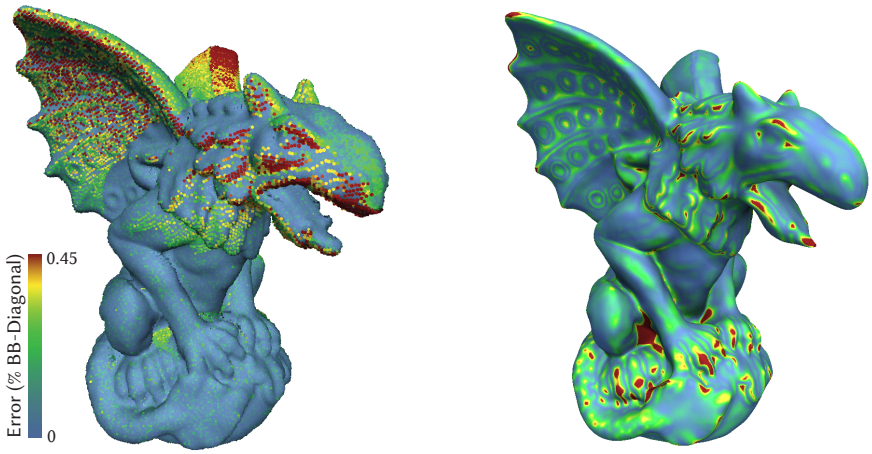
Fig. 6.16 also gives the memory footprints of the input point cloud and the different reconstructions in parentheses. Note that since the point set surface is defined by the entirety of input points, its memory footprint is considered to be equal.

Noise and high-frequency features. Another example of noise contamination is given by the chair detail shown in Fig. 6.17. SPSS produces numerous artifacts around outliers, but also at regions of too high curvature, where the local balancing plane either collapses sharp features to a thin sheet (see inset in Fig. 6.17a), or produces ghost surfaces in between concavities. In contrast, a high-accuracy SPS reconstruction preserves these sharp features, but exhibits more subtle surface noise (Fig. 6.17b). Finally, a moderately compressed KPS (∼20 points per Gaussian) ignores far outliers during triangulation and preserves a continuous surface at high curvatures, while smoothing subtle noise through Gaussian clustering (Fig. 6.17c).

6.6.3 Limitations.

Boundary smoothness. Fig. 6.15b shows the sparse control covariance mesh of the Moebius strip superimposed over the resulting kernel-product surface. At its boundaries, KPS produces slightly oscillating silhouettes due to varying valences of vertices, which influence the RBF bandwidths that define the power functions ψ_j in Eq. (6.19) (see Section 6.5.2). Altering these powers close to the boundaries or clipping the surface based on the underlying pdf $f_{\mathcal{M}}$ could be possible ways to obtain smoother silhouettes in future work.

Outlier sensitivity. The Gaussian locations obtained by HEM represent least-squares fits to their surrounding points, making our method sensitive to strong noise and outliers



(a) Input (0.115)

(b) KPS $\sigma_0 = 0.45$ (0.153)

(c) Avg + KPS
$\sigma_0 = 0.3$ (0.112)

(d) CLOP + KPS
$\sigma_0 = 0.3$ (0.096)

Figure 6.18: KPS on outlier-driven data (a), using (b) large kernels initialized in the points, (c) smaller kernels initialized in the locally weighted average of points, and (d) small kernels initialized in CLOP-resampled points. Red numbers in parentheses give the ground truth RMS error (in % of the model diameter).

in the input data. Figure 6.18a shows the noisy Gargoyle point cloud generated using the virtual scanning framework of Berger et. al [BLN+13], consisting of 16 scans that have been registered with ICP using a plausible amount of misalignment. The coloring indicates the normal distance to the ground-truth surface; overall RMS errors are given in red parentheses. While noise is generally modeled by the Gaussian covariances, too strong noise amplitudes and irregular outliers require larger kernels that can significantly smooth the resulting surface, as shown in Fig. 6.18b. Prefiltering the Gaussian means as proposed in Section 6.3 allows reducing the required kernel sizes, but is known to

produce a smoothing effect as well. In Fig. 6.18c, we performed such a weighted-average initialization of the Gaussian means using a weight kernel radius of 0.8%. The resulting KPS clearly reduces the overall error, but still exhibits a clear bias at features and concavities of high curvature. In order to improve the reconstruction in such cases, robust prefiltering techniques should be applied to the input in advance. Figure 6.18d shows the KPS after resampling the input using the robust CLOP operator, which has been introduced in Chapter 5. Computing a covariance mesh on this point set using the same HEM parameters produces a KPS that reduces the overall RMS error by additional 14%. This shows that although our method provides an efficient reconstruction framework for smooth surfaces, it does not offer the robustness of consolidation techniques like CLOP, and is better used in combination with these for very noisy data.

6.6.4 Further Applications

Mesh-based simplification. A given Gaussian triangulation allows us to further simplify a model through successive edge collapses without destroying its surface topology. This way we can achieve compression levels beyond what is possible through direct triangulation of similarly sparse mixtures. When collapsing two Gaussians Θ_i and Θ_j, we replace them by a new Gaussian Θ_{ij} that represents the maximum likelihood estimate of their common density:

$$\omega_{ij} = w_i + w_j \qquad \mu_{ij} = \omega_i \mu_i + \omega_j \mu_j$$
$$\Sigma_{ij} = \omega_i(\Sigma_i + \mu_i \mu_i^T) + \omega_j(\Sigma_j + \mu_j \mu_j^T) - \mu_{ij} \mu_{ij}^T$$

where $\omega_i = w_i/(w_i + w_j)$, $\omega_j = 1 - \omega_i$. To this end, the edges to be collapsed can be prioritized by the probabilistic similarity between their Gaussians. Fig. 6.19 shows such a simplification process, where we have successively collapsed the edge with the lowest mutual Kullback-Leibler divergence,

$$\min(D_{KL}(\Theta_i, \Theta_j), D_{KL}(\Theta_j, \Theta_i)).$$

To control the degree of geometry degeneration, the maximum allowed divergence for collapsing is constrained by the same regularization threshold α as is used by the HEM process in the initial mixture computation (see Section 6.3). To avoid flattening artifacts, we again do not increase α when aiming for stronger levels of simplification, but instead increase the scale of the covariances in the mesh. This reduces the divergence between neighboring Gaussians and thus allows for further collapses. Note that as discussed in Section 6.5.1, a globally uniform scaling of the covariances itself does not affect the kernel-product interpolation (Eq. (6.10)) and, consequently, the resulting surface. Since α is not increased, and the scaling does not increase the anisotropy of individual Gaussian kernels, the surface is prevented from over-flattening throughout further simplification. The covariances in Fig. 6.19 have first been scaled by a factor of 1.4^2, and then successively collapsed until a threshold of $\alpha = 2.0$ was reached. After four repetitions of this process we obtain a topology-preserving compression that would have been too sparse and blurry for a faithful direct triangulation.

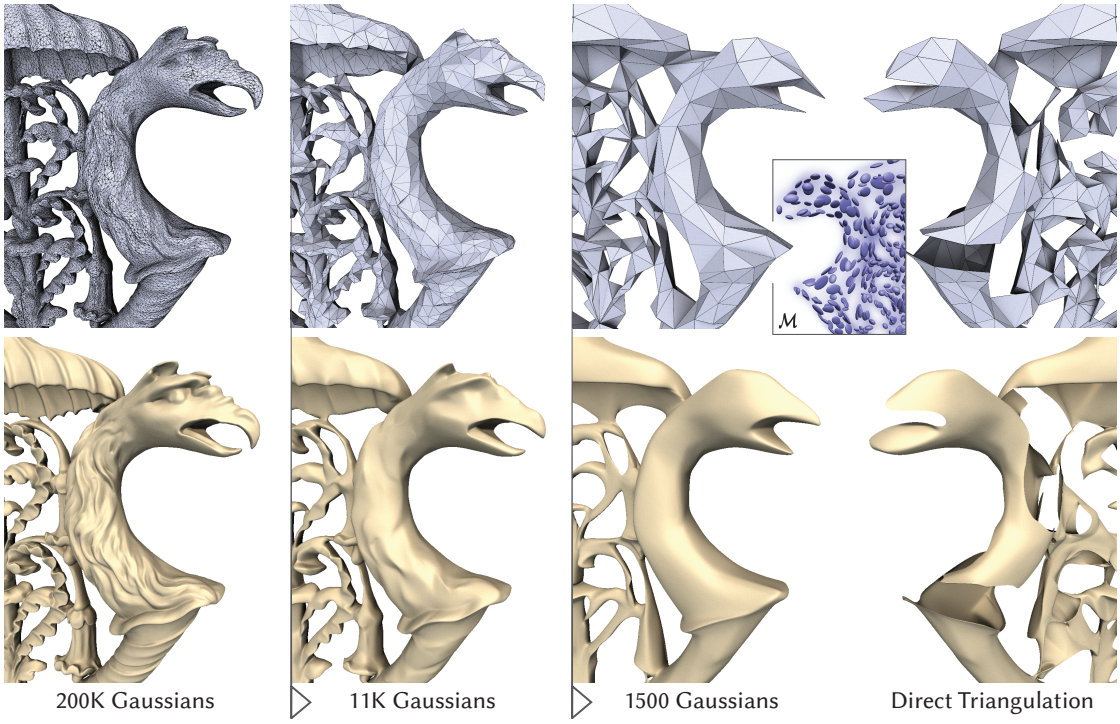
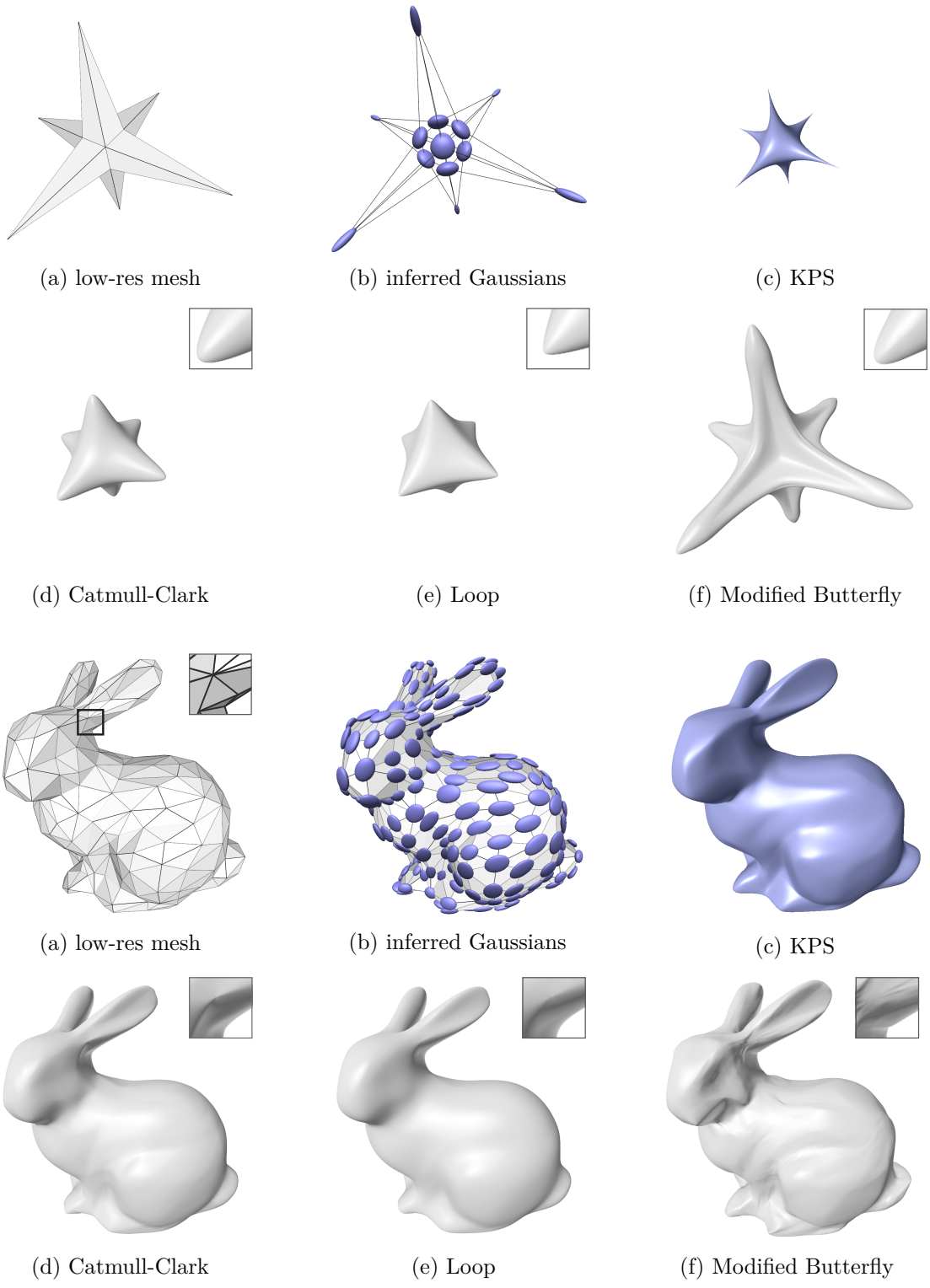


Figure 6.19: Topology-preserving simplification of the chair's covariance mesh (upper row) and its resulting KPS (lower row) after two and four iterations. The inset to the right shows the size and distribution of the final simplified Gaussians, where a direct triangulation would fail due to their sparseness and ambiguous probability overlaps.

Smooth surface modeling. Our approach offers a competitive alternative to subdivison surfaces for modeling smooth surfaces based on given low-resolution input meshes (Figure 6.20). As opposed to the previously proposed reconstruction pipeline, where a covariance mesh is computed by triangulating a Gaussian mixture, we can thereby start with a coarse input triangle mesh (Figure 6.20a) and infer Gaussian tensors at its vertices v_i, using the covariance

$$\Sigma_i = |J|^{-1} \sum_j v_j v_j^T - |J|^{-2} (\sum_j v_j)(\sum_j v_j)^T$$

of the local umbrella vertices $v_{j \in J}$, where $J = \{i \cup \mathcal{N}(i)\}$. This results in a covariance mesh (Figure 6.20b) that can be used for rendering a smooth surface (Figure 6.20c). The comparison to various subdivision surfaces demonstrates that we can produce a smooth, feature-rich surface where other methods produce unwanted singularities (lower row of Figure 6.20d), strong smoothing (Figure 6.20e), or high-frequency artifacts (Figure 6.20f). Nevertheless, the inherent anisotropy of the mesh vertices still allows for sharp features where their presence is natural to be assumed (pointy star in the upper row of the figure).



(a) low-res mesh (b) inferred Gaussians (c) KPS

(d) Catmull-Clark (e) Loop (f) Modified Butterfly

(a) low-res mesh (b) inferred Gaussians (c) KPS

(d) Catmull-Clark (e) Loop (f) Modified Butterfly

Figure 6.20: Comparison of Gaussian kernel-product surfaces on given low-resolution meshes with inferred Gaussians to different subdivision-surface techniques.

6.7 Summary

In this chapter, we have developed a surface definition based on a Gaussian mixture representation of input points. Gaussians are naturally capable of modeling uncertainty in sampled data, and can encode a set of points using only a few parameters, thus allowing a vastly compressed data representation. While previous work used this model solely to accelerate operators on point clouds, our method provides the first surface definition directly based on *sparse* Gaussian mixtures, which obviates the need to go back to a point-based representation and apply costly surface-reconstruction methods on the full dense point data. Our key technical contribution is the use of the expectation of a product of neighboring Gaussians for the geometry definition, varying the powers of their pdfs to smoothly interpolate between their covariance kernels.

We have shown that the quality of our surfaces can compete with those of elaborate state-of-the-art techniques like the widely used Screened Poisson surface reconstruction, while being independent of orientable normal information, and reducing both the required reconstruction time and memory storage costs due to the sparse representation. We demonstrate surface reconstructions of point-sampled models with moderate noise while relying on previous Gaussian-mixture based methods to clean models with more significant noise and outliers, thus providing a natural embedding into a fully Gaussian point-processing pipeline.

CHAPTER 7

Conclusion

7.1 Résumé

In this thesis, we have presented a line of research in which we developed a high-quality point-rendering technique for unstructured dynamic point clouds and discovered new ways of fast point-cloud processing and surface reconstruction that employ probabilistic models and improve on the quality and performance of state-of-the-art methods.

In Chapter 3, we have laid out the foundations for an in-situ surface reconstruction applied to raw unstructured points projected to the screen. By rasterizing conservative quad splats covering the projected neighborhood of each point, we were able to establish the communication between neighboring points and perform a variety of local operations directly in screen-space. This allowed for performing efficient nearest-neighbor queries for each point, which is an essential building block of any local surface reconstruction strategy. We have used this nearest-neighbor information for a fixed-sized neighborhood to perform a local screen-space triangulation of large and/or dynamic point sets.

Chapter 4 picks up on the idea of directly reconstructing surfaces from points at render-time and improves on both the flexibility and the view independence of the nearest-neighbor search, which ensures a much more stable reconstruction. We have shown that we can use arbitrarily sized neighborhoods to locally fit elliptical splats to each point, even without explicitly storing the individual neighbor points themselves. These splats allow for a much smoother surface rendering by employing high-quality surface-splatting methods, performing elliptical weighted-average blending on the GPU.

To address the common issue of noise and outliers in the input point data, Chapter 5 aims at integrating the robust L_1-based Locally Optimal Projection operator into our splat-reconstruction framework, enabling instant feature-preserving point-set resampling

in screen space directly before splat computation. To overcome the high computational effort of this iterative operator and yet ensure an execution at real-time frame rates, we introduced a new probabilistic approach that changes the representation of the input point cloud to a strongly compressed Gaussian mixture and reformulates the LOP operator to perform on this continuous probabilistic model. Our experiments show that this method is not only several times faster than the original discrete operator, but also exhibits a higher reconstruction accuracy, a more balanced sample distribution, and, unlike previous LOP variants, even allows a robust point-cloud upsampling in noisy data.

Finally, due to the various observed beneficial properties of Gaussian mixtures for point-cloud processing, Chapter 6 has been devoted to the development of an entirely probabilistic surface-reconstruction pipeline, which employs this sparse mixture model for a new, memory-efficient surface representation. We have introduced a technique that allows us to define smooth surfaces solely over sparse Gaussian mixtures that represent the distribution of given input point clouds. We employ a straight-forward approach for probabilistically triangulating the individual components of such a mixture, and introduced a new parametric interpolation technique that interpolates the Gaussians under consideration of their covariance kernels. We have shown that this method can outperform state-of-the-art surface reconstruction methods in both speed and accuracy, and enables to represent smooth surfaces with salient features in a highly memory-efficient way.

7.2 Epilogue

The research path documented in this thesis has been one of many branchings, backtracks, and unexpected surprises. My Master thesis on reconstructing reflections for point-cloud scenes in screen space has originally raised the need for obtaining as-smooth-as-possible depth maps when rendering unstructured point clouds. Starting my PhD with this classic rendering-related research problem, I would never have guessed I would finalize it solving the problem of defining surfaces over a compressed spatial probability model.

When developing the CLOP technique described in Chapter 5, the major aim was to accelerate the operator by simplification, while keeping the quality loss we expected due to the data simplification to a minimum. During the evaluations, we were actually surprised twice: Operating on the compressed data, i.e., the Gaussian mixture, resulted in lower ground-truth error measurements than operating on the full input point data (Section 5.10.2). Similarly, reducing the frequency of repulsion updates for acceleration showed an improved convergence behavior of sampling uniformity (Section 5.7.2). Both these results were entirely contrary to our expectations, and had us check our evaluation code and the resulting numbers twice before accepting these beneficial properties. What were originally despaired attempts of acceleration turned out to be an approach that immediately bent my subsequent path of PhD research. It is one of the exciting features

of research that lets you suppose you determine your direction, while your research is in fact directing you.

While I believe that probabilistic approaches bear lots of potential for improving further operations on point sets and other data in the future, I hope that my future research will keep me positively surprised and will open up further exciting new directions.

~

APPENDIX A

Virtual Scanning Parameters

In order to generate realistic models that can also be used in a ground-truth comparison, we applied a virtual scanning framework to some of the models used in this thesis to simulate an optical laser scanner [BLN+13]. In Table A.1 we describe the scanning parameters used for each model. The additive noise parameter simulates noise in the form of laser speckle, which stems from diffuse surface imperfections and can lead to outliers near depth continuities. According to the authors, typical noise magnitudes vary between 0 and 0.6, where the latter results in a highly corrupted signal. The peak magnitude threshold rejects points that have a low-radiance signal and hence a high likelihood to be an outlier. Setting it to a low value of 0.05 will keep most points.

Three of our models also exhibit registration errors. In particular, 4 of the 16 scans have an initial random rotational alignment error of 1.0°. They are subsequently registered using a locally weighted ICP algorithm [BR07]. Such an initial misalignment is a common source of outliers, when ICP converges to a bad local minimum [BLN+13].

Model	Camel	Daratech	Gargoyle	Garg. small
# scans	18	16	16	16
resolution	245^2	300^2	300^2	150^2
noise magnitue	0.2	0.1	0.1	0.1
laser smoothing	0.5	0.1	0.1	0.1
laser beam FOV	8	4.5	4.5	4.5
peak mag. threshold	0.05	0.1	0.1	0.1
registration error	0	1.0	1.0	1.0

Table A.1: Virtual scan parameters used for our models. For the non-listed parameters, the default values suggested by the framework were used.

APPENDIX B

HEM Algorithm Outline

Algorithm B.1 outlines the complete hierarchical EM procedure for computing a mixture $\mathcal{M}$ describing both the point distribution (by Gaussians Θ_s) and the spatially associated distribution of normals (by WN distributions Φ_s) as described in Chapter 5.

Base Mixture Initialization. For each point in parallel (Line 1), we first determine a conservative radius r (Line 2), being a multiple α_0 ($2 \sim 3$) of the point's nearest neighbor distance. Using this radius, we perform a kernel-accumulation pass (Line 3), simultaneously computing both the point's initial covariance $\Sigma_j^{(0)}$ and its local density weight v_j. The initial estimator for its surface normal $\mathbf{m}_j$ is then extracted using the smallest eigenvector of $\Sigma_j^{(0)}$ (Line 4). These quantities define the initial weight $w_j^{(0)}$, Gaussian $\Theta_j^{(0)}$ and WN $\Phi_j^{(0)}$ assigned to the j-th point (Lines 5-7), which make up the initial mixture $\mathcal{M}^{(0)}$ (Line 9).

Hierarchical Clustering. Each iteration of the following hierarchical clustering loop (Line 10) represents an EM step fitting a reduced set of *parent* components $\{\Theta|\Phi\}_s^{(l+1)}$ for the next level to the current set of *child* components $\{\Theta|\Phi\}_i^{(l)}$. Here, the i-th WN component Φ_i is always coupled to the i-th spatial component Θ_i when clustering. The parent set is initialized by randomly selecting elements from the current mixture $\mathcal{M}^{(l)}$ (Line 11), where we recommend a selection probability $\pi \approx 1/3$. For each such parent (Line 12) in parallel, Line 13 applies the regularization constraint α (Section 5.4.3) to select the index set I of child components it is *responsible* for and thus allowed to merge. Note that the centers $\mu_i^{(l)}$ of all child components can be found within a conservative ball with radius $\alpha \cdot \sigma_{max}$ around $\mu_s^{(l+1)}$, σ_{max} being the root of the largest eigenvalue of $\Sigma_s^{(l+1)}$. Line 14 computes the responsibilities r_{is} of the s-th component for its selected set of children, which are then used to update the maximum-likelihood estimate of the parent's parameters (Line 15). Finally, the function *Orphans* in Line 18 selects child components

that are not within the responsibility set I of any parent. They have to be taken along to the next level's mixture $\mathcal{M}^{(l+1)}$ together with the updated parent components in S.

Algorithm B.1: Outline of the HEM computation of mixture $\mathcal{M}$ including both spatial and spherical components.

input : point set $P = \{p_j\}_{j \in J}$, levels l_{max}, regularization α
output : mixture $\mathcal{M}^{(l_{max})} = \{w_s, \Theta_s, \Phi_s\}$

// Mixture Initialization
foreach $j \in J$ **do in parallel**
  $r \leftarrow \alpha_0 \cdot$ *NearestNeighborDist*(p_j); //Section 5.4.3
  $(\Sigma_j^{(0)}, v_j) \leftarrow$ *KernelCovDensity*$(p_j,\ r)$; //Eq. (5.13) / Sec. 5.6
  $\mathbf{m}_j \leftarrow$ *MinEigenVec*$(\Sigma_j^{(0)})$;
  $w_j^{(0)} \leftarrow (v_j |P|)^{-1}$; //Eq. (5.27)
  $\Theta_j^{(0)} \leftarrow (\mathbf{p}_j, \Sigma_j^{(0)})$;
  $\Phi_j^{(0)} \leftarrow (\mathbf{m}_j, \rho^{(0)})$; //Section 5.8.2
end
$\mathcal{M}^{(0)} \leftarrow \{w_j^{(0)}, \Theta_j^{(0)}, \Phi_j^{(0)}\}$;

// Hierarchical EM clustering
for $l \leftarrow 0$ **to** $l_{max} - 1$ **do**
  $S \leftarrow$ *RandomIndexSubset*$(\mathcal{M}^{(l)},\ \pi)$;
  foreach $s \in S$ **do in parallel**
    $I \leftarrow \{i \mid D_{KL}(\Theta_i^{(l)} || \Theta_s^{(l+1)}) < \alpha^2/2\}$; //Eq. (5.12)
    $\{r_{is}\}_{i \in I} \leftarrow$ *Responsibility*$(\Theta_s^{(l+1)},\ \Theta_i^{(l)})$; //Eq. (5.9)+(5.10)
    $(w_s, \Theta_s, \Phi_s)^{(l+1)} \leftarrow$ *UpdateMLE*$(\mathcal{M}^{(l)}, \{r_{is}\}_{i \in I})$; //Eq. (5.11),(5.30),(5.31)
  end
  $\mathcal{M}^{(l+1)} \leftarrow \{w_s^{(l+1)}, \Theta_s^{(l+1)}, \Phi_s^{(l+1)}\} \cup$ *Orphans*$(\mathcal{M}^{(l)})$;
end

APPENDIX C

Derivation of the Kernel-Product Expectation

We derive the closed matrix form for the expectation $E[\mathbf{x}|\Theta_J]$ of the product of multiple Gaussians in Eq. (6.10) and Eq. (6.20) using their general form

$$f(\mathbf{x}|\Theta_J) = w_J^{-1} \prod_{j \in J} f(\mathbf{x}|\Theta_j)^{\psi_j}.$$

First note that any power of a Gaussian is another weighted Gaussian

$$\begin{aligned} f(\mathbf{x}|\mu_j, \Sigma_j)^{\psi_j} &= |2\pi\Sigma_j|^{-\frac{1}{2}\psi_j} \, e^{-\frac{1}{2}(\mathbf{x}-\mu_j)^T \psi_j \Sigma_j^{-1}(\mathbf{x}-\mu_j)} \\ &= w_j \, f(\mathbf{x}|\mu_j, \psi_j^{-1}\Sigma_j) \end{aligned}$$

whose covariance has been scaled by its inverse power. Since the product Θ_J of any number of such scaled Gaussians is again Gaussian, the expectation $E[\mathbf{x}|\Theta_J]$ of this product equals its mean μ_J. To obtain μ_J, we bring the pdf of Θ_J into canonical form

$$\begin{aligned} f(\mathbf{x}|\Theta_J) &= |2\pi\Sigma_J|^{-\frac{1}{2}} \cdot e^{-\frac{1}{2}(\mathbf{x}-\mu_J)^T \Sigma_J^{-1}(\mathbf{x}-\mu_J)} \\ &= c \cdot e^{-\frac{1}{2}(\mathbf{x}^T \Sigma_J^{-1}\mathbf{x} \, - \, 2\mathbf{x}^T \Sigma_J^{-1}\mu_J)} \end{aligned} \tag{C.1}$$

where c collects all constant factors in the expression. After simplifying

$$\begin{aligned} &\prod_{j \in J} f(\mathbf{x}|\Theta_j)^{\psi_j} = \prod_{j \in J} w_j \, f(\mathbf{x}|\mu_j, \psi_j^{-1}\Sigma_j) \\ &= c \cdot e^{-\frac{1}{2}\sum_{j \in J}(\mathbf{x}-\mu_j)^T \psi_j \Sigma_j^{-1}(\mathbf{x}-\mu_j)} \\ &= c \cdot e^{-\frac{1}{2}\left(\mathbf{x}^T\left(\sum\limits_{j \in J} \psi_j \Sigma_j^{-1}\right)\mathbf{x} - 2\mathbf{x}^T\left(\sum\limits_{j \in J} \psi_j \Sigma_j^{-1}\mu_j\right) + \left(\sum\limits_{j \in J} \mu_j^T \psi_j \Sigma_j^{-1}\mu_j\right)\right)} \end{aligned}$$

and moving the constant third term in the exponent again over to c, we obtain the canonical form of Eq. (C.1) from which we can directly read out

$$\Sigma_J^{-1} = \sum_{j \in J} \psi_j \Sigma_j^{-1} \qquad \text{and} \qquad \Sigma_J^{-1} \mu_J = \sum_{j \in J} \psi_j \Sigma_j^{-1} \mu_j .$$

Inverting the first term to obtain Σ_J and right-multiplying the second term, we finally get the form in Eq. (6.20),

$$E[\mathbf{x}|\Theta_J] = \mu_J = \Sigma_J \, \Sigma_J^{-1} \mu_J = \Big(\sum_{j \in J} \psi_j \Sigma_j^{-1} \Big)^{-1} \Big(\sum_{j \in J} \psi_j \Sigma_j^{-1} \mu_j \Big).$$

List of Figures

1.1 Computational paths in point-based processing, reconstruction, and rendering. 3

3.1 Comparison between box splat rendering and screen-space triangulation. . 19
3.2 Processing pipeline of screen-space triangulation 20
3.3 Overview of the SST rendering and feedback loop 22
3.4 Storage layout for the nearest neighbors in SST 23
3.5 Two-pass screen-space neighbor search procedure 24
3.6 Screen-space triangulation scheme of a point and its neighbors 25
3.7 Deferred shading of a scene reconstructed with SST 26
3.8 Comparison of density-based box splats, uniform box splats, and SST . . 28
3.9 Comparison of heuristic box splats and SST in noisy datasets 29
3.10 Comparison of heuristic box splats and SST on a complex-geometry scan 29
3.11 Comparison of normal maps produced with box splatting and with SST . 30
3.12 Comparison of precomputed normals and SST normals in noisy data 31
3.13 Time consumptions of the individual SST shader passes 31

4.1 A dynamic point-cloud scene with 1 Million points rendered with Auto Splats 35
4.2 Auto-Splats processing pipeline 36
4.3 Overview of the Auto-Splatting algorithm 38
4.4 Addressing of world-space neighbors through their screen-space projections 39
4.5 Utilization of an eligible depth footprint for Early-Z discards 40
4.6 K-radius search algorithm and textures used for data writes and reads. . . 42
4.7 Silhouette quality of Auto Splats compared to precomputed splats 46
4.8 Auto-Splat reconstruction quality in varying point density 47
4.9 A huge out-of-core dataset rendered with Auto Splats 48
4.10 Performance decompositions of the Auto-Splats computation pipeline . . 49
4.11 Convergence behaviour of the k-radius search in Auto Splats 49
4.12 A dynamically scanned object reconstructed by Auto Splats 51
4.13 Auto-Splatted image of a range scan with in-situ curvature visualization . 52

5.1 Comparison of a CLOP resampling and an L_2-based reconstruction 57
5.2 CLOP processing pipeline 58
5.3 Gaussian Mixture on a signal with an outlier and its LOP reconstruction 63
5.4 Gaussians at different levels of HEM compression 65

5.5 Density functions involved in continuous attraction 66
5.6 Function α used in LOP and its approximation in CLOP 67
5.7 CLOP particle distribution in weighted mixtures 69
5.8 Volatility of repulsive-moments and NN variances for interleaved repulsion . 71
5.9 WN distribution, product of two WN distributions, and CSW step on the $\mathbb{S}^2$ 73
5.10 Model statistics and timings of the CLOP algorithm 76
5.11 Comparison of WLOP and the CLOP on the Face model 77
5.12 CLOP reconstruction error on two noisy registered scans 78
5.13 CLOP reconstruction error on two noisy registered scans 79
5.14 WLOP and CLOP on sharp edges and varying Gaussian noise 80
5.15 Mean error development with increasing mixture compression 81
5.16 Sampling regularity σ; point collapse of WLOP on a subsampled model . 82
5.17 Comparison of CLOP and WLOP for upsampling 83
5.18 Comparison of PCA normals and CSW normals 84
5.19 L_2 normals vs. CSW normals on a Kinect stream 85

6.1 KPS reconstruction overview 89
6.2 Processing pipeline of the KPS reconstruction. 90
6.3 Ridge contour and kernel-product contour on a 2d Gaussian mixture . . . 92
6.4 Overview of a probabilistic reconstruction pipeline 93
6.5 Defects of ridge surfaces in high-curvature regions 94
6.6 Different levels of hierarchical EM on a noisy point cloud 95
6.7 Probabilistic triangulation in thin-sheet situations 96
6.8 Kernel-product interpolation of two Gaussians 99
6.9 Influence of the shape of Gaussian kernels on their kernel-product contour 100
6.10 A kernel-product surface patch defined by 3 Gaussians 101
6.11 PPM construction and definition of an associated KPS 102
6.12 Weight function ξ and distance-based Gaussian power function ψ 104
6.13 Effects of increasing α and σ_0 on the resulting KPS 106
6.14 KPS reconstruction performance compared to Screened Poisson 107
6.15 SPSS, KPS and Screened Poisson on a non-orientable bounded surface . . 109
6.16 Reconstructions of a large, non-uniformly sampled noisy point cloud . . . 110
6.17 SPSS, KPS and SPS on a noisy model with high-frequency features 111
6.18 KPS on outlier-driven data after different kinds of prefiltering 112
6.19 Topology-preserving simplification of a covariance mesh 114
6.20 KPS on meshes with inferred Gaussians vs. subdivision surfaces 115

List of Tables

4.1 k-radius search times of Auto Splats compared to a GPU kd-tree. 50

6.1 KPS reconstruction statistics for different models 108

A.1 Virtual scanning parameters 121

Bibliography

[AA04] Anders Adamson and Marc Alexa. Approximating bounded, non-orientable surfaces from points. In *Proceedings of the Shape Modeling International 2004*, SMI '04, pages 243–252, Washington, DC, USA, 2004. IEEE Computer Society.

[AA06] Anders Adamson and Marc Alexa. Anisotropic point set surfaces. In *Proceedings of the 4th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*, AFRIGRAPH '06, pages 7–13, New York, NY, USA, 2006. ACM.

[AA09] Marc Alexa and Anders Adamson. Interpolatory point set surfaces - convexity and hermite data. *ACM Trans. Graph.*, 28(2):20:1–20:10, May 2009.

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

[ABCO+03] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15, Jan. 2003.

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

[AK04] Nina Amenta and Yong Joo Kil. Defining point-set surfaces. *ACM Trans. Graph.*, 23(3):264–270, August 2004.

[ASGCO10] Haim Avron, Andrei Sharf, Chen Greif, and Daniel Cohen-Or. l_1-sparse reconstruction of sharp point set surfaces. *ACM Trans. Graph.*, 29(5):135:1–135:12, November 2010.

[BDGS05] Arindam Banerjee, Inderjit S. Dhillon, Joydeep Ghosh, and Suvrit Sra. Clustering on the unit hypersphere using von Mises-Fisher distributions. *J. Mach. Learn. Res.*, 6:1345–1382, December 2005.

[BHZK05] M. Botsch, A. Hornung, M. Zwicker, and L. Kobbelt. High-quality surface splatting on today's GPUs. In *Point-Based Graphics, 2005. Eurographics/IEEE VGTC Symposium Proceedings*, pages 17 – 141, june 2005.

[BK03] Mario Botsch and Leif Kobbelt. High-quality point-based rendering on modern GPUs. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, PG '03, pages 335–, Washington, DC, USA, 2003. IEEE Computer Society.

[BLdG+16] Max Budninskiy, Beibei Liu, Fernando de Goes, Yiying Tong, Pierre Alliez, and Mathieu Desbrun. Optimal voronoi tessellations with Hessian-based anisotropy. *ACM Trans. Graph.*, 35(6):242:1–242:12, November 2016.

[BLN+13] Matthew Berger, Joshua A. Levine, Luis Gustavo Nonato, Gabriel Taubin, and Claudio T. Silva. A benchmark for surface reconstruction. *ACM Trans. Graph.*, 32(2):20:1–20:17, April 2013.

[BM12] Alexandre Boulch and Renaud Marlet. Fast and robust normal estimation for point clouds with sharp features. *Computer Graphics Forum*, 31(5):1765–1774, 2012.

[BR07] Benedict J. Brown and Szymon Rusinkiewicz. Global non-rigid alignment of 3-d scans. *ACM Trans. Graph.*, 26(3), July 2007.

[BSK04] Mario Botsch, Michael Spernat, and Leif Kobbelt. Phong splatting. In *Proceedings of the First Eurographics Conference on Point-Based Graphics*, SPBG'04, pages 25–32, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[BTS+14] Matthew Berger, Andrea Tagliasacchi, Lee M. Seversky, Pierre Alliez, Joshua A. Levine, Andrei Sharf, and Claudio T. Silva. State of the Art in Surface Reconstruction from Point Clouds. In Sylvain Lefebvre and Michela Spagnuolo, editors, *Eurographics 2014 - State of the Art Reports*. The Eurographics Association, 2014.

[BWK02] Mario Botsch, Andreas Wiratanaya, and Leif Kobbelt. Efficient high quality rendering of point sampled geometry. In *Proceedings of the 13th Eurographics workshop on Rendering*, EGRW '02, pages 53–64, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[CB14] Stephane Calderon and Tamy Boubekeur. Point morphology. *ACM Transactions on Graphics (Proc. SIGGRAPH 2014)*, 2014.

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

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

[CNPGVA02] J. Cotrina-Navau, N. Pla-Garcia, and M. Vigo-Anglada. A generic approach to free form surface generation. In *Proceedings of the Seventh ACM Symposium on Solid Modeling and Applications*, SMA '02, pages 35–44, New York, NY, USA, 2002. ACM.

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

[DB07] Rosen Diankov and Ruzena Bajcsy. Real-time adaptive point splatting for noisy point clouds. In *GRAPP (GM/R)*, pages 228–234, 2007.

[DG06] Tamal K. Dey and Samrat Goswami. Provable surface reconstruction from noisy samples. *Computational Geometry Theory and Applications*, 35(1):124–141, August 2006.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society Series B*, 39(1):1–38, 1977.

[DMKF16] M. Danelljan, G. Meneghetti, F. S. Khan, and M. Felsberg. A probabilistic framework for color-based point set registration. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1818–1826, June 2016.

[DRL10] Petar Dobrev, Paul Rosenthal, and Lars Linsen. Interactive image-space point cloud rendering with transparency and shadows. In Vaclav Skala, editor, *Communication Papers Proceedings of WSCG, The 18th International Conference on Computer Graphics, Visualization and Computer Vision*, pages 101–108, Plzen, Czech Republic, 2 2010. UNION Agency – Science Press.

[DS06] Tamal K. Dey and Jian Sun. Normal and feature approximations from noisy point clouds. In *Proceedings of Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 4337 of *Lecture Notes in Computer Science*, pages 21–32, 2006.

[Ebe11] David Eberly. Eigensystems for 3x3 symmetric matrices (revisited). http://www.geometrictools.com/Documentation/EigenSymmetric3x3.pdf, 2011. [Online; accessed 13-July-2017].

[FCOS05] Shachar Fleishman, Daniel Cohen-Or, and Cláudio T. Silva. Robust moving least-squares fitting with sharp features. *ACM Trans. Graph.*, 24(3):544–552, July 2005.

[GAB12] T. Guillemot, A. Almansa, and T. Boubekeur. Non local point set surfaces. In *2nd International Conference on 3D Imaging, Modeling, Processing, Visualization Transmission*, pages 324–331, Oct 2012.

[GDB08] V. Garcia, E. Debreuve, and M. Barlaud. Fast k nearest neighbor search using GPU. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, pages 1 –6, june 2008.

[GG07] Gaël Guennebaud and Markus Gross. Algebraic point set surfaces. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[GH95] Cindy M. Grimm and John F. Hughes. Modeling surfaces of arbitrary topology using manifolds. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 359–368, New York, NY, USA, 1995. ACM.

[GHJ+08] Xianfeng Gu, Ying He, Miao Jin, Feng Luo, Hong Qin, and Shing-Tung Yau. Manifold splines with a single extraordinary point. *Computer-Aided Design*, 40(6):676 – 690, 2008. Selected Papers from the ACM Solid and Physical Modeling Symposium 2007.

[GHQ06] Xianfeng Gu, Ying He, and Hong Qin. Manifold splines. *Graphical Models*, 68(3):237 – 254, 2006.

[GKS00] M. Gopi, S. Krishnan, and C.T. Silva. Surface reconstruction based on lower dimensional localized delaunay triangulation. *Computer Graphics Forum*, 19(3):467–478, 2000.

[GM04] Enrico Gobbetti and Fabio Marton. Layered point clouds: a simple and efficient multiresolution structure for distributing and rendering gigantic point-sampled models. *Computers & Graphics*, 28(6):815–826, 2004.

[GNN10] Vincent Garcia, Frank Nielsen, and Richard Nock. Hierarchical Gaussian mixture model. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Dallas, Texas, USA, March 2010.

[GP07] Markus Gross and Hanspeter Pfister. *Point-Based Graphics (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[Gro09] Markus Gross. Point based graphics: state of the art and recent advances. In *ACM SIGGRAPH 2009 Courses*, SIGGRAPH '09, pages 18:1–18:68, New York, NY, USA, 2009. ACM.

[HAT11] R. Hartley, K. Aftab, and J. Trumpf. L_1 rotation averaging using the weiszfeld algorithm. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 3041–3048, 2011.

[HDD+92] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '92, pages 71–78. ACM, 1992.

[HK04] Takashi Kanai Hiroaki Kawata, Alexandre Gouaillard. Interactive point-based painterly rendering. In *Proc. International Conference on Cyberworlds 2004 (CW2004)*, pages 293–299, Nov. 2004.

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

[HPS08] Kai Hormann, Konrad Polthier, and Alia Sheffer. Mesh parameterization: Theory and practice. In *ACM SIGGRAPH ASIA 2008 Courses*, SIGGRAPH Asia '08, pages 12:1–12:87, New York, NY, USA, 2008. ACM.

[IKH+11] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of Symposium on User Interface Software and Technology*, UIST '11, pages 559–568. ACM, 2011.

[JGBZ10] Tang Jie, Wu Gangshan, Xu Bo, and Gong Zhongliang. Interactive point clouds fairing on many-core system. In *Proceedings of the International Symposium on Parallel and Distributed Processing with Applications*, ISPA '10, pages 557–562, Washington, DC, USA, 2010. IEEE Computer Society.

[JLSW02] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. *ACM Trans. Graph.*, 21(3):339–346, July 2002.

[JRJ11] Wenzel Jakob, Christian Regg, and Wojciech Jarosz. Progressive expectation–maximization for hierarchical volumetric photon mapping. *Computer Graphics Forum*, 30(4), June 2011.

[JV11] Bing Jian and Baba C. Vemuri. Robust point set registration using Gaussian mixture models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(8):1633–1645, August 2011.

[Kai67] T. Kailath. The divergence and Bhattacharyya distance measures in signal selection. *IEEE Transactions on Communication Technology*, 15(1):52–60, February 1967.

[KB04] Leif Kobbelt and Mario Botsch. A survey of point-based techniques in computer graphics. *Computers & Graphics*, 28:801–814, December 2004.

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

[KH13] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Trans. Graph.*, 32(3):29:1–29:13, July 2013.

[KSNS07] Evangelos Kalogerakis, Patricio Simari, Derek Nowrouzezahrai, and Karan Singh. Robust statistical estimation of curvature on discretized surfaces. In *Proceedings of the Eurographics Symposium on Geometry Processing*, SGP '07, pages 13–22, 2007.

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

[LCOLTE07] Yaron Lipman, Daniel Cohen-Or, David Levin, and Hillel Tal-Ezer. Parameterization-free projection for geometry reconstruction. *ACM Trans. Graph.*, 26(3), July 2007.

[Lev98] David Levin. The approximation power of moving least-squares. *Mathematics and Computation*, 67(224):1517–1531, October 1998.

[Lev03] David Levin. Mesh-independent surface interpolation. In Guido Brunnett, Bernd Hamann, Heinrich Müller, and Lars Linsen, editors, *Geometric Modeling for Scientific Visualization*, pages 37–49. Springer, Berlin, Heidelberg, 2003.

[LLP+10] Ruosi Li, Lu Liu, Ly Phan, Sasakthi Abeysinghe, Cindy Grimm, and Tao Ju. Polygonizing extremal surfaces with manifold guarantees. In *Proceedings of the 14th ACM Symposium on Solid and Physical Modeling*, SPM '10, pages 189–194, New York, NY, USA, 2010. ACM.

[LP03] Lars Linsen and Hartmut Prautzsch. Fan clouds - an alternative to meshes. In Tetsuo Asano, Reinhard Klette, and Christian Ronse, editors, *Geometry, Morphology, and Computational Imaging*, volume 2616 of *Lecture Notes in Computer Science*, pages 39–57. Springer, Berlin, Germany, 2003. linsenprautzsch.

[LS81] P. Lancaster and K. Salkauskas. Surfaces generated by moving least squares methods. *Mathematics of Computation*, 37:141 – 158, 1981.

[LSK+10] Bao Li, Ruwen Schnabel, Reinhard Klein, Zhiquan Cheng, Gang Dang, and Jin Shiyao. Robust normal estimation for point clouds with sharp features. *Computers & Graphics*, 34(2):94–106, April 2010.

[LTdF+09] P. Leite, J. Teixeira, T. de Farias, V. Teichrieb, and J. Kelner. Massively parallel nearest neighbor queries for dynamic point clouds on the GPU. In *Computer Architecture and High Performance Computing, 2009. SBAC-PAD '09. 21st International Symposium on*, pages 19 –25, oct. 2009.

[LXJF13] Bin Liao, Chunxia Xiao, Liqiang Jin, and Hongbo Fu. Efficient feature-preserving local projection operator for geometry reconstruction. *Computer-Aided Design*, 45(5):861–874, 2013.

[MDGD+10] Patrick Mullen, Fernando De Goes, Mathieu Desbrun, David Cohen-Steiner, and Pierre Alliez. Signing the unsigned: Robust surface reconstruction from raw pointsets. *Computer Graphics Forum*, 29(5):1733–1741, 2010.

[MJ09] K.V. Mardia and P.E. Jupp. *Directional Statistics*. Wiley Series in Probability and Statistics. Wiley, 2009.

[MKC07] Ricardo Marroquim, Martin Kraus, and Paulo Roma Cavalcanti. Efficient point-based rendering using image reconstruction. In *Proceedings Symposium on Point-Based Graphics*, pages 101–108. Eurographics Association, 2007.

[MNG04] N. J. Mitra, A. Nguyen, and L. Guibas. Estimating surface normals in noisy point cloud data. In *special issue of International Journal of Computational Geometry and Applications*, volume 14, pages 261–276, 2004.

[MOC08] R. Marroquim, A. Oliveira, and P.R. Cavalcanti. High quality image reconstruction of point models. In *Computer Graphics and Image Processing, 2008. SIBGRAPI '08. XXI Brazilian Symposium on*, pages 297 –304, oct. 2008.

[NG00] J. Cotrina Navau and N. Pla Garcia. Modeling surfaces from meshes of arbitrary topology. *Computer Aided Geometric Design*, 17(7):643 – 671, 2000.

[OBA+03] Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. Multi-level partition of unity implicits. *ACM Trans. Graph.*, 22(3):463–470, July 2003.

[OE11] Umut Ozertem and Deniz Erdogmus. Locally defined principal curves and surfaces. *Journal of Machine Learning Research*, 12:1249–1286, 2011.

[OGG09] Cengiz Oztireli, Gaël Guennebaud, and Markus Gross. Feature preserving point set surfaces based on non-linear kernel regression. *Computer Graphics Forum*, 28(2):493–501, 2009.

[Paj03] Renato Pajarola. Efficient level-of-details for point based rendering. In *Computer Graphics and Imaging*, pages 141–146, 2003.

[PLM10] Jia Pan, C. Lauterbach, and D. Manocha. Efficient nearest-neighbor computation for GPU-based motion planning. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2243 –2248, oct. 2010.

[PMG04] Mark Pauly, Niloy J. Mitra, and Leonidas J. Guibas. Uncertainty and variability in point cloud surface data. In *Proceedings of the First Eurographics Conference on Point-Based Graphics*, SPBG'04, pages 77–84, Aire-la-Ville, Switzerland, 2004. Eurographics Association.

[PP12] K. B. Petersen and M. S. Pedersen. The matrix cookbook, November 2012. Version 20121115.

[PZvBG00] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: surface elements as rendering primitives. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 335–342, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[QMN09] Deyuan Qiu, Stefan May, and Andreas Nüchter. GPU-accelerated nearest neighbor search for 3d registration. In *Proceedings of the 7th International Conference on Computer Vision Systems: Computer Vision Systems*, ICVS '09, pages 194–203, Berlin, Heidelberg, 2009. Springer-Verlag.

[RL08] Paul Rosenthal and Lars Linsen. Image-space point cloud rendering. In *Proceedings of Computer Graphics International*, pages 136–143, 2008.

[RL16] Mael Rouxel-Labbe. *Anisotropic mesh generation*. Theses, Université Côte d'Azur, December 2016.

[RLWB16] M Rouxel-Labbé, M Wintraecken, and J.-D Boissonnat. Discretized Riemannian Delaunay triangulations. In *Proceedings 25th International Meshing Roundtable (IMR25)*, Washington DC, United States, September 2016. Elsevier.

[RPZ02] Liu Ren, Hanspeter Pfister, and Matthias Zwicker. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. In *Computer Graphics Forum (Eurographics 2002)*, volume 21, pages 461–470, September 2002.

[SG07] Jochen Suessmuth and Guenther Greiner. Ridge based curve and surface reconstruction. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, SGP '07, pages 243–251, Aire-la-Ville, Switzerland, 2007. Eurographics Association.

[SJW07] Daniel Scherzer, Stefan Jeschke, and Michael Wimmer. Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In Jan Kautz and Sumanta Pattanaik, editors, *Rendering Techniques 2007 (Proceedings Eurographics Symposium on Rendering)*, pages 45–50. Eurographics, Eurographics Association, June 2007.

[SXG+09] Marcelo Siqueira, Dianna Xu, Jean Gallier, Luis Gustavo Nonato, Dimas Martinez Morera, and Luiz Velho. A new construction of smooth surfaces from triangle meshes using parametric pseudo-manifolds. *Computers & Graphics*, 33(3):331 – 340, 2009. {IEEE} International Conference on Shape Modelling and Applications 2009.

[SZW09] Claus Scheiblauer, Norbert Zimmermann, and Michael Wimmer. Interactive domitilla catacomb exploration. In *10th VAST International Symposium on Virtual Reality, Archaeology and Cultural Heritage (VAST09)*, pages 65–72. Eurographics Association, September 2009.

[Vas98] Nuno Vasconcelos. Learning mixture hierarchies. In *Proceedings of Advances in Neural Information Processing Systems*, NIPS'98, pages 606–612. MIT Press, 1998.

[VJ09] G. Vecchia and B. Jüttler. Piecewise rational manifold surfaces with sharp features. In *Proceedings of the 13th IMA International Conference on Mathematics of Surfaces XIII*, pages 90–105, Berlin, Heidelberg, 2009. Springer-Verlag.

[VJK08] Giovanni Della Vecchia, Bert Jüttler, and Myung-Soo Kim. A construction of rational manifold surfaces of arbitrary topology and smoothness from triangular meshes. *Computer Aided Geometric Design*, 25(9):801 – 815, 2008. Classical Techniques for Applied Geometry.

[vKvdBT07] Kees van Kooten, Gino van den Bergen, and Alex Telea. Point-based visualization of metaballs on a GPU. *GPU Gems*, (3), 2007.

[WBKP08] B. Walter, K. Bala, M. Kulkarni, and K. Pingali. Fast agglomerative clustering for rendering. In *IEEE Symposium on Interactive Ray Tracing*, pages 81–86, 2008.

[WS06] Michael Wimmer and Claus Scheiblauer. Instant points. In *Proceedings Symposium on Point-Based Graphics 2006*, pages 129–136. Eurographics, Eurographics Association, July 2006.

[WZK05] Jianhua Wu, Zhuo Zhang, and Leif Kobbelt. Progressive Splatting. In Marc Alexa, Szymon Rusinkiewicz, Mark Pauly, and Matthias Zwicker, editors, *Eurographics Symposium on Point-Based Graphics (2005)*. The Eurographics Association, 2005.

[YHGT10] Jason C. Yang, Justin Hensley, Holger Grün, and Nicolas Thibieroz. Real-time concurrent linked list construction on the gpu. In *Proceedings of the 21st Eurographics Conference on Rendering*, EGSR'10, pages 1297–1304, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[YZ04] Lexing Ying and Denis Zorin. A simple manifold-based construction of surfaces of arbitrary smoothness. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 271–275, New York, NY, USA, 2004. ACM.

[ZGHG11] Kun Zhou, Minmin Gong, Xin Huang, and Baining Guo. Data-parallel octrees for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 17:669–681, 2011.

[ZHWG08] Kun Zhou, Qiming Hou, Rui Wang, and Baining Guo. Real-time kd-tree construction on graphics hardware. In *ACM SIGGRAPH Asia 2008 papers*, SIGGRAPH Asia '08, pages 126:1–126:11, New York, NY, USA, 2008. ACM.

[ZSW+10] Qian Zheng, Andrei Sharf, Guowei Wan, Yangyan Li, Niloy J. Mitra, Daniel Cohen-Or, and Baoquan Chen. Non-local scan consolidation for 3d urban scenes. *ACM Trans. Graph.*, 29:94:1–94:9, July 2010.

Reinhold **Preiner**

Doctoral Researcher

contact

Favoritenstraße 9-11
1040, Wien
Austria

rp@cg.tuwien.ac.at

languages

native German
fluent English
basic Spanish

interests

visual computing
geometry
reconstruction
probabilistic methods

education

2010–2017 **Doctor** of Technical Sciences — TU Wien, Austria
Dynamic and Probabilistic Point-Cloud Processing
(exp. date of defense: Oct 2017)

2008–2010 **Diplom-Ingenieur (Dipl.-Ing.)** in Visual Computing — TU Wien, Austria
Interactive Curved Reflections in Large Point Clouds
Graduation with honors

2004–2008 **Bachelor (Bakk.rer.soc.oec.)** — TU Graz, Austria
in Software Engineering and Knowledge Management
Adaption der Grafikausgabe eines First-Person-Shooters für den Betrieb in einer virtuellen Umgebung

professional

2010–2017 **TU Wien** — Vienna, Austria
University Assistant and Research Assistant
at the Rendering and Modeling group of the Institute of Computer Graphics and Algorithms

2007–2008 **Siemens Austria** — Vienna, Austria
Software-quality controlling and software-engineering internships

teaching

2014, 2016 **Algorithms for Real-Time Rendering** — TU Wien
Responsible for specific lecture units on real-time rendering

2014–2015 **Modelling in Computer Graphics** — TU Wien
Responsible for lecture units on specific modeling topics

2010–2013 **Introduction to Visual Computing** — TU Wien
Assistance in lecture units and design, execution and correction of final exams

2010 **Computer Graphics 2** — TU Wien
Tutor for student support in C++/OpenGL game-engine programming

2009 **Computer Graphics 1** — TU Wien
Tutor for student support in introductory CG assignments

2008–2010 **Introduction to Programming** — TU Wien
Tutor of student groups for object-oriented programming exercises in Java

reviewing

CESCG 2011/2012/2013, **CGI** 2011, **GCH** 2011/2014, **PG** 2012, **CAG** 2013, **JOCCH** 2014, **TVCG** 2014/2016, **CAGD** 2016, **SIGGRAPH ASIA** 2016, **TVCJ** 2016, **CGF** 2017, **HPG** 2017, **SIGGRAPH** 2017

supervision

Student Projects

2010	*GPU Point-Cloud Raytracing*	Probst, Kolesik
2013	*Real-Time Global Illumination using Virtual Area Lights*	Weinzierl
2016	*Kinect Fusion using Gaussian Mixtures*	Jahrmann

Bachelor Theses

2013	*Rasterized Curved Reflections in Screen Space*	Szabo
2016	*Molecule-Rendering in Unity3D*	Prost

Diploma Theses

2017	*Adaptively-Clustered Virtual Area Lights for Real-Time Global Illumination*	Weinzierl

publications

full papers

Murat Arikan, **Reinhold Preiner**, Michael Wimmer. *Multi-Depth-Map Raytracing for Efficient Large-Scene Reconstruction*. IEEE Transactions on Visualization & Computer Graphics, 22(2):1127–1137, February **2016**

Johanna Schmidt, **Reinhold Preiner**, Thomas Auzinger, Michael Wimmer, Meister Eduard Gröller, Stefan Bruckner. *YMCA – Your Mesh Comparison Application*. In IEEE Visual Analytics Science and Technology. November **2014**

Murat Arikan, **Reinhold Preiner**, Claus Scheiblauer, Stefan Jeschke, Michael Wimmer. *Large-Scale Point-Cloud Visualization through Localized Textured Surface Reconstruction*. IEEE Transactions on Visualization & Computer Graphics, 20(9):1280-1292, September **2014**

Reinhold Preiner, Oliver Mattausch, Murat Arikan, Renato Pajarola, Michael Wimmer. *Continuous Projection for Fast L1 Reconstruction*. ACM Transactions on Graphics (In Proceedings of ACM SIGGRAPH 2014), 33(4):47:1–47:13, August **2014**

Thomas Auzinger, Przemyslaw Musialski, **Reinhold Preiner**, Michael Wimmer. *Non-Sampled Anti-Aliasing*. In Proceedings of the 18th International Workshop on Vision, Modeling and Visualization (VMV 2013), pages 169–176. September **2013**

Reinhold Preiner, Stefan Jeschke, Michael Wimmer. *Auto Splats: Dynamic Point Cloud Visualization on the GPU*. In Proceedings of Eurographics Symposium on Parallel Graphics and Visualization, pages 139–148. May **2012**

non-peer reviewed papers

Johanna Schmidt, Bernhard Fröhler, **Reinhold Preiner**, Johannes Kehrer, Meister Eduard Gröller, Stefan Bruckner, Christoph Heinzl. *Visual Analysis of Volume Ensembles Based on Local Features*. Technical Report, TR-186-2-16-2, May **2016**

Reinhold Preiner, Michael Wimmer. *Interactive Screen-Space Triangulation for High-Quality Rendering of Point Clouds*. Technical Report, TR-186-2-12-01, April **2012**

Reinhold Preiner, Michael Wimmer. *Real-Time Global Illumination in Point Clouds*. In Proceedings of Central European Seminar on Computer Graphics. May **2010**