FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Lokale Rekonstruktion

mit Isotropisch Ausgeglichenen Nachbarschaften

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Daniel Prieler BSc
Matrikelnummer 0726319

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Mag.rer.soc.oec. Stefan Ohrhallinger PhD

Wien, 18. Jänner 2016

Daniel Prieler	Michael Wimmer

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Local Reconstruction

Using Isotropically Fair Neighborhoods

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Media Informatics and Visual Computing

by

Daniel Prieler BSc
Registration Number 0726319

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Mag.rer.soc.oec. Stefan Ohrhallinger PhD

Vienna, 18th January, 2016

Daniel Prieler	Michael Wimmer

Erklärung zur Verfassung der Arbeit

Daniel Prieler BSc
Zieglergasse 94/21, 1070 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, 18. Jänner 2016

Daniel Prieler

Kurzfassung

Durch die stetig steigende Verfügbarkeit von 3D Scannern in der Industrie und der Unterhaltungsbranche, sind schnelle und zuverlässige Rekonstruktionsmethoden besonders wichtig. Punktwolken, vor allem sogenannte “Range Images” aus 3D Scannern sind oft nicht uniform abgetastet und haben auch variables Rauschverhalten. Aktuelle Methoden des Stands der Technik verlassen sich meist darauf, dass die GröSSe des Rauschens über Parameter durch den Benutzer bestimmt wird. Dadurch wird ein, für das gesamte Modell gültiger Rausch-Wert bestimmt, was bei variablen (nicht uniformen) Rausch-Werten zu gröSSeren Fehlern in der Rekonstruktion führt. In dieser Arbeit stellen wir eine isotropisch ausgeglichene Nachbarschaftsdefinition vor, die speziell für nicht uniform abgetastete Punktwolken entworfen ist. Unser iterativer resampling-Ansatz schätzt den Rauschanteil an jedem Punkt und passt sich diesem an. Dadurch wird die Qualität der Rekonstruktion erhöht, ohne, dass der Benutzer Parameterwerte setzen muss. Die Datenstrukturen, die während des resamplings erzeugt werden, sorgen für eine schnellere, global konsistente Orientierung der Normalen der Punktwolke. Evaluierung unserer Methode zeigt, dass sie bessere Ergebnisse bei verschieden groSSen Rauschanteilen und nicht uniformer Abtastung liefert als Verfahren des aktuellen Stands der Technik. Sowohl der resampling Vorgang als auch die konsistente Normalen Orientierung arbeiten nur mit lokalen Daten und können daher effizient parallel implementiert werden. Unsere GPU-Implementierung für die Kugel-Regression ist um den Faktor 20 schneller als der gewöhnliche sequentielle Ansatz.

Abstract

The increasing availability of 3D scanning devices in both industrial and entertainment environments (e.g., Microsoft Kinect) creates a demand for fast and reliable resampling and reconstruction techniques. Point clouds, especially raw range images, are often non-uniformly sampled and subject to non-uniform noise levels. Current state-of-the-art techniques often require user-provided parameters that estimate the noise level of the point cloud. This produces sub-optimal results for point sets with varying noise extent. We propose an isotropically fair neighborhood definition which is specifically designed to address non-uniformly sampled point clouds. Our iterative point cloud resampling method estimates and adapts to the local noise level at each sample. This increases the reconstruction quality for point clouds with high noise levels while being completely parameter free. The data structures built during the resampling process are reused to speed up the process of creating a consistent normal orientation. Evaluation of the resampling quality shows that our technique outperforms current state-of-the-art methods for varying noise levels and non-uniform sampling. Both the resampling algorithm and the subsequent consistent normal orientation operate locally and can be implemented efficiently in parallel. Our GPU sphere regression implementation outperforms the standard sequential procedure by a factor of 20.

Contents

Kurzfassung **vii**

Abstract **ix**

Contents **xi**

1 Introduction **1**
1.1 Motivation 2
1.2 Problem description: Surface Resampling and Reconstruction 2
1.3 Contributions 3
1.4 Structure of the Work 4

2 Related Work **5**
2.1 Resampling 5
2.2 Surface Reconstruction 12

3 Resampling on Global Manifold **19**
3.1 Idea and Definition 20
3.2 Construction in 2D space 21
3.3 Discussion 25

4 Local Resampling using Isotropically Fair Neighborhoods **29**
4.1 Isotropically Fair Neighborhoods 30
4.2 Local Surface Fitting 37
4.3 Spherical Regression 43

5 Globally Consistent Normal Orientation **47**
5.1 Problem definition 47
5.2 Improvements using local neighborhoods 51

6 Implementation Details **57**
6.1 Regression Sphere Calculation 57
6.2 Sphere Filtering 59
6.3 Consistent Normal Orientation 60

6.4 Local Triangulation 60

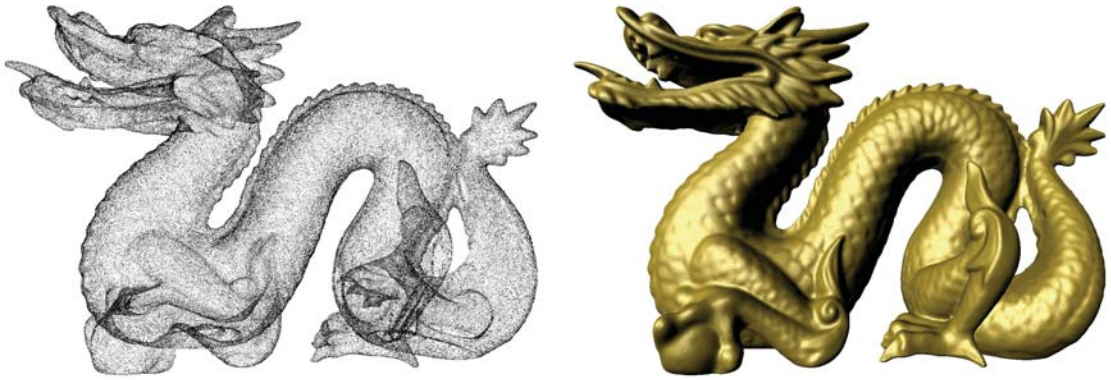
7 Results **63**
7.1 Noise-Reduction Quality 64
7.2 Consistent Normal Orientation 73
7.3 Performance 76

8 Conclusion **83**
8.1 Synopsis 83
8.2 Future Work 83

Bibliography **85**

CHAPTER 1

Introduction



3D models are used in different application domains. Industrial design, entertainment (games and movies), education and many others need high-quality digital representations of real-world objects. Scanners are used to generate 3D models of small-scale objects as well as bigger entities like facades and whole landscapes. Besides sophisticated expensive scanners for professional use, low-cost devices are available and used more and more often. The output of scanning devices is not an already completed model of the surface but a huge set of points – discretized positional values without any connectivity information. Sometimes the normal vector of a surface point can also be measured by the scanning device.

Models are commonly represented as a triangular mesh data structure: The surface is piecewise linearly approximated by triangles. There are many algorithms and methods in the literature to edit, process and render meshes. Transforming an unorganized point cloud (points in R^3 without normals or connectivity information) into a triangular mesh therefore is an important operation.

1.1 Motivation

As with all measuring procedures, scanning a surface introduces errors. The process of recovering the surface of the underlying object must account for this. State-of-the-art techniques tackle this problem by modeling the noise and smoothing the point cloud. Often, the user has to estimate the extent of the error by choosing the right parameters of the method. An automatic method that can estimate the noise level of any given point cloud would eliminate the need for user input.

Inaccurate point samples are not the only challenge. In general, the surface of an object is sampled non-uniformly by the scanner. Depending on the angle of view of the scanner, occlusion effects and different material properties, the resulting point cloud covers the surface of the object unevenly. This is barely accounted for in the current state-of-the-art. Extending reconstruction methods such that they can deal with the non-uniformity of the sampling would increase the quality of the reconstructed surface.

One important step when reconstructing the global boundary of an object is to find a globally consistent normal orientation. The normals of local patches determine the direction of the patch but not its orientation in the global context. Normals have to be flipped so that neighboring normals on the surface point in the same direction. As the problem is global by definition, current techniques need to build global data structures, which are expensive to compute for very large point clouds. Using the relation between local points and propagating the orientation in a small vicinity may speed up this process.

1.2 Problem description: Surface Resampling and Reconstruction

This thesis deals with the problem of surface reconstruction from unorganized point clouds. The input only includes the positions of the samples (created for example with a 3D scanner) $\mathbf{P} = \{p_i \in R^3\}$. There is no additional information about the points or the underlying surface available. In this thesis, we assume a Gaussian distributed noise (equation 1.1):

$$p_i = (x_i, y_i, z_i), \text{ where } \quad x_i = \tilde{x_i} + \epsilon \quad y_i = \tilde{y_i} + \delta \quad z_i = \tilde{z_i} + \gamma \tag{1.1}$$

The values $\tilde{x_i}$, $\tilde{y_i}$ and $\tilde{z_i}$ are the real coordinates, ϵ, δ and γ are isotropic Gaussian distributed random variables $\sim \mathcal{N}(\mu, \sigma^2)$, where σ is the standard deviation and the mean of the distribution μ is set to 0. Although the distribution of each single point is assumed to be isotropic, no assumptions are made about the noise level (σ^2) in general: each single point may have a different noise extent σ_i^2.

Sharp features like edges and corners cause discontinuities in the surface. Detecting and handling such cases is an additional challenge. The surfaces of natural objects are often smooth and lack sharp features. We specifically limit the problem domain to non man-made objects without sharp features.

Resampling is the process of finding a new sampling for the underlying surface. The input point cloud is transformed into a new set of points. This is often used to reduce

the number of samples (down-sampling) or increase it (up-sampling). In this thesis, the term is also used for finding new positions for the points, i.e., projecting the points on the approximated surface to reduce the noise.

The meaning of the term **surface reconstruction** ranges from finding a model for the surface (e.g., implicit representation) to generating a triangular mesh of the input points. We use the term local surface reconstruction for the process of finding the local surface of each input point without setting all those surface patches in a global context. Chapter 2 gives an overview of different methods for reconstruction and resampling. Moving least squares (MLS) is a class of methods which find local approximations of the underlying surface. One recent technique using this approach is called *algebraic point set surfaces* (APSS) [GG07]. In this thesis, we use the general approach of APSS to generate local surface patches and to resample the point clouds.

Most surface reconstruction and resampling methods are isotropically biased: In general, the local surface of a point p_i is not weighted fairly, because the sampling is not uniform. Therefore, a region with more dense local sampling has more influence on the reconstruction. Due to the fact that point clouds are not guaranteed to be sampled uniformly, this can lead to reconstruction errors (see Chapter 4.1 for details). To address this problem, we define a new isotropically fair neighborhood definition.

Another challenge for resampling and reconstruction methods is that the noise level and the sampling density can vary drastically between different point clouds. Even in the same point cloud, the accuracy of different points may differ. Most reconstruction methods offer a set of parameters to adjust the behavior of the technique. Although this enables high-quality reconstructions for different point clouds, it requires that the user manually tunes the method for each individual point cloud. In this thesis, we discuss an automatic method that adjusts its behavior to every single point to overcome the requirement for user interaction.

1.3 Contributions

In this thesis a new approach to use isotropically fair neighborhoods is evaluated. It extends the algebraic point set surfaces (APSS) method of [GG07]. The contributions of this thesis are:

- Isotropically fair neighborhoods for local surface reconstruction are introduced. An algorithm is proposed to build isotropically fair neighborhoods incrementally in order to handle non-uniformly sampled point clouds. This type of neighborhood is evaluated by comparison with a state-of-the-art technique that uses isotropically biased neighborhoods.

- An iterative regression method for finding the optimal neighborhood size for each point in the point cloud is proposed and evaluated. The overall spherical regression approach is similar to the method used in [GG07]. Together with the isotropically fair neighborhoods, this succeeds in reconstructing non-uniformly sampled points with arbitrary noise levels without any parameter.

- We propose a local agreement scheme that determines areas of consensus of the normal vector orientation. The computation of this step is very cheap and can be done in parallel. It is integrated into a global approach after the principal idea of [HDD+92] and [XWH+03].
- An alternative approach for noise estimation and reduction is presented. The *Medial Manifold* is defined as the surface that is enclosed by the noise. A method to efficiently calculate this manifold in 2D is presented.

1.4 Structure of the Work

The rest of the thesis is structured as follows: Previous work on the subject of resampling and reconstruction of point clouds is summarized in Chapter 2. Also, the special topic of sphere regression is addressed there.

In Chapter 3, a first approach to estimate the noise level is discussed. It operates by first constructing a noise geometry that encloses the underlying manifold. Then the medial manifold as an approximation for the original surface is found. This approach was successfully implemented in 2D but was not feasible in 3D.

The main contribution is presented in Chapter 4: At first, isotropically fair neighborhoods are motivated and the N-ring neighborhood is introduced. This is then used in an adaptive regression method to estimate the noise level and approximate the surface at each point individually. Finally, the chapter concludes with an explanation of the sphere regression method, which is similar to APSS [GG07] with statistical improvements of [AsC09].

Chapter 5 describes a way to reuse the neighborhood data structures for the global normal orientation problem. A local normal agreement scheme is presented, which generates a local consensus of the normal orientation and is combined with a global approach after [HDD+92] and [XWH+03].

The presented methods are evaluated in Chapter 7: First, the quality of the resampling and the local reconstruction is compared with state-of-the-art techniques. Then, results of the local normal orientation method are shown. Finally, the performance of the individual stages of the reconstruction pipeline is evaluated.

The main findings of the thesis are summarized in Chapter 8. An outlook on improvements and future work concludes the thesis.

CHAPTER 2

Related Work

This chapter summarizes state-of-the-art methods and recent advances in mesh surface reconstruction and point cloud resampling. In this thesis, the term surface reconstruction is used to describe the process of finding a representation – continuous or discrete – of the whole surface. A resampling method generates a new set of points which lies on the surface that was sampled to create the original point set. Resampling has to reconstruct the local surface but does not produce global connectivity information. Therefore, resampling with connectivity information between the points can be seen as (discrete) surface reconstruction.

2.1 Resampling

The process of resampling a point cloud is used for up- and down-sampling and to "clean up" a noisy point cloud, i.e., reducing the noise and generating a more uniform point distribution across the surface.

2.1.1 LOP

One important approach is the locally optimal projection (LOP) [LCOLTE07]: Let $\mathbf{I}, \mathbf{J}$ be two index sets. A set of new points $\mathbf{X} = \{x_i\}_{i \in \mathbf{I}}$ is generated and placed near the original points $\mathbf{P} = \{\mathbf{p_j}\}_{\mathbf{j} \in \mathbf{J}}$. Then, attractive forces towards the original points p_i and repulsion forces between the new points x_i result in a distribution of the points across the surface. This dynamic behavior is defined by an iterative system: The initial positions of the new points are set to the points in $\mathbf{P}$ and are denoted $\mathbf{X^0}$. Then the equation 2.1 is applied until convergence:

$$\mathbf{X}^{k+1} = \sum_{i \in \mathbf{I}} \sum_{j \in \mathbf{J}} \|x_i - p_i\| * \theta(\|\xi_{ij}^k\|) + \lambda_i \sum_{i' \in \mathbf{I} \setminus \{i\}} \eta(\|x_i - x_{i'}^k\|) * \theta(\|\delta_{ii'}^k\|) \tag{2.1}$$

I and **J** are the index sets, $\theta(x)$ and $\eta(x)$ are quickly decreasing functions. $\xi_{ij}^k = x_i^k - p_j$ are the difference vectors between the original points and the new points and $\delta_{ii'}^k = x_i^k - x_{i'}^k$ denote the difference vectors among the new points. λ_i controls the relation between the attraction and repulsion forces.

This procedure has the characteristics of the multivariate L_1 median. It is therefore not sensitive to outliers. In general it is a $O(h^2)$ approximation to the underlying surface, where h is the radius of influence of the fast decreasing function $\theta(x)$. [LCOLTE07] A drawback of the LOP method is that it is computationally expensive and that it is not well-suited if the original points p_i are distributed non-uniformly across the underlying surface. Scanned point clouds often have this characteristic. [ABCO$^+$01] [Bol10].

An extension of this method is WLOP or weighted LOP [HLZ$^+$09]. New weights are introduced that account for the distribution of the input points in space. The weighted local densities are estimated and steer the iterative process. This results in a more even distribution of the new points across the surface. Figure 2.1 shows the effect of the distribution weights: the input points of a scanned surface are not uniformly distributed across the surface. The additional weighting of WLOP ensures that the new points cover the surface evenly.

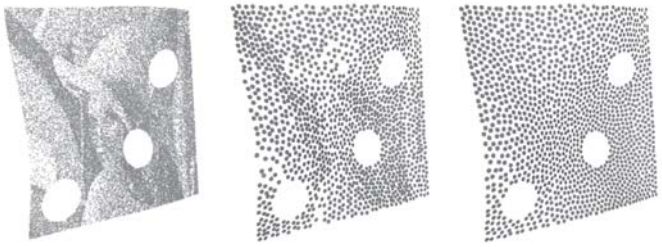


Figure 2.1: Left sub-image shows the input point cloud, the middle point cloud was generated with LOP, the right with WLOP which results in a more evenly distributed point cloud across the surface. (image from [HLZ$^+$09])

The major problem with the LOP approach is its performance. Although the support of the individual points is local, it still requires many iterations and depends on the number of input points. A further advancement of the LOP is CLOP or continuous LOP [PMA$^+$14]. The attraction and repulsion forces are not computed using the input points, but by a Gaussian mixture model. At the beginning, each point is represented as a Gaussian distribution. Then a hierarchical expectation maximization procedure combines the individual distributions to create a continuous representation of the surface. Figure 2.2 visualizes the resulting continuous model of the point cloud.

The mixture of Gaussians allows for a more compact representation and can be implemented efficiently on the GPU, yielding a 7x better performance than WLOP [PMA$^+$14]. Apart from being faster, the method is also more accurate and produces surface reconstructions with higher quality.

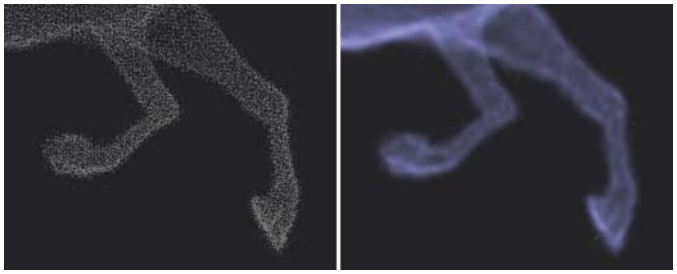


Figure 2.2: Continuous representation of a point cloud (left sub-image) with Gaussian mixture model. (image from [PMA+14])

2.1.2 MLS

A different approach was introduced by [Lev01] and [ABCO+01]: MLS stands for moving least-squares, moving local system or meshless surface [Lev01]. The basic approach is to find a local reference frame and to define a projection on a local polynomial for each input point. Figure 2.3 illustrates the MLS procedure: A planar approximation on a point r is found by minimizing the equation 2.2.

$$\sum_{i=1}^{N}(n \cdot p_i - D)^2\theta(\|p_i - q\|) \tag{2.2}$$

The plane is then defined by $H = \{x|n \cdot x - D = 0, x \in R^3\}$. θ is a decreasing function and weights the distance from each point p_i to q, which is the projection of the current point r onto H. This reduces the influence of distant points and effectively defines an isotropically biased neighborhood. The local reference frame is then centered at q and the heights f_i of the neighboring points to the plane H are calculated. In the next step, a bivariate polynomial g is obtained by minimizing the function 2.3.

$$\sum_{i=1}^{N}(g(x_i, y_i) - f_i)^2\theta(\|p_i - q\|) \tag{2.3}$$

This polynomial approximates the height values relative to the planar reference frame on the projection of point r (q). The final position of r is defined as the projection of r onto the polynomial in regard to H by $q + g(0, 0)n$; i.e., the height of the final point is obtained by evaluating the polynomial on position $(0, 0)$ and extruding the projected point q along the normal vector of H.

The choice of the weighting function θ has a major impact on the smoothing characteristics of the algorithm. It is often set to the Gaussian function $\theta(x) = e^{\frac{-x^2}{h^2}}$, where h denotes the window or feature size. A large value for h results in higher smoothing of the surface.

One limitation of the MLS method is that it is targeted towards smooth surfaces and therefore cannot reconstruct sharp features. The reason for this is the least-squares

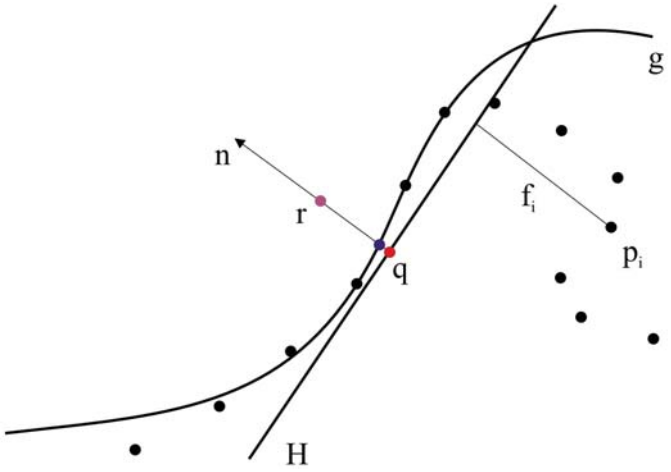


Figure 2.3: Illustration of the MLS procedure in 2D: The local reference frame H is found for a point r and is centered on q, which is the projection of r on H. Then, a polynomial g is found which approximates the heights of the neighboring points p_i to the local reference frame H. Point r is then projected onto the polynomial g to obtain the final position. (image from [ABCO$^+$01])

approximation, which is also sensitive to outliers. The MLS technique can be extended to reconstruct sharp features: In [FCOS05] instead of the least-squares operator, the least median of squares (LMS) is used to obtain a robust approximation of the underlying surface: $\arg_\beta \min_i |f_\beta(x_i) - y_i|$, where β is the parameter vector of the underlying model f. The set of points in the neighborhood of a point r is partitioned into subsets which each describe a local smooth and outlier free surface. The surfaces of each subset are obtained with the LMS method. Outliers are thereby removed and a robust estimate of the local surface is calculated. Then the point r is projected onto the closest local surfaces with a special projection procedure. The drawbacks of this approach are the high computational demand and the complex handling of special cases during the surface classification and projection.

With RIMLS [OGG09], robust implicit MLS, a further extended MLS variant is published, which is designed to handle sharp features. Instead of explicitly modeling multiple surfaces and applying complicated projection operators on them, RIMLS defines the minimization function as a robust local kernel regression problem. Instead of the least squares estimator of MLS, a different objective function is minimized, which is also differentiable but assigns a small weight to outliers. This results in better reconstruction quality for objects with sharp edges and other non-differentiable features.

The original MLS first determines a local planar reference frame and then fits a non-linear surface to the points relative to the local plane. This reduces the number of variables of the non-linear polynomial to 2. APSS (algebraic point set surfaces) [GG07] is an extension of the MLS idea to directly fit higher order surfaces to the underlying points without the need of a planar reference frame. This improves the quality and

stability of low sampled point clouds and increases the reconstruction of geometry with high curvature.

In this thesis an approach similar to APSS is presented. The key ingredient for this method is the spherical regression of point sets. The next section will give an overview of the state-of-the-art of robust spherical regression.

2.1.3 Circular and Spherical Regression

The goal of MLS methods like APSS [GG07] is to create an efficient and exact approximation of the local surface at each input point. Fitting a geometric model to a set of points is equivalent to regression in statistics. The simplest model is the plane (or line in 2D) and can be calculated with principal component analysis. The planar approximation of the surface is often sufficient and is used in [HDD+92]. Higher order models offer more flexibility but are more complex to compute: The effort for fitting m points to a surface of order n is $(m+n)n^2$ [Pra87]. Spheres (or circles in 2D) are used quite often, since they have higher flexibility and offer a tighter fit to curved data, while having reasonable computational cost. According to [AS13] there is no unambiguous best-fitting quadric regression model but that it strongly depends on the local behavior of the surface. Thus there is no single regression model which has a best fit for arbitrary objects. Furthermore, quadrics behave similarly on a small scale (locally). Therefore the rather simple sphere is preferable for general local surface regression.

Another advantage of spheres is that the radius is a good estimation of the curvature of the local surface. This section gives an overview of circular and spherical regression.

Fitting a circle to a set of points is a well studied problem in statistics. (See e.g., in [Pra87], [AsC09]) The equivalent problem in R^3 is fitting a sphere to a set of 3-dimensional points. In general, there are two different approaches to the solution of this problem: geometric fitting and algebraic fitting. Note that in both cases the noise is assumed to be an isotropically distributed Gaussian function.

Both fitting algorithms minimize a specifically defined distance of the points to the surface of the sphere.

$$F_r = \sum_{i=1}^{n} d_i^2 \rightarrow \min \tag{2.4}$$

In equation 2.4 the general minimization problem is stated: Minimize the sum of all squared distances d of every element in the set with n points.

In **geometric fitting** the geometric or Euclidean distance is used. The sphere is represented by the equation 2.5:

$$(x-c_x)^2 + (y-c_y)^2 + (z-c_z)^2 = r^2 \tag{2.5}$$

Where c_x, c_y, c_z are the coordinates of the center of the sphere and r denotes the radius. For geometric fitting the distance function of the sphere is defined in equation 2.6:

$$d_i = \sqrt{(x_i - c_x)^2 + (y_i - c_y)^2 + (z_i - c_z)^2} - r \tag{2.6}$$

Where x_i, y_i, z_i are the coordinates of point p_i of the point set.

If this geometric distance metric is used in equation 2.4, the resulting minimization function is a non-linear least squares problem which has no closed form solution. It can be approximated with iterative methods like Gauss-Newton. These are computational expensive and may have bad convergence properties [AsC09]. The geometric fitting process is particularly unstable if the points lie approximately in a plane.

The obvious advantage of geometric fits is that they are most accurate, which means that they equal the maximum likelihood estimation of the sphere parameters (at least for the assumption of Gaussian noise) [AsC09].

Algebraic fitting methods solve the problem for a different distance definition. Although the chosen distance function approximates the geometric distance, it has better algebraic properties. This leads to an elegant reformulation of the minimization problem that can be transformed into an eigenvalue problem.

In [Pra87] the following parametrization of a sphere (as a replacement to equation 2.5) is chosen:

$$Aw + Bx + Cy + Dz + E = 0 \qquad \text{where } w = x^2 + y^2 + z^2 \tag{2.7}$$

In this representation a sphere is not defined by the 4-dimensional parameter vector (x, y, z, r) but by the 5-tuple $\mathbf{P} = (A, B, C, D, E)$. If $A = 0$ the equation describes a line. Therefore this representation can also handle collinear points.

The algebraic parameters can be transformed back into the "natural" representation of the sphere with equation 2.8:

$$c_x = \frac{B}{-2A} \quad c_y = \frac{C}{-2A} \quad c_z = \frac{D}{-2A} \quad r^2 = \frac{B^2 + C^2 + D^2 - 4AE}{4A^2} \tag{2.8}$$

With this notation the general algebraic function to be minimized is stated as follows:

$$F_a = \sum_{i=1}^{n} [Aw_i + Bx_i + Cy_i + Dz_i + E]^2 \rightarrow \min \tag{2.9}$$

Several algebraic spherical (or circular) fits have been proposed (see [CL05] for an overview of different circular fits). All of them use the same minimization function (equation 2.9) and sphere representation (equation 2.7). The important and distinctive part is the constraint under which the function is to be minimized. One job of the constraint is to rule out the trivial solution to the minimization problem ($A = B = C = D = E = 0$). Other important properties of the constraint are to avoid degenerate spheres (r=0) and to ensure that the solution of the algebraic problem is close to the geometric solution (which is equal to the maximum likelihood estimation).

An example of a spherical fit is [Pra87], which is used in APSS ([GG07]). Equation 2.10 gives the constraint for this spherical fit. This constraint is derived directly from

the definition of the algebraic spheres and particularly its back-transformation (Equation 2.8): The algebraic sphere represents a "real" sphere if and only if the expression $B^2 + C^2 + D^2 - 4AD > 0$ holds true, since this value, the squared radius, has to be positive. Equation 2.10 by [Pra87] expresses this as an equality constraint: The algebraic spherical representation is invariant under scalar multiplication, so the actual value is not important (see Equations 4.13 and 4.14).

$$B^2 + C^2 + D^2 - 4AD = 1 \tag{2.10}$$

It is a convenient property of the algebraic sphere representation that the minimization function can be written in concise matrix notation:

$$\mathbf{Z} = \begin{bmatrix} w_1 & x_1 & y_1 & z_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_n & x_n & y_n & z_n & 1 \end{bmatrix} \qquad \mathbf{M} = \frac{1}{n} Z^T Z \qquad \mathbf{N} = \begin{bmatrix} 0 & 0 & 0 & 0 & -2 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -2 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.11}$$

The $5x5$ Matrix $\mathbf{M}$ contains the moments of the matrix $\mathbf{Z}$. The expression $\mathbf{P^T M P}$ represents the general algebraic minimization function (see equation 2.9). The constraint 2.10 by [Pra87] is contained in the matrix $\mathbf{N}$.

The algebraic sphere regression problem can now be formulated as a constraint optimization problem:

$$F_{\text{opt}} = \mathbf{P^T M P} \rightarrow \min \tag{2.12}$$

$$\text{s.t. } \mathbf{P^T N P} = 1 \tag{2.13}$$

The equality constraint can be included in the objective function with Lagrange multipliers (introduction of the variable d in equation 2.14):

$$F_u = \mathbf{P^T M P} - d(\mathbf{P^T N P} - 1) \tag{2.14}$$

Differentiating this equation with respect to P leads to the generalized eigenvalue problem:

$$\mathbf{MP} = d\mathbf{NP} \tag{2.15}$$

Which can be solved by the eigenvalue problem:

$$(\mathbf{N^{-1}M})\mathbf{P} = d\mathbf{P} \tag{2.16}$$

The columns of $\mathbf{P}$ are the eigenvectors of the matrix $\mathbf{N^{-1}M}$. Each eigenvector contains the algebraic representation of a sphere ($\mathbf{P_i} = (A_i, B_i, C_i, D_i, E_i)$). The best fitting sphere, i.e., the sphere minimizing the distance defined in equation 4.7, is the

eigenvector corresponding to the smallest positive eigenvalue [AsC09]. Since the matrix $\mathbf{M}$ is symmetric and positive definite and $\mathbf{N^{-1}}$ is defined as in equation 2.11, the following holds true: Four eigenvalues of $\mathbf{N^{-1}}$ are positive, one is negative. According to Sylvester's law of inertia, multiplication with a symmetric positive definite matrix does not change the signs of the eigenvalues (see proof in [AsC09]). Therefore there are also four positive and one negative eigenvalue in $\mathbf{N^{-1}M}$. If the eigenvectors are sorted according to the ascending eigenvalues, the desired eigenvector is the second column of the matrix $\mathbf{P}$. This 5-dimensional vector $\mathbf{P_i}$ can then be transformed to the "natural" parameters of the sphere with equation 2.8.

This approach reduces the spherical regression to an eigenvalue problem. In contrast to the geometrical fitting process it does not require an iterative process with uncertain convergence characteristics. There are several algebraic fits available, see [CL05] for an overview of circular fits. They all use the same matrix structure and differ only in the constraint matrix $\mathbf{N}$ in Equation 2.11.

In this thesis a circular fit "Hyper Fit" by [AsC09] is extended to 3D. A reformulation of the minimization problem (Equation 2.15) makes it possible to solve a symmetric eigenvalue problem instead of the general asymmetric problem. (See Chapter 4.3)

2.2 Surface Reconstruction

Surface reconstruction aims to obtain a representation of the complete surface of an object. The most common representation of geometry in 3D graphics is the polygonal mesh, which is a piecewise linear 2-dimensional surface embedded in R^3. Methods which directly produce meshes are called interpolating methods, since they work directly on the input points and generate connectivity information for the point cloud. Implicit methods build an implicit, continuous model of the surface of the object. This implicit representation has to be discretized and turned into a triangular mesh. The following sections cover the two categories of surface reconstruction methods.

2.2.1 Interpolating Methods

Interpolating methods find a piecewise interpolation of the input point set $\mathbf{P}$. Starting point of those methods is often the Delaunay triangulation and the Voronoi diagram, where one is the dual of the other. The Voronoi cell of a point $p \in \mathbf{P}$ is defined as $V_p = \{x \in R^3 : \forall q \in \mathbf{P} \backslash \{p\}, \|x-p\| \leq \|x-q\|\}$. The facets and edges between the Voronoi cells define the Voronoi diagram. The Delaunay triangulation DT (or tetrahedralization in R^3) is a subdivision of the space into cells (tetrahedrons) such that no point is in the circumsphere of any other cell. This maximizes the minimum angle of the triangles in DT.

Cocone and its extension, *Tight Cocone* [DG03] use these two definitions to find a watertight surface triangulation of $\mathbf{P}$. The Voronoi cells of a point p are in general elongated in the direction of its normal vector n_p of the underlying local surface S. The cocone of p C_p is then defined as the region between the two cones at the point p (see

Figure 2.4). The intersection between triangles in DT and C_p produces a set of candidate triangles. The final surface is then determined in a manifold extraction step.

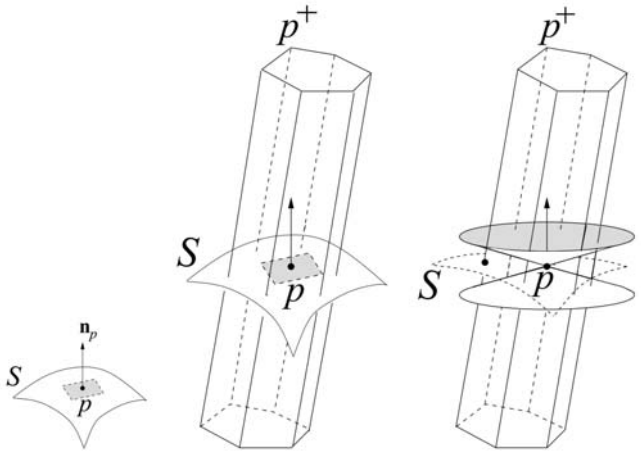


Figure 2.4: Left image shows the true surface S around a point p and its normal vector n_p. The Voronoi cell of p is illustrated in the middle image. The most distant point of the Voronoi cell to p is denoted p^+. Notice how the elongation of the Voronoi cell corresponds to the normal vector n_p. The right image shows the cocone of p: The area between the two cones touching at p defines the cocone. (image from [DG03])

This approach only works if the sampling satisfies certain conditions which are often not met in real world point clouds. A modified version of this algorithm [DG03] detects under-sampled regions and leaves holes there, instead of adding wrong triangles to the final surface. Then the erroneous parts of the triangulation are repaired to obtain a water tight surface for the point cloud.

The *Tight Cocone* works well for evenly sampled objects and even for cases with local sparse regions. In general, sparsely sampled point clouds are problematic and introduce artifacts to the reconstructed triangulation. The method published by [OMW13] is specifically designed to process point clouds of different sampling densities. The general approach is to find the closed triangulation which minimizes a global minimization function. As a criterion for each triangle the maximum edge length is chosen. The minimum boundary B_{min} is defined in equation 2.17.

$$B_{min} = \arg\min_{B \in \mathbf{B}} \sum_{t \in B} \max_{e \in t} \|e\| \tag{2.17}$$

$\mathbf{B}$ is the set of all non-self-intersecting closed triangulations of the input point set. Since finding the minimum of equation 2.17 is NP-hard, an approximation of B_{min} is calculated instead, the boundary complex BC_{min}. BC is a subset of the Delaunay triangulation of the point set such that every vertex in BC has at least one umbrella in BC. An umbrella of a vertex v is defined as a closed triangle fan centered at v. So instead of finding the minimum triangulation over all elements in B as in equation 2.17,

the best triangulation in BC is obtained. The choice of the Delaunay triangulation for the BC is motivated by the minimizing criterion of equation 2.17: The Delaunay triangulation also minimizes the maximum edge length.

The boundary complex BC_{min} can be approximated well and then is the starting point for further topological operations: Artifact boundaries are determined and hull holes are filled. An *inflate* step produces a manifold triangulation and a *sculpturing* step is used to find the final interpolation of the point set. Figure 2.5 illustrates the process of turning the boundary complex into the final triangulation in 2D.

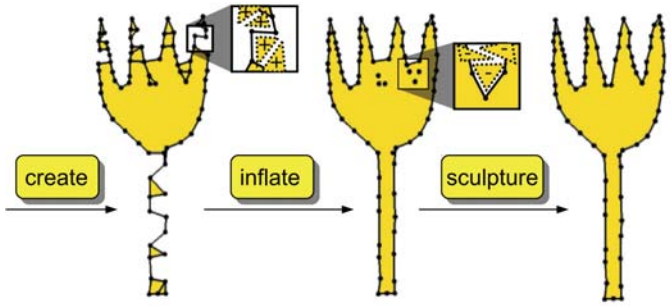


Figure 2.5: Basic topological operations on the boundary complex in 2D to generate the final interpolation of the point set: The *inflate* step turns the degenerated hull into a manifold (center image). *Sculpturing* exposes the interior points to the boundary (right image). (image from [OMW13])

The method proposed by [OMW13] produces good results even in the presence of sparse sampled point clouds. One drawback of the technique is that it requires global data structures (the Delaunay triangulation) to generate the boundary complex and is therefore not suited for a fast parallel implementation.

Adamy et al. [AGJ02] describe an approach of choosing a manifold locally for each vertex: The umbrella is defined as the triangles from the Delaunay triangulation which are incident to a vertex and satisfy a certain criterion. This definition is then used by [KA08] to circumvent the need to construct a global Delaunay triangulation: For each vertex v of the input point cloud, a heuristic generates a set of candidate triangles T_v incident to v based on the nearest neighbors. Then elements of T_v are removed until a triangle fan centered at v remains – the umbrella. Under a certain sampling condition this method succeeds in generating a triangulation covering the whole point cloud. The advantage of this is that the computations are independent for each vertex and can therefore be executed in parallel. The major shortcoming of this technique is that it relies on the quality of the sampling of the input and also on a good estimate of the spacing between the points.

2.2.2 Implicit Methods

In contrast to interpolating methods, implicit techniques do not produce a mesh by generating connectivity information between the input points. In a first step, a functional model of the underlying surface is defined based on the input point cloud. This analytic representation then has to be discretized into a triangular mesh. The surface of the object is described by an implicit function such that the zero-set of this function represents the boundary of the object (the desired surface). The quality and performance of this class of methods is dependent on the used functional model and the discretization method.

One class of methods of implicit surface reconstruction is **RBF**, radial basis functions [CBC+01]. Techniques of this class require points with oriented normal vectors as input. In a first step, the point cloud is expanded with additional points. For each point two vertices are created along the normal vector: one in the positive direction, one in the negative direction with a certain offset ϵ. Figure 2.6 illustrates the resulting extended point cloud $\mathbf{P}_{\text{ext}}$, which forms a volume around the actual surface. The value of ϵ has to be chosen s.t. the outer and inner boundary do not intersect with each other and with the actual surface.

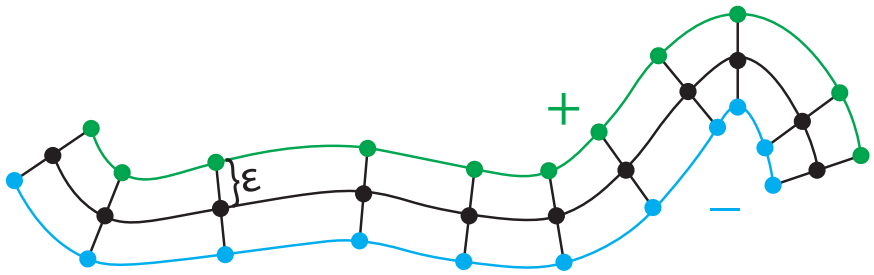


Figure 2.6: Extending the input point set (black) for RBF: creating inner (green) and outer points (blue) along the normals. The value of ϵ has to be chosen such that no intersections occur.

In the next step, a specific value is assigned to every point of $\mathbf{P}_{\text{ext}}$: 0 to the input points, -1 and $+1$ to the outside and inside points, respectively. The most expensive procedure of RBF reconstruction is to find an interpolating function $f : R^3 \rightarrow R$ which evaluates to the specific values for all points in $\mathbf{P}_{\text{ext}}$. This interpolant is composed of multiple radial basis functions:

$$f(x_i) = \sum_{j=1}^{N} c_i \phi(\|x_i - x_j\|) = y_i \quad x_i \in \mathbf{P}_{\text{ext}} \quad \forall i \in [1, .., N] \tag{2.18}$$

Points x_i and x_j are from the extended input set $\mathbf{P}_{\text{ext}}$ and y_i are the respective values $\in \{-1, 0, 1\}$. The function ϕ is a radial basis function, for example the Gaussian function. The coefficients c_i determine the interpolating function and are found by solving a system of linear equations. In order to decrease the runtime of the method,

the number of basis functions can be reduced: Instead of using all input points as centers for the basis functions, only a sub-set is used. (see [CBC+01] for details).

The main problem of this method is the computational effort: Solving the system of equations requires $O(N^3)$ operations, where N is the number of basis functions. Solving the system of equations in parallel using the GPU [CGGS13] can increase the performance. Still, it requires a few hundred seconds to find the interpolating RBF representation for even small (≈24000 vertices) point clouds [CGGS13]. Furthermore, this technique relies on oriented normal vectors and multiple parameters: ϵ for the extended point cloud, the amount of support of the radial basis function and the size of the evaluation grid (discretization).

Poisson surface reconstruction [KBH06] has the same goal as RBF: defining an indicator function that defines an inside, outside and the actual surface. Instead of building an interpolating function with non-local support for each input point, the whole problem is considered as a Poisson problem (partial differential equation): $\Delta f = \nabla V$, where Δ is the divergence of the gradient field of a function and ∇ is the divergence of a vector field. That means, the gradient of the scalar function f (f is the approximation of the indicator function of the surface) is estimated by the vector field V, which is given by the input points and their normal vectors. The inverse of the gradient of f is then the approximation of the indicator function, which can then be used to extract the surface of the object. The Poisson equation is used in physics and other domains and is well researched.

One shortcoming of the Poisson reconstruction is that it tends to over-smooth the data. Kazhdan et al. [KH13] published the screened Poisson reconstruction which introduces an interpolation constraint. A point weight factor determines how closely the surface passes through the input points. Figure 2.7 shows an example of this behavior in 2D: The red graph shows the reconstruction with Poisson reconstruction, the blue line was obtained with screened Poisson reconstruction. With the interpolation constraint, the method creates a better approximation of the input points and prevents over-smoothing of the data. This approach allows the formulation as a sparse problem and can be solved efficiently. Further optimizations by [KH13] result in a multi-threaded implementation which scales linearly with the number of input points.

Setting the point weight (parameter of the interpolation constraints) to a high value ($\geq$ 16) results in a surface that goes through the input points. While this may not be desirable in all cases, it can be used to simulate the behavior of an interpolating technique. In this thesis, screened Poisson reconstruction is used to assess the quality of our resampling and normal computation.

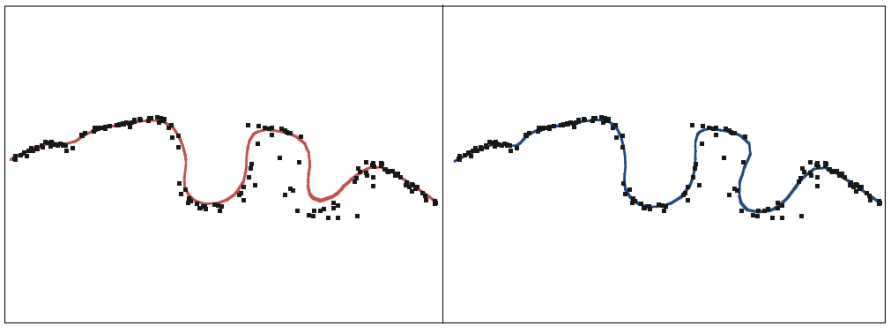


Figure 2.7: Comparison of points and the reconstructed surface projected on 2D: Left sub-image: Poisson reconstruction tends to over-smooth the data-points. Right sub-image: screened Poisson reconstruction has a better approximation of the underlying surface due to the interpolation constraint. (images from [KH13])

CHAPTER 3

Resampling on Global Manifold

The problem of point-cloud resampling is a well-researched topic. See Chapter 2 for an overview of state-of-the-art methods. In the context of this thesis, resampling describes the process of denoising a point cloud, i.e., reducing the noise while preserving the details and structure of the point data. In this thesis, two approaches are discussed: Chapter 4 describes a local, regression-based approach. In this chapter, we describe our first attempt, a global approach:

The method starts by creating a global approximation of the surface, and then, resampling is performed on this approximated manifold. This approach can then be easily extended to a full surface-reconstruction method: Once the global connectivity of the points is established, the resampled points constitute the reconstructed surface.

The challenging problem is to find a well-suited approximation of the original manifold based on the set of input points $\mathbf{P}$. If a sampling condition is fulfilled, the surface can be reconstructed by computing local umbrellas (see Section 4.1.1 for the definition and computation thereof). Umbrellas are basically local approximations of the global surface. The union of all local umbrellas is then the reconstruction of the point cloud. This works only under special conditions: a low noise level and uniform sampling are required. See [GKS01] for specific sampling conditions. In [KA08], a similar local-umbrella reconstruction technique is described.

Point clouds which satisfy the specific sampling conditions can be reconstructed and resampled efficiently with such local umbrella methods. This section describes a unique approach to tackle the noise estimation and surface reconstruction problem for those areas which cannot be reconstructed by local umbrellas. The basic idea is to model the noise explicitly as geometry and reconstruct the surface by calculations on the noise geometry. Section 3.1 gives an overview of the idea and lists the assumptions and simplifications that were made. Also, a formal definition of the "medial manifold" is given. The construction of this manifold surface in 2D is described in Section 3.2. Finally, Section 3.3 gives reasons why this approach was not carried out in 3D and the limitations of this idea.

3.1 Idea and Definition

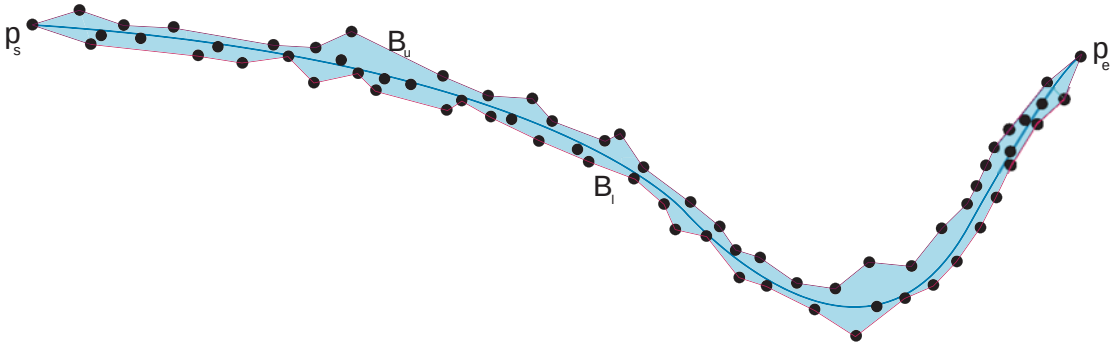


Figure 3.1: Visualization of the noise geometry in 2D: The dark blue line represents the actual surface, the sample points are distributed around the original manifold. Connection of the outer points results in the light blue noise volume (noise area in 2D). B_u represents the upper boundary, B_l the lower boundary of the noise geometry. The two sets are connected by the start- and endpoint p_s, p_e.

The surface of a model is a 2D manifold embedded in the 3D space. Sampling the surface (with 3D scanners) introduces errors or deviations from the actual surface. Under the assumption of uniformly distributed Gaussian noise, the resulting samples are equally distributed between the two half-spaces divided by the surface (i.e., in- and outside for an unbounded manifold). The hull of these points forms a volume in which the actual surface resides. Figure 3.1 illustrates this situation in 2D: Sampling the original surface generates the point set, which is not exactly on the manifold but forms a "tube" around it.

The goal of this method is to generate a noise volume, based solely on $\mathbf{P}$, and then define a manifold that lies within this volume. This manifold is called "medial manifold" and represents the reconstructed surface. Intuitively speaking, it is the manifold that lies in the middle of two outer manifolds. For the 2-dimensional space, the medial manifold is an edge chain. It is defined as follows:

Definition 3.1.1. Let a closed piece-wise linear curve C decompose itself into two non-empty subsets, the boundaries B_u and B_l, separated by the end points p_s, p_e. The medial manifold MM of C is then the set of points inside the area enclosed by C for which a maximum circle centered in the point touches both B_u and B_l. MM is a singly connected piece-wise linear curve between p_s and p_e.

This definition is related to the medial axis: It is more restrictive than the medial axis, since a point on MM has to be equidistant to the two distinct boundaries B_u and B_l, instead of two arbitrary points on C. Therefore, the set MM is a subset of the

corresponding medial axis, in which the small branches of the medial axis, which refer to the same boundary (B_u or B_l), are removed.

According to this definition, it is not required that the two sets B_u and B_l are disjoint. Therefore, it is possible that they share points and collapse to one line. In those areas, the noise is assumed to be 0 and $B_u = B_l = MM$. As an additional constraint, we state that the two lines B_u and B_l must not cross each other.

Definition 3.1.1 assumes the knowledge of C and its decomposition into two curves B_u and B_l. Section 3.2 gives a possible heuristic for their computation.

3.2 Construction in 2D space

In order to find the surface reconstruction of noisy and not uniformly sampled points, we calculate the "medial manifold" (see definition 3.1.1). In a first step, a boundary C and its partition into two sets B_u and B_l (see Figure 3.1) has to be found for the point set $\mathbf{P}$. We used a simple heuristic that relies on a dense and regular sampling.

Approximation of the outer boundaries B_u and B_l

The outer boundaries are closed, piecewise linear curves. Our approach first generates a set of edges $\mathbf{E}$ connecting the points of $\mathbf{P}$. The resulting graph is required to be connected such that a closed, global boundary can be found. The boundary C is then extracted from the edge set. A simple approach would be to connect each point of $\mathbf{P}$ with its k-nearest neighbors (k-NN). This has the drawback that the resulting graph is not guaranteed to be connected. Because of this limitation we do not use this approach.

Instead, we set $\mathbf{E}$ to be the edges of the Delaunay triangulation $\mathbf{DT}$ of the input point set $\mathbf{P}$. This is guaranteed to generate a connected graph. A drawback (for our case) of the Delaunay triangulation is that it contains the convex hull of $\mathbf{P}$. Its boundary is in general not a suitable approximation for B_u and B_l. Figure 3.2 depicts the Delaunay triangulation of a sampled curve in 2D. It can be observed that many edges of the Delaunay triangulation are well suited to approximate the outer boundaries of the manifold. On the other hand, a few long edges connect far away points to obtain the convex hull. In order to get a tight fit to the original surface and to avoid such connections, the edges exceeding a certain length are removed. A possible threshold can be computed with: $t_l = \text{median}(\text{length}(\mathbf{E})) * c$. The median is well suited for this purpose since it is robust against outliers (long edges). The choice of the weight c defines the amount of smoothing. In our experiments, a value between 8 and 15 works well for the parameter c. Filtering the edges of $\mathbf{DT}(\mathbf{P})$ with threshold t_l results in edge set $\mathbf{E}'$. The right image of Figure 3.2 shows the result of filtering the edges with a value of $c = 10$. This heuristic assumes that the number of short edges is significantly higher than the number of long edges. This requires a dense and regular sampling with respect to the feature size. With sparse sampling, some of the long edges that are important connections (i.e., define a feature of the geometry), would be removed. Also, it may fail for certain geometric

structures: If the underlying manifold touches itself on some positions (e.g., a loop), the resulting edge set $\mathbf{E}'$ cannot recover the connectivity of the manifold correctly.

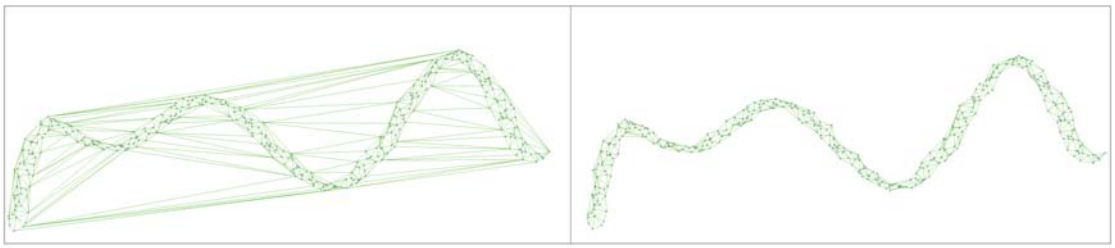


Figure 3.2: Left: Delaunay triangulation of the samples. Right: Removing all edges, that are longer than $10 * \text{median}(\text{length}(\mathbf{E}))$, approximates the underlying manifold.

The extraction of C out of $\mathbf{E}'$ is done by selecting one starting point p_s. This point has to be adjacent to the outside of $\mathbf{P}$, i.e., a point outside of the convex hull of $\mathbf{P}$ can be connected with p_s with a line l_{p_s} such that l_{p_s} does not intersect any element of $\mathbf{E}'$. Starting at p_s, the outer edges of $\mathbf{E}'$ are traversed: At each point, all incident edges are sorted according to their angle with respect to the direction of traversal. Then, the edge with the smallest angle is added to C and the next point on the boundary is reached. This procedure generates a closed line chain, which is the outer boundary of $\mathbf{E}'$.

Finally, C is partitioned into its subsets B_u and B_l. These subsets are connected at the point p_s and p_e. The endpoint p_e is chosen in a way that both p_s and p_e divide the set C into two subsets B_u and B_l of approximately the same length.

Calculation of the medial manifold

Since the outer boundaries are piecewise linear, the manifold itself is also composed of linear pieces. Figure 3.3 illustrates the basic principle: Circles are fitted between the two piecewise linear boundaries. Algorithm 3.1 finds the vertices of the medial manifold MM by constructing circles based on the geometric relation between the boundary edges. Consider the circles of MM as a function of a parameter $u \in \{1; 0\}$: $S(u) = [m(u), r(u)]$, where $m(u), r(u)$ are the position of the center and the radius, respectively. As long as the circle touches two lines (of both boundary sets B_u and B_l), the radius changes only linearly with the position of the circle center on MM: Consider two distinct lines a, b: If they are parallel, the enclosing circles between the lines have constant radius. If not, the lines intersect at a point P_i. The centers of all circles that touch a and b lie on the angle bisector of a and b that goes through P_i: For each point $m(u)$ on the angle bisector, the minimum distance to a is equal to the minimum distance to b, which equals the radius of the corresponding circle $r(u)$. Advancing on the angle bisector to a new point $m(u * \mu)$ increases the minimum distance to a and b by μ.

Both $m(u)$ and $r(u)$ are continuous, linear functions, as long as no additional line intersects the corresponding circle. That means that only the vertices of the medial

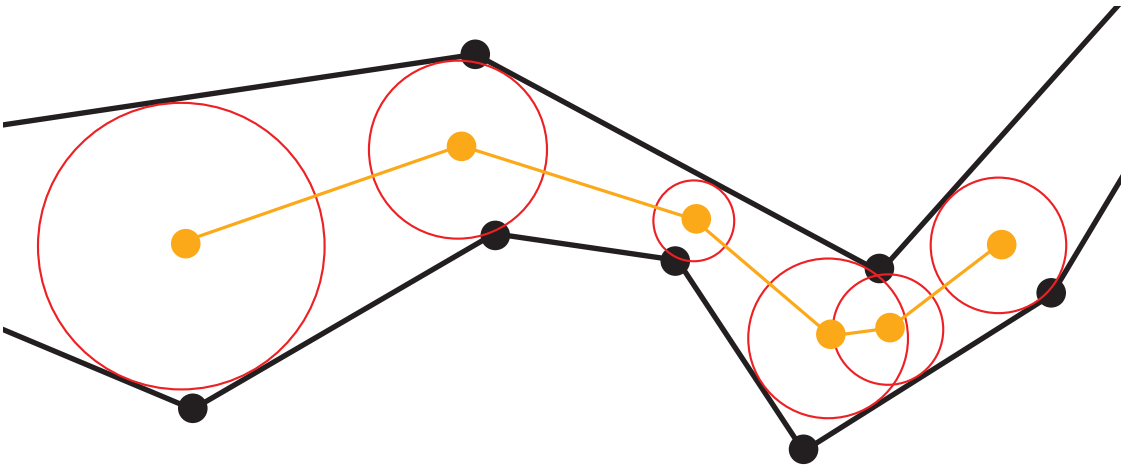


Figure 3.3: Basic principle of manifold construction: Both boundaries are traversed, and at each step a new circle is constructed between the corresponding lines

manifold, i.e., the discontinuities of $S(u) = [m(u), r(u)]$, have to be obtained. The vertices together with their connectivity define MM. The rough outline of the algorithm for calculating the medial manifold can be seen in Algorithm 3.1. Its function 3.1 is stated in Algorithm 3.2.

The basic principle of the algorithm is illustrated in Figure 3.4.

The boundary is traversed on both sides (B_u and B_l) while keeping track of the current active edge on the upper ($u_i \in B_u$) and lower ($l_i \in B_l$) side of the boundary. In each step, the algorithm finds a circle (if possible) that fits between the two active line segments. The center of the circle is a new point on the medial manifold and is connected to the center of the previous circle.

If no previous circle is present (i.e., at the start point p_s or if the two boundaries touch at some point), the intersection between the boundaries is treated as the center of a circle with a zero length radius.

The procedure of finding the next point on the medial manifold is divided into two phases: The first phase is the construction of the initial circle S_i, shown in sub-image (a) of Figure 3.4: The orthogonal vector on the endpoint of edge u_1 is intersected with the bisector of the angle of the two edges u_1 and l_1. This circle is guaranteed to touch the two lines (but not necessarily within the line segments of u_1 and l_1). This is done with both active edges. The circle having a center closer to the previous computed medial manifold circle (MM_i in Figure 3.4(a)) is chosen to be the initial circle.

The second phase (Algorithm 3.2) deals with adapting the initial circle S_i to the edges succeeding the current edges (u_1 and l_1). To achieve this, the initial circle is first intersected with the boundary belonging to the active edge (in this example u_1). If it intersects with the boundary, a new circle is formed using the current edges u_1 and l_1

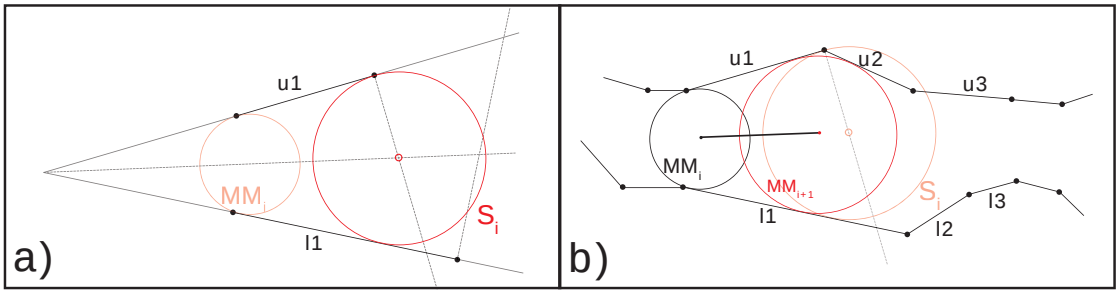


Figure 3.4: Medial manifold generation through circle fitting to two piecewise linear boundaries.

and the intersecting edge. Due to the fact that the boundary consists of line pieces, the limiting entity for a circle can be an edge or one of its end points.

This procedure is repeated until a circle is found which does not intersect the current boundary. Afterwards, the same algorithm is applied using the other boundary (belonging to edge l_1) for intersection. In Figure 3.4(b), S_i intersects the edges u_2 and u_3. The new circle is constructed with edges $u1$, $l1$ (the current active edges) and $u2$ (the intersected edge). Function *construct_circle* of Algorithm 3.2 is a special case of the *Apollonius' Problem* [GR04]. Since we know the orientation of the edges and the rough positioning of the solution circle, there exists a unique solution.

The newly found circle MM_{i+1} does not intersect any boundary edge and is added to the set of medial manifold vertices. The connectivity between the generated circles is stored and the active edges are updated: For each boundary set, the furthest edge touching the new circle is the new active edge. If no new edge is intersected by S_i, the generating edge is set to the next edge on the boundary.

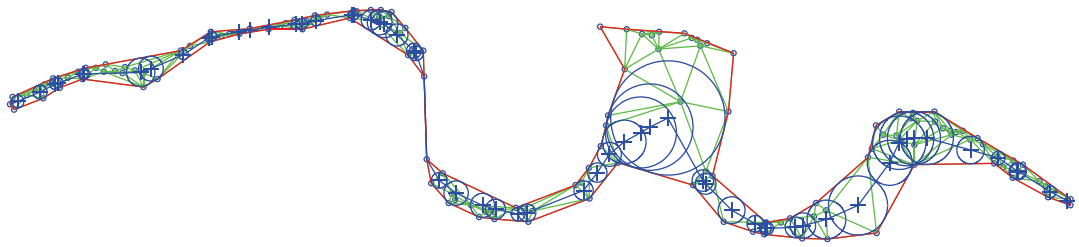


Figure 3.5: Result of the medial manifold calculation

An illustration of the overall construction process of the circles in the noise geometry is shown in Figure 3.3. Examples of point clouds can be seen in Figures 3.5 and 3.6.

Algorithm 3.1: Piecewise linear computation of the medial manifold. *fit_circle* is described in Algorithm 3.2.

Input: Boundary-sets $\mathbf{B_u}$, $\mathbf{B_l}$, starting point p_s, end point p_e
Output: Set **MM** containing the edges of the medial manifold in sorted order

```
1  MM.add([p_s, epsilon]) ;
2  e_u = adjacent_edge(p_s, B_u) ;
3  e_l = adjacent_edge(p_s, B_l) ;
4  while end_reached(e_u) == false && end_reached(e_l) == false do
5  |  S_u = initial_circle(e_u, e_l) ;
6  |  S_l = initial_circle(e_l, e_u) ;
7  |  if distance(S_u, MM.last) < distance(S_l, MM.last) then
8  |  |  [S_f, e_u, e_l] = fit_circle(S_u, e_u, e_l, B_u, B_l) ;
9  |  else
10 |  |  [S_f, e_l, e_u] = fit_circle(S_l, e_l, e_u, B_l, B_u) ;
11 |  end
12 |  MM.add(S_f) ;
13 end
14 return MM ;
```

3.3 Discussion

This section discusses the limitations and drawbacks of the proposed solution and explains why the implementation of Section 3.2 was not extended to 3D.

The prerequisites are non-trivial by themselves: The heuristic for the calculation of the boundaries B_u and B_l of the noise geometry, presented in Section 3.2, has a big influence on the quality of the medial manifold. If insufficiently many edges are removed from the convex hull, important details of the geometry and the point cloud cannot be reconstructed. The removal of too many edges results in a noisy geometry that degenerates to line segments. Therefore, the noise level cannot be assessed and the medial manifold reconstructs only the noise. The problem of finding the appropriate noise level is thereby just shifted into the boundary extraction phase.

During construction of the medial manifold, points with smaller deviation from the true geometry are likely to be ignored, because they are completely contained in the hull, supported by neighbors with higher noise extent. This means that points with bigger deviations have significantly more influence on the resulting medial manifold. This procedure is not robust and very sensitive to outliers, as they can distort the noise geometry arbitrarily – and therefore also the medial manifold.

There are multiple reasons why the extension of the present method to 3D is not easy: The most obvious one is that the implementation of Section 3.2 relies on the notion of direction. A curve AB in 2D has two directions: From A to B or vice versa. By defining one of them to be the general direction (from p_s to p_e in Section 3.2) the connectivity of the constructed circles, as well as the set of next active edges, is always clear. The

Algorithm 3.2: *fit_circle* of Algorithm 3.1. Calculating the best fitting circle based on an initial circle S_i, the generating edge e_{cis} and its opposite e_{trans}.

Input: Initial circle S_i, generating edge e_{cis} and its opposite e_{trans}, boundary edge sets B_{cis} and B_{trans}
Output: Fitting circle S_f, new active edge e_{active} and new opposite edge e_{opp}

```
1  e_active = e_cis ;
2  I_cis = intersecting_edges(S_i, B_cis) ;
3  for e_i ∈ I_cis do
4  |   e_active = e_i ;
5  |   S_i = construct_circle(e_i, e_cis, e_trans) ;
6  |   I_cis = intersecting_edges(S_i, B_cis) ;
7  end
8  e_opp = e_trans ;
9  I_trans = intersecting_edges(S_i, B_trans) ;
10 for e_i ∈ I_trans do
11 |   e_opp = e_i ;
12 |   S_i = construct_circle(e_i, e_active, e_trans) ;
13 |   I_l = intersecting_edges(S_i, B_trans) ;
14 end
15 S_f = S_i ;
16 if e_active == e_active then
17 |   e_active = next_edge(e_active) ;
18 end
19 return S_f, e_active, e_opp ;
```

equivalent simplex of a line in 3D is the triangle. Instead of having a start and an endpoint, a circle of edge segments would be the starting set and the resulting manifold would then be a disc (topologically). Since there are infinitely many possible directions along the surface of a disc, there is no easy way to define a unique direction. Instead, an advancing front mechanism would be needed. This, on the other hand, makes the whole selection of the next facets (the 3D equivalent of line segments) hard.

Finding the appropriate spheres between two piecewise linear boundaries (triangle sets) is also more complex: In two dimensions, circles are constrained by edges, i.e., the supporting lines or the respective endpoints. In 3D, spheres can be constrained by points, edges (and their supporting lines) and facets (and their supporting planes). This requires a far more complex code-path to cover all possible cases. Just consider a vertex on one boundary: In 2D, exactly two boundary-edges are incident to the vertex, and they can have a convex or a concave angle (in relation to the incident boundary). In 3D, a vertex on the boundary can have an arbitrary number of incident boundary triangles forming the umbrella of the vertex. The umbrella itself cannot be classified as convex or concave easily.

The presented method, for both 2D and 3D, is a sequential approach. A local, easily

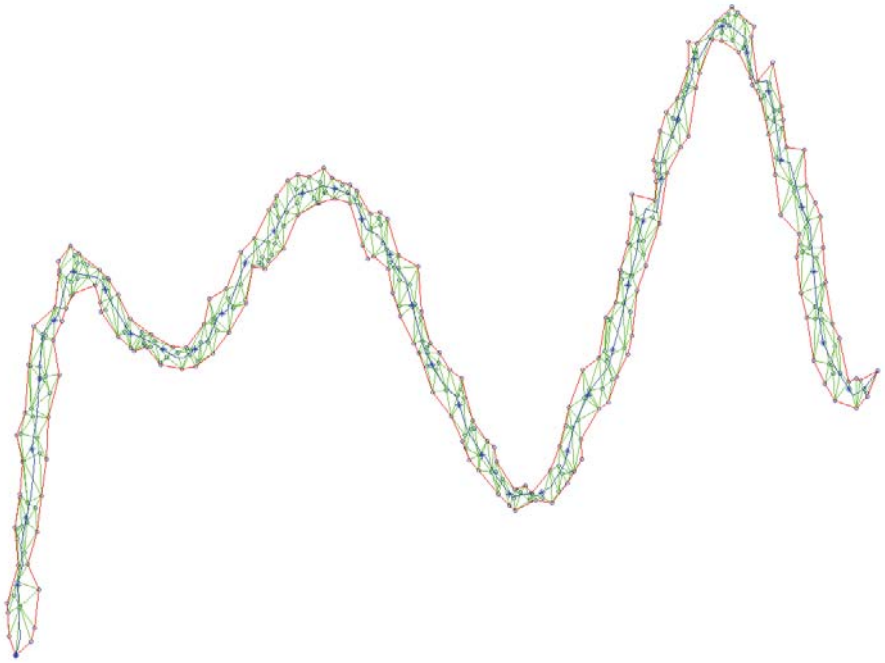


Figure 3.6: Example of the manifold generation of a different point set

parallelizable method is more favorable, because it can benefit from modern graphics hardware and multiprocessor systems.

CHAPTER 4

Local Resampling using Isotropically Fair Neighborhoods

In Chapter 3, a global approach to surface reconstruction and resampling was introduced. This method works for certain assumptions (dense regular sampling) in 2D. Unfortunately, a generalization into three dimensions proved to be difficult. In real-world applications, resampling (denoising of a point cloud) and reconstruction of three-dimensional data is of great importance. The methodology of Chapter 3 is not well suited for this purpose. Hence, a different approach was pursued, which does not suffer from these limitations.

The overall goals for this new technique are that it should be well suited for 3D and can be subdivided into local sub-problems, such that it is scalable for big data sets.

This chapter introduces such a new technique for point-cloud resampling. The underlying principle of this method is based on "Algebraic Point Set Surfaces" (*APSS*) [GG07]: The surface of the scanned object is locally approximated with sphere surfaces which are fitted to the scanned points with non-linear regression. Projecting points on the union of the sphere surfaces produces a resampling of the point cloud. See Chapter 2 for a more detailed explanation of this method.

In this thesis, APSS [GG07] is extended and improved by the following aspects:

- Isotropically fair neighborhood definition
- Adaptive local fitting scheme
- Slightly improved regression method

State-of-the-art surface reconstruction methods use the distance between points to determine the influence (or weight) of a point to its neighbors (e.g., [LCOLTE07], [HLZ+09], [ABCO+01], [GG07]). The most established and intuitive metric used is the Euclidean distance. In APSS, for example, the distances are used as a weighting for calculating

the local regression spheres. For reasons of efficiency, the distances are not actually calculated between all points. Each point has an associated set of points (or neighborhood) which has an influence on the current point. This neighborhood is defined as the k-nearest neighbors with a fixed number k. The parameter k has to be chosen by the user and is used as a trade off between noise reduction and feature preservation. If the noise level and the sampling density are uniform throughout the point cloud, this approach works well. In general, surface patches of an object may have different levels of sampling densities. This leads can lead to artifacts and reduces the overall quality of the reconstruction or the resampling. In Section 4.1, **Isotropically Fair** neighborhoods are introduced and motivated. They are designed to handle irregular and sparse sampling.

In reconstruction techniques like APSS [GG07], the surface is locally approximated by fitting a (mostly non-linear) model to a set of points. The size of this set of points is usually fixed. Common examples are k-nearest neighbors (k-NN) or a ball neighborhood (include all points inside a ball of fixed radius). These fixed-sized neighborhoods define the quality and the smoothing factor of the whole method. This works well for point data sets with constant noise levels, but can lead to artifacts with scanned point clouds of real-world objects, which often have varying noise levels (see [ABCO$^+$01] [Bol10]). In Section 4.2, an adaptive method to solve this problem is described. It estimates the noise level at each point of the input data and increases the neighborhood accordingly.

The actual regression method, which uses an *algebraic* representation of spheres, is very similar to APSS. The special algebraic form leads to a closed-form solution of the non-linear regression problem but also introduces a statistical bias. A circle regression method of Chernov [AsC09] is adapted in Section 4.3 to be used as a sphere estimation method for the local reconstruction. This method has a smaller bias while still providing a closed form solution to the problem.

4.1 Isotropically Fair Neighborhoods

Most methods of the state-of-the-art use a distance function to weight or rank relations between points (mostly either the Euclidean distance or a function of it). All those methods have in common that they apply equal importance to all directions. That means, the distance increases in all directions with the same amount. If a neighborhood is defined by such a function (e.g., k points with the smallest distance to a reference point), we call it **isotropically biased**. The neighborhood is influenced by the sampling of the points in this region, therefore it is biased. Our goal is to find a distance function (note that it is not a real metric in the mathematical sense) and a corresponding neighborhood that is less influenced by the sampling. We assume that the point cloud was obtained by sampling a closed manifold surface. That means that we do not cover the case of holes in the geometry.

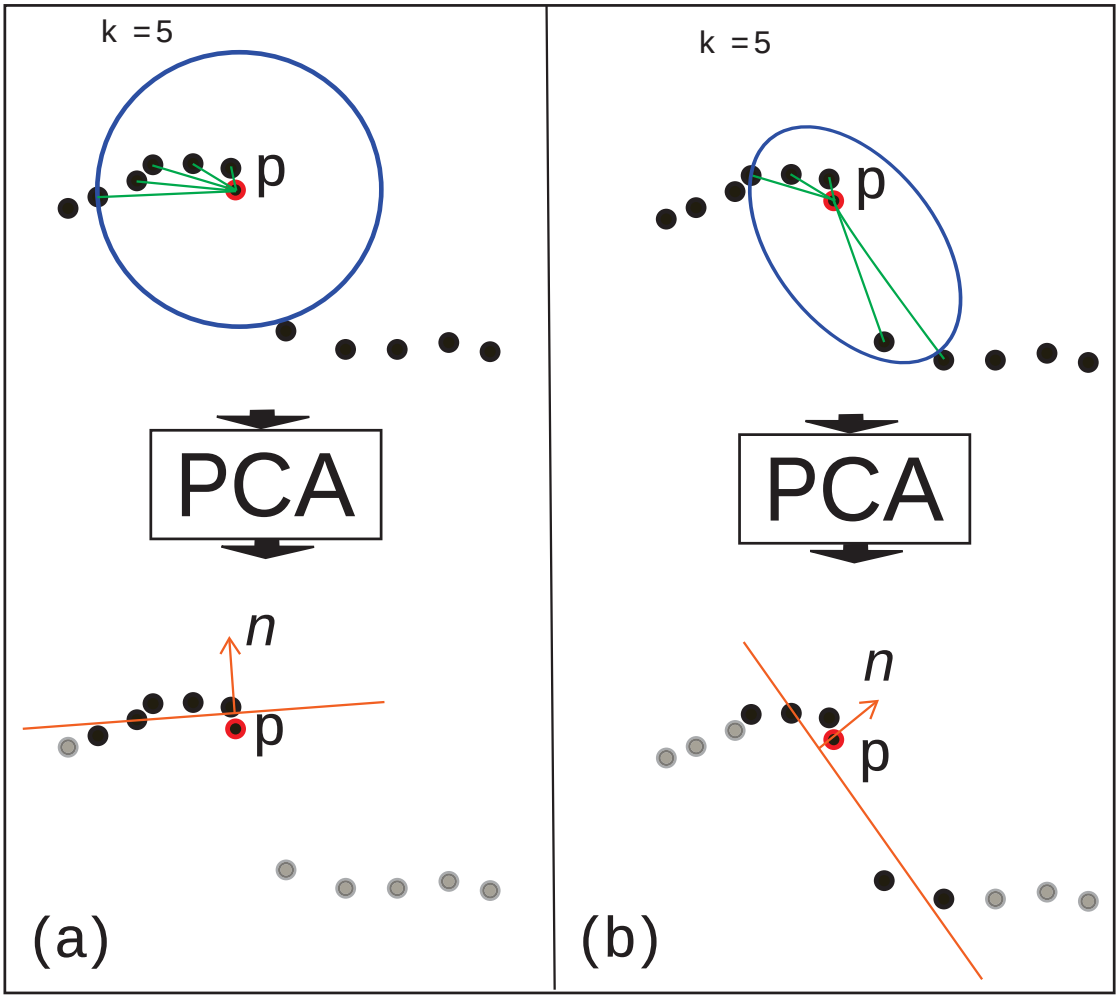


Figure 4.1: Sub-image (a): isotropically biased neighborhood, only distance is taken into account. Sub-image (b): isotropically fair neighborhood classifies points according to multiple parameters.

Figure 4.1 illustrates the overall problem in 2D: In the upper sub-images, the black dots form the point cloud of a sampled line. Due to scanning artifacts there is a gap in the middle of the point set. The usual approach (left sub image of Figure 4.1) takes a fixed-size neighborhood (in this case $k = 5$) and calculates the regression line with PCA (principal component analysis). The resulting normal vector is directed straight up, although the gradient of the line – which was sampled in the beginning – would be different. The right side of Figure 4.1 displays a different neighborhood, which adapts to the local sampling. This neighborhood takes the same number of points and uses the same model (PCA) but produces a regression line which is much closer to the original geometry. Note that in 2D, we assume that the desired geometry is a closed curve

without holes.

In this thesis we use the term **isotropically biased** for the standard, Euclidean distance-based neighborhoods.

Let's consider a point on the reconstructed surface. The points in the vicinity of the reference point are weighted by the distance between them and the reference point. If the weights are not distributed evenly among all directions, the coverage of the weighting is biased by the sampling of the surface. In contrast to this, an **isotropically fair** neighborhood definition finds a set of points with a balanced or "fair" influence in all directions ("isotropic") of the reconstructed manifold.

One idea of implementing an isotropically fair neighborhood is to iteratively perform PCA on increasing point sets and determine the best suited ellipse (or ellipsoid in 3D) defining the neighborhood. This would also require a parameter defining the initial point set size, which has a big influence on the result. This approach is also limited because it assumes that the local neighborhood can be approximated well by an ellipse (or ellipsoid in 3D).

In this thesis, a multi-ring (or multi-layer) approach is used. This divides the process of finding an isotropically fair neighborhood into two phases: Finding a small immediate neighborhood for each point and then merging the immediate neighborhoods to arbitrarily large sets. This has the advantage that we do not make general assumptions about the shape of the object and that we can also compute the first phase independently for each point. For every point p_i of the input point set, a zero ring (i.e., a set of immediate neighbors) $\mathbf{Z}_i$ is computed. The neighborhood is then defined recursively. The r-ring neighborhood of point i, $\mathbf{nh}_r(p_i)$, is the union of all zero-rings of all points in $\mathbf{nh}_{r-1}(p_i)$ (the current neighborhood). Then the final neighborhood $\mathbf{NH}_i$ of point p_i is composed of all rings up to a specified number $r_{\max}$. A possible implementation of this scheme is defined in Algorithm 4.1.

The first loop, which constructs the set $\mathbf{Z}$, is just used for memoization, i.e., creating a look-up table. This way, the function *zeroRing* is called exactly once per input point. Otherwise it would be called each time line 11 is executed. The choice of the *zeroRing* is important for the properties of the resulting neighborhoods. The N-Ring construction itself does not guarantee a isotropically fair neighborhood. The *zeroRing* itself must be isotropically fair such that the whole N-Ring neighborhood has this property. In sections 4.1.1 and 4.1.2, two candidates for the *zeroRing* function are discussed.

Algorithm 4.1 produces a neighborhood for each input point. For certain tasks in surface reconstruction (e.g., regression), a weighting function between a point and its neighborhood is needed. The following weighting scheme of points p_j in the (N-Ring) neighborhood of point p_i is proposed:

$$\phi_{p_i}(p_j) = \frac{\overline{w_j}}{r_i(p_j)} \qquad \overline{w_j} = \frac{1}{|\mathbf{Z}_j|} \sum_{p_k \in \mathbf{Z}_j} \|p_j - p_k\| \tag{4.1}$$

Let $\overline{w_j}$ be the average distance between point p_j and the neighbors in its zero-ring $\mathbf{Z}_j$. The quantity $\overline{w_j}$ is computed for every point in the point cloud. This weight is

Algorithm 4.1: Accumulation of the N-Ring neighborhood

Input: Set $\mathbf{P} = \{p_i\}$ of n points in R^3, function *zeroRing*, number of rings $r_{\max}$
Output: Set $\mathbf{NH} = \{\mathbf{NH}_i\}$ of neighbors for every point i

```
1  Z = {} ;
2  NH = {} ;
3  for p_i in P do
4  |  Z_i = zeroRing(p_i);
5  end
6  for p_i in P do
7  |  NH_i = Z_i ;
8  |  for 1 to r_max do
9  |  |  NH_tmp = NH_i ;
10 |  |  for p_j in NH_tmp do
11 |  |  |  NH_i = NH_i ∪ Z_j ;
12 |  |  end
13 |  end
14 end
15 return NH ;
```

additionally divided by the index of the ring, $r_i(p_j)$ (p_j was found in the r-th ring in the N-Ring neighborhood of point p_i).

The rationale behind this formula is to favor points that have long connections in their zero-ring neighborhood. The weighting of the zero-ring lengths (w_j) is normalized over all elements of the current neighborhood. Furthermore, points in higher-degree (bigger values for $r_i(p_j)$) rings have less influence on the weighting. This favors near points and reduces the smoothing effect. This weighting scheme is included in the spherical regression presented in Section 4.3.

In our empirical tests, we only need up until $r = 3$ rings for all tested data sets and noise levels. The total 3-ring neighborhoods consist of about 30 to 50 elements. Our adaptive regression method (see Section 4.2) never chooses more than the points of the 3-ring neighborhood. Depending on the noise level, less rings suffice. Therefore, it is not a necessity to compute all rings for all input points. During the adaptive iterations presented in Section 4.2, the neighborhood is constructed and extended on the fly.

In the following two sections (4.1.2 and 4.1.1), two possible candidates for the *zeroRing* function are presented.

4.1.1 Local Umbrella Zero Rings

Isotropically fair neighborhoods are designed to account for non-uniform sampling of a surface. The multi layer approach, discussed in Section 4.1, requires a zero neighborhood to iteratively find the set of points that make up the neighborhood. An important

requirement of the zero neighborhood is that it has an even coverage of the space surrounding a point p_i.

An umbrella of a point p_i in a triangulation is defined as the set of points that are directly connected to p_i with an edge. An umbrella is well suited for a zero neighborhood since it captures – by definition – the geometry around a point on the surface. In our case we do not have a triangulation to extract the umbrellas from, but we can approximate them. Umbrella approximations are also used in some surface reconstruction methods (e.g., [AGJ02] and [KA08]). See Section 2.2.1 for more information about umbrellas and their usage in surface reconstruction.

The local umbrella approximation method used in this thesis is motivated by the findings in [OMW13]. The overall goal of this technique is to find for each point p_i an umbrella with low curvature (i.e., the angle between the facets is maximized) and with minimal edge lengths. These properties are motivated by the *Gestalt Principles*. Figure 4.3 illustrates the basic outline of the approximation.

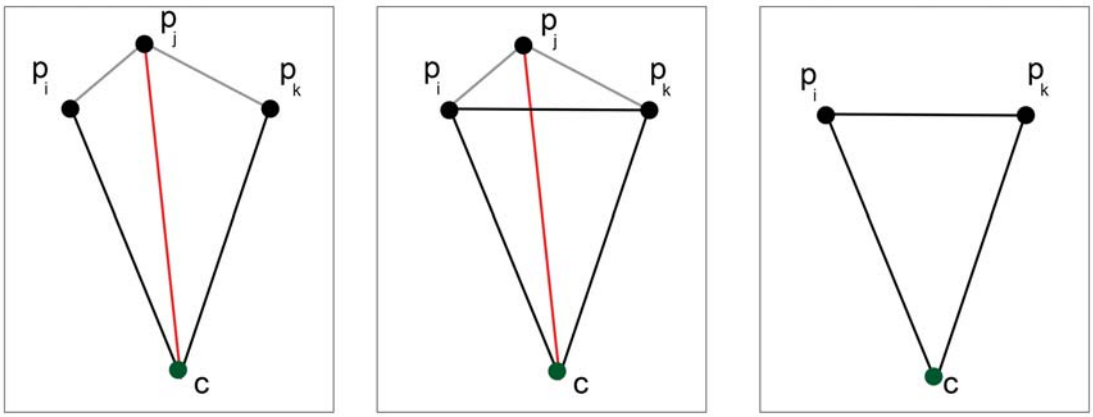


Figure 4.2: Optimization of the umbrella edges: the green point c is the center of the umbrella. The current point p_j is evaluated: the maximum distance to its neighbors and to the center point is calculated. Since the direct connection of the neighboring points p_i, p_k is shorter than the maximum of the incident edges of p_j, it gets removed (together with its edges).

Our approach first finds a set of candidate neighbor points $C(p_i)$ with a fixed isotropically biased neighborhood (k-nearest neighbors, with $k = 16$). Then the regression plane H_i of $C(p_i)$ is computed with principal component analysis. Every point of $C(p_i)$ is then projected onto H_i. The projected points p'_j are sorted according to the angle of the vectors from p'_i to p'_j. The initial triangulation $T(p_i)$ is obtained by connecting all outer points p'_j with the center point p'_i and connecting the outer points in a clockwise fashion (see right sub-images of Figure 4.3).

$T(p_i)$ is then improved in a greedy manner (see Figure 4.2): For each point p_j the

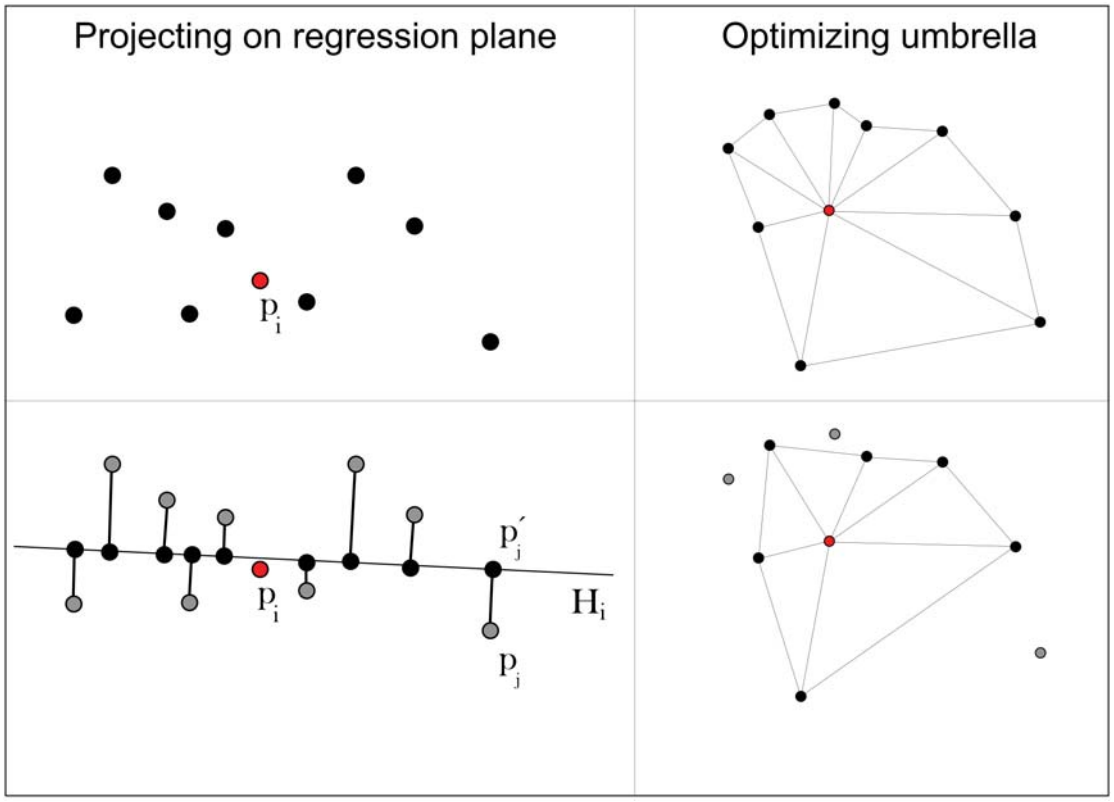


Figure 4.3: Left sub-images: points of neighborhood are projected onto common regression plane (line in 2D). Right sub-images: Points are sorted with respect to their angle to the center point p_i. Points get removed to reduce the total edge length until criterion reached (e.g., number of points reached a threshold).

distances to the adjacent neighbors in the umbrella (p_i, p_k in Figure 4.2) and the center point is calculated. If the maximum of those lengths is greater than the direct connection of the neighbors, p_j is removed from the umbrella along with its edges. This is done until a threshold is reached (in our experiments 6 was an appropriate value) or if no further flips are possible. Note, that in every step the overall edge length (sum of all lengths of edges) is reduced.

The umbrella neighborhood for a point p_i is then defined to be the set of points that constitute the local umbrella $T(p_i)$. Tests carried out with this neighborhood showed inferior results than with the neighborhood described in Section 4.1.2. Particularly in regions with high curvature, the performance of the umbrella neighborhood is worse. An explanation for this behavior is that during the umbrella construction process, the points are projected onto the regression plane of the point set. This introduces artifacts in neighborhoods with high curvature.

4.1.2 BSP-Neighborhood Zero-Rings

The incremental process of finding an *isotropically fair* neighborhood (see Section 4.1) requires a method to create the immediate neighborhood of a point. The BSP-neighborhood proved to be better suited than the aforementioned Umbrella Zero-Rings (Section 4.1.1). The Binary-Space-Partitioning or BSP-neighborhood was proposed by [GG07]. In their paper it is used for defining neighborhood relations for the propagation of normal orientations. See Chapter 5 for a description of this problem.

The characteristics of the BSP-neighborhood are illustrated in Figure 4.4. Every point p_h in the neighborhood of p_i defines a plane (a line in 2D) that goes through p_h and has a normal-vector of $p_i - p_h$. This plane creates two half-spaces. On the inside (side of p_i) and the outside. The definition of the BSP neighborhood demands that every point in the BSP-neighborhood is in the inside space of every other point in the BSP-neighborhood.

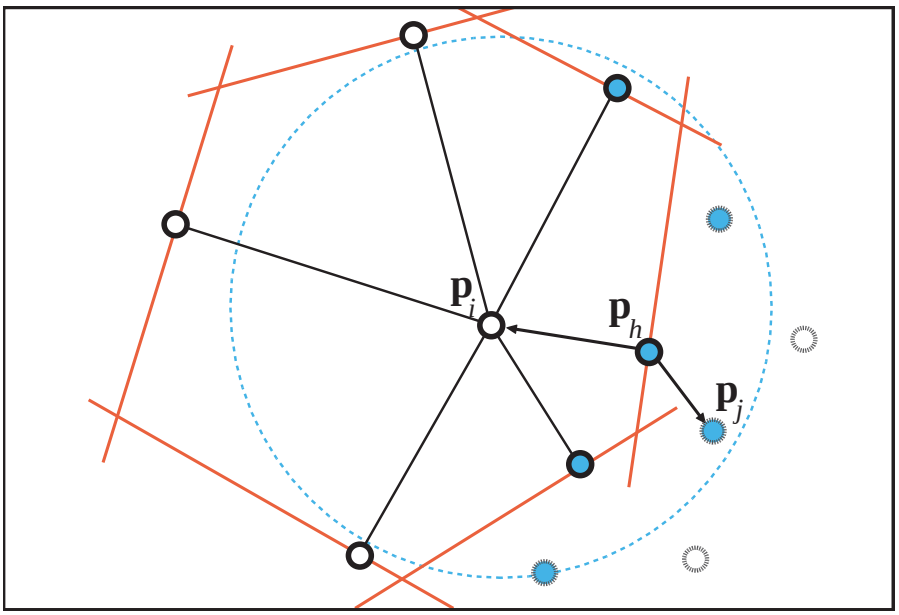


Figure 4.4: BSP-Neighborhood: Each point in the neighborhood restricts the available space for new neighbors. The blue circle illustrates an isotropically biased neighborhood. Image from [GG07].

Equation 4.2 gives a formal definition of this neighborhood.

$$\mathbf{BSP}(\mathbf{p_i}) = \{p_h \in \mathbf{P} \setminus \{p_i\} : (p_i - p_h)^T (p_j - p_h) > 0, \forall p_j \in \mathbf{P}\} \qquad p_i \in \mathbf{P} \tag{4.2}$$

To construct the neighborhood, a set of candidate points is created. In our experiments, we used k nearest neighbors with $k = 30$. For every candidate neighbor p_j of p_i, it must be checked if there exists a p_h that is "in front" of p_j, i.e., p_j is in the outside half-space of p_h. (see Figure 4.4)

Informally, this neighborhood defines a set of points that are distributed fairly over all directions. An "unfair" neighborhood would include all points of a cluster that is approximately in the same direction relatively to the point p_i. With the BSP-neighborhood, we only include one point "in each direction". This property is desired by the generation of the N-Ring neighborhood in Section 4.1. In this thesis we use the BSP-neighborhood as the *zeroRing* function of Algorithm 4.1.

4.2 Local Surface Fitting

In Section 4.1, the term *isotropically fair* neighborhood was introduced to define a neighborhood that does not only rely on the Euclidean distance but uses also information about the structure and connectivity of the points. Such neighborhoods have the benefit to better approximate points clouds with non-uniform sampling. The Section 4.1 explains how to build and increase an isotropically fair neighborhood. This section describes a method to select the best-fitting size for a neighborhood during local surface reconstruction. Our technique uses isotropically fair neighborhoods, but can also be applied with any other neighborhood definition.

Other state-of-the-art reconstruction and resampling methods (e.g., [HDD+92], [GKS01] (both k-NN), [GG07], [HLZ+09] (common radius of neighborhood)) work with fixed-size neighborhoods; i.e., a fixed number of neighbor points k or points in a ball neighborhood with a fixed radius which are considered for the reconstruction at a certain point on the manifold. This is a global property which is equal for all input points. It can be automatically calculated or has to be set by the user. For models with different noise levels and sample densities, the user has to guess the optimal parameter with a trial-and-error approach. Another shortcoming of global fixed values is that they treat every input point equally. The assumption that the noise level is uniform across all data points is not always true. Especially range images of 3D scanners have different variance in different image regions, depth ranges and for different surface angles ([ABCO+01], [Bol10]).

The iterative scheme presented in this section does not require the user to set a parameter for the input point clouds and adapts to the noise level for every point of the input set. We assume Gaussian noise of the input data, the variance (σ^2) being the noise level. The general idea of this adaptive method is to find a fitting model for a small neighborhood. If a heuristic detects that the noise is too high to find a good fit for the particular neighborhood, the neighborhood is increased and the process is repeated. The output of the algorithm is a set of **noise spheres**. The center of one sphere determines the position of the resampled point, and the radius represents the estimated noise level for the current sample. The general algorithm for the calculation of the noise spheres is outlined in Algorithm 4.2.

For each point p_i, the N-Ring neighborhood Q is calculated (see Section 4.1). It is initialized with the zero ring of p_i (in our case the BSP neighborhood). Note that the point itself (p_i) is also included in the neighborhood Q and used for the regression. The crucial part of the iterative scheme is the neighborhood criterion: It increases the

Algorithm 4.2: CPU implementation of noise sphere calculation

Input: Set $\mathbf{P} = \{p_i\}$ of n points in R^3
Output: Set $\mathbf{N} = \{\mathbf{c_i}, r_i^2\}$ of noise spheres

```
1  for p_i in P do
2      Q = N-Ring_Nhood(p_i, P) ∪ p_i ;
3      while nhood_criterion(S, Q) == false do
4          Q.extend() ;
5          s = regressionSphere(Q) ;
6      end
7      c_i = proj(p_i, s) ;
8      D = distances(Q, s) ;
9      r_i = max(D) ;
10     N = N ∪ {c_i, r_i^2} ;
11 end
12 return S ;
```

neighborhood until the regression sphere is a suitable fit for the local surface. The criterion compares the extent of the neighborhood with the size of the regression sphere s. Equation 4.3 gives an exact description thereof. In Line 4, the neighborhood Q is extended, i.e., more points are added according to the neighborhood definition of Section 4.1. In our implementation, we added the next 3 points to the N-Ring neighborhood in each iteration to reduce the number of invocations of the sphere regression function. S denotes the regression sphere of the neighborhood Q. Its calculation is explained in Section 4.3.

Function *proj* computes the projection of point p_i onto the sphere s. The projected point c_i defines the center of the noise sphere. The radius of the noise sphere represents the noise level (variance at the specific point). It is approximated with the maximum distance of the neighboring points to the regression sphere s (line 9 in Algorithm 4.2). Figure 4.5 shows an illustration of the projection, and Figure 4.7 visually explains the iterative process.

If the regression sphere S (together with the neighborhood Q) does not meet the neighborhood criterion, the neighborhood is increased and the process is repeated. Increasing Q is done on the fly according to Algorithm 4.1: A new ring is only calculated when needed. The reasoning behind this approach is that a good approximation of the underlying manifold has to be found. If the neighborhood is too small, the points of the neighborhood are not a good representation of the local surface. Figures 4.7(a),(b) show regression spheres that are not a good fit to the local surface. After extension of the neighborhood, the regression sphere (or circle in 2D) is a reasonable model of the local surface.

We fixed the maximum neighborhood size to the 3-ring neighborhood: On the one hand, using bigger neighborhoods results in over-smoothing, and the resulting projected points and their normals are not a good approximation of the local surface. On the

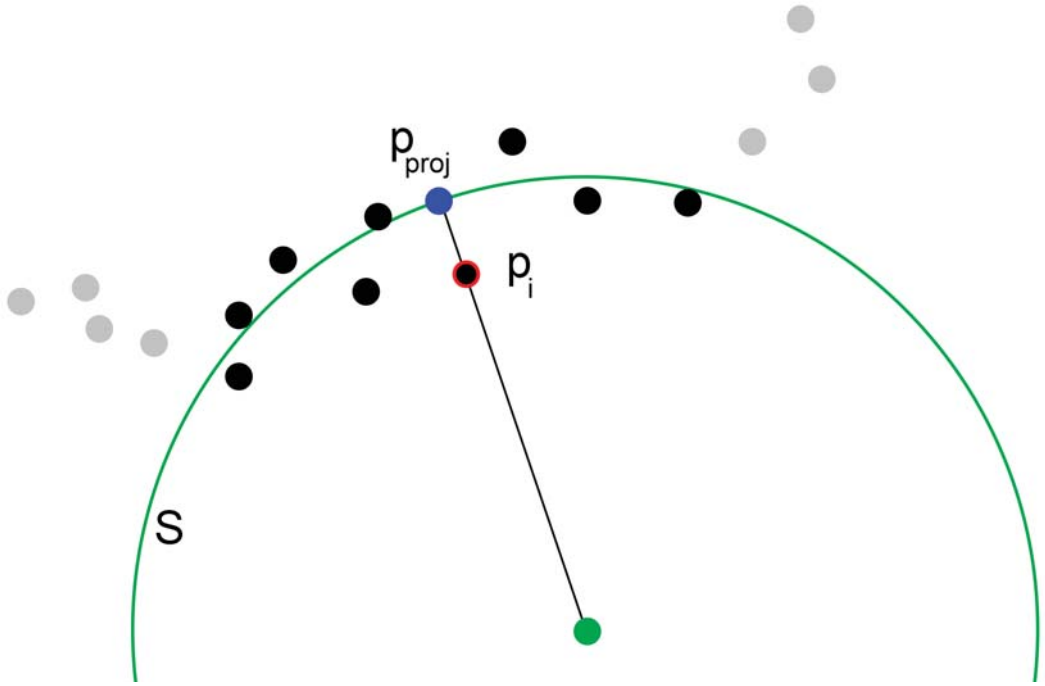


Figure 4.5: Projection of the point p_i onto its regression sphere S produces the new sample p_{proj}.

other hand, the run time is also less predictable when arbitrarily large neighborhoods are allowed. If no noise sphere was found after reaching the maximum neighborhood, the heuristic could not find a good surface approximation at this specific position. Instead of finding a bad fitting point, it is simply discarded; it is not projected, and no normal vector is generated for this particular sample. This occurs particularly for noisy and densely sampled point clouds. In our tests with different point clouds, discarding non-fitting points did not introduce holes or other artifacts.

This method succeeds in finding noise spheres for almost all elements of noise-free point clouds. (None discarded for simple models like the ellipsoid with small features in Figure 7.6 and about 1% for complex models like the dragon in Figure 7.9).

Table 4.1 shows the average and maximum number of iterations used for a model and different noise levels. Points that are discarded do not count into these values. On average only between 1 and 2 iterations are needed: As long as the local neighborhood is approximately planar, a small number of neighbors is sufficient. Some points (in noisy and high-curvature areas) require up to 20 and more iterations.

The key element of this method is to find a well-suited criterion that decides if the regression sphere is a good fit to the local neighborhood (with respect to the underlying manifold) or if the neighborhood has to be enlarged. This is an ill-posed problem since

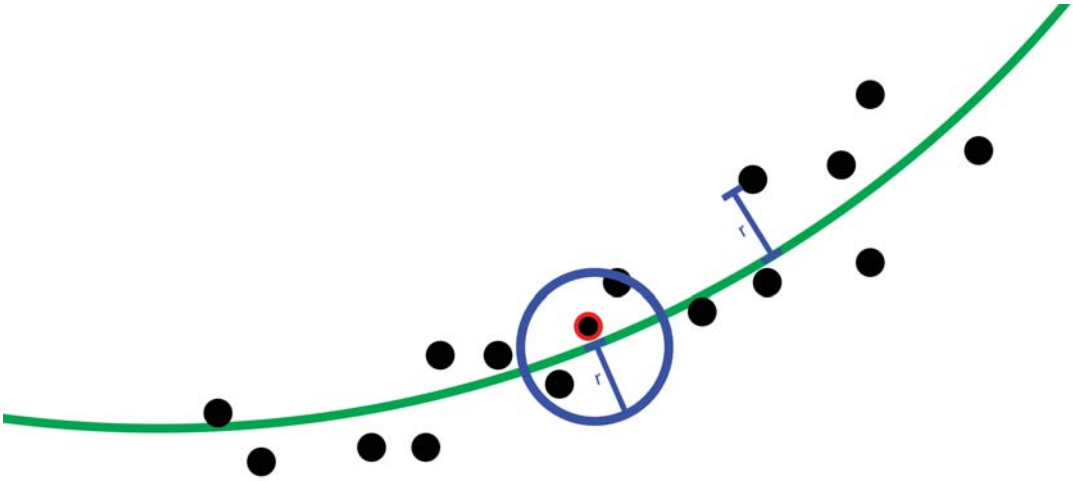


Figure 4.6: 2D equivalent of noise sphere: radius of the circle is the maximum distance of the neighborhood to the green regression sphere.

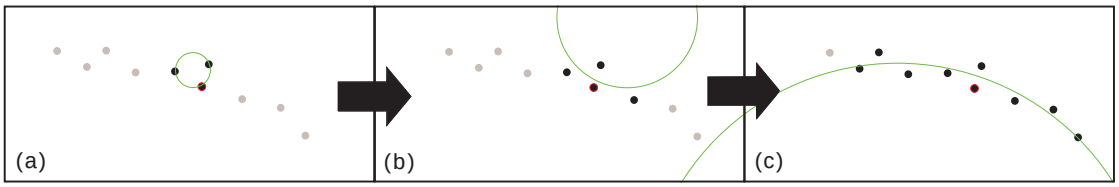


Figure 4.7: Adaptive regression iteration for a circle in 2D

both the true manifold and the noise level are unknown. An exact solution to this problem is therefore not possible. Instead, a heuristic is needed that fulfills the following requirements:

- uses local information only
- no assumptions about noise level
- works well for most cases

A simple, yet effective heuristic is the following criterion (equation 4.3):

$$\text{isFitting} = \frac{r_i}{ex_i} > c \tag{4.3}$$

	# iterations		# neighbors	
	max	avg	max	avg
dragon (0.0008)	18	1.32	60	5.62
dragon (0.002)	22	1.89	73	7.51
dragon (0.0035)	26	2.63	89	9.92

Table 4.1: Number of iterations of the adaptive regression and the number of used neighbors.

r_i is the radius of the regression sphere S_i, ex_i denotes the extent of the neighborhood Q: $ex_i = \|p_i - Q_{\text{last}}\|$. Specifically, it is the distance between the input point p_i and the furthest point in the current neighborhood Q. The constant c was empirically determined to be 2.1. Intuitively, this criterion ensures that the regression sphere is at least twice as big as the point set. The radius of the regression sphere is also an approximation of the local surface curvature [GG07]. The radii of the approximation spheres are visualized in the left sub-image of Figure 4.8: Red denotes high curvature and small radius, green is less curvature.

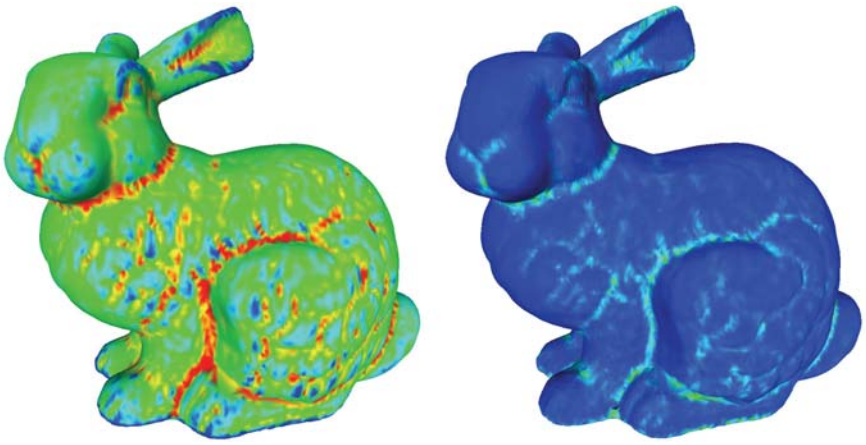


Figure 4.8: Left sub-image: Visualized radii of the regression spheres as an approximation of the curvature as suggested by [GG07]. Right sub-image: Radii of the noise spheres of the samples are encoded in color: blue means small radius (small estimated noise), lighter areas show bigger noise spheres.

Figure 4.7 illustrates this process in 2D: The regression circle in the left sub-image is not representative for the local manifold. After some iterations the circle approximates the underlying geometry well.

If a suitable regression sphere S is found, the noise level of the current point p_i is approximated. The distances of all samples in the neighborhood $\mathbf{NH}_i$ to the regression sphere are computed and the maximum is taken as an approximation of the noise level.

The reasoning behind this approach is the following: We assume that the manifold can be approximated with parts of surfaces of spheres. The spheres are estimated with spherical regression (Section 4.3) for every point (and its neighborhood) in the input point-set. Since the assumption implies that all points in a specific neighborhood have the regression sphere as the underlying model, the difference between the actual point coordinates and the ideal position on the surface of the regression sphere is the deviation from the true value.

Note that this is just a heuristic for estimating the noise level (variance σ^2). The assumption of a local spherical surface introduces (as every other model) model errors. Depending on the actual geometry approximated by the point cloud and the noise extent, this method under- or overestimates the actual noise level (which was created artificially for the tests). Tests were conducted with three different models (ellipsoid with small features, Armadillo and Bunny): Figure 4.9 shows the relation between the noise extent approximated with the noise spheres and the actual noise level. Especially with low noise extent, the estimation is too high. The approximation of the noise extent gets better with higher noise levels.

An explanation for this observation is the number of points used for the estimation of the noise level: In areas with low noise level, the neighborhoods are small therefore the estimation relies on a small number of points. In areas with high noise level, many points are used to determine the noise extent, therefore the estimate is more reliable.

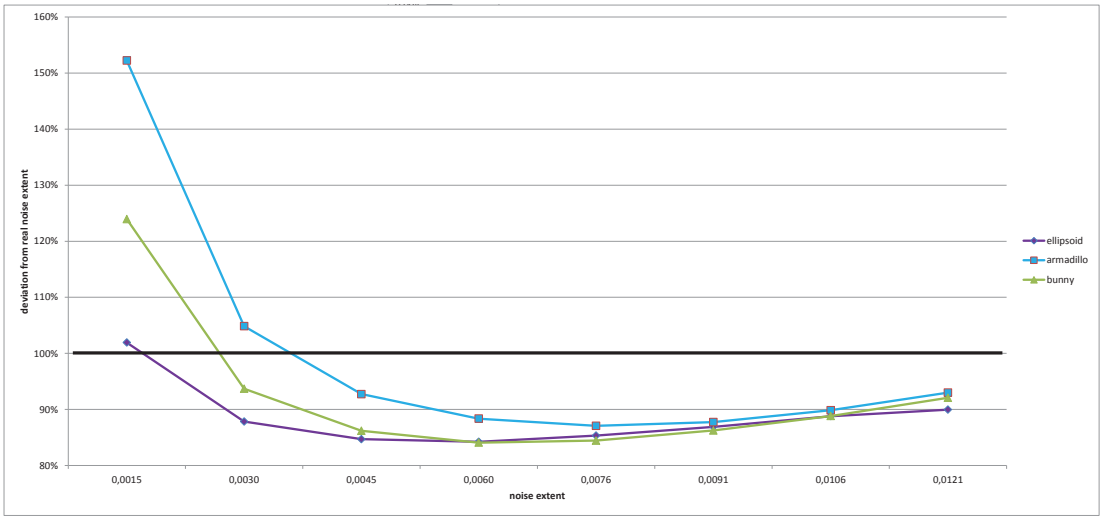


Figure 4.9: Illustration of the relation $\frac{\text{estimated noise extent}}{\text{true noise extent}}$. At low noise levels the noise extent is over-estimated, with higher noise it gets under-estimated. Tested models: ellipsoid with small features (see also Figure 7.6), Armadillo and Bunny (Figure 7.3)

4.3 Spherical Regression

The whole resampling process presented in Section 4.2 relies on a good and fast local approximation of the surface surrounding a specific point. In this thesis, spherical regression is used as the local approximation method. The reason why spherical fitting was chosen is a trade-off between model complexity – and therefore also computational effort – and accuracy. A more complex model may be a better approximation for some local patches, but it is also more prone to over-fitting and more costly. Empirically, the sphere turned out to be well suited for planar regions with dense sampling and for high curvature patches with lower sampling density. More complex models may produce better results in some corner cases but are significantly slower. The computational complexity increases with the cube of the order of the model. See [Pra87]: the effort for fitting m points to a surface of order n is $(m+n)n^2$.

In this section, a circle fit method is extended to obtain a spherical fitting technique. A fast parallel implementation of this fitting is also presented here.

Algebraic sphere fitting offers a good closed-form spherical approximation while also being able to handle planar cases [Pra87]. Section 2.1.3 gives an overview of this topic and explains the difference between algebraic fitting and geometric fitting. It also explains the regression with algebraic spheres used in [GG07], which uses the fitting technique by Pratt [Pra87].

Two years after APSS ([GG07]), Chernov et al. published an algebraic circular regression technique, which is called "Hyper-Fit" [AsC09]. This estimator has less bias (difference between the actual value and the expected value of the regression) than previous circle regression methods: In [AsC09], various circle regression methods – both geometric and algebraic – are investigated concerning their statistical properties. In that paper, the term *essential bias* is used for the bias of the leading order of $O(\sigma^2)$, which is independent of the number of used samples (here, σ^2 denotes the variance of the noise of the input points). They argue that this square bias term has significant impact on the mean squared error and is therefore *essential* for the quality of the estimator.

All methods researched in [AsC09] (including geometric fits) have non-zero essential bias. The *Hyper-Fit* method of [AsC09] is specifically designed to have zero essential bias. In this thesis, the circle regression *Hyper-Fit* is extended to 3D and used for approximating the local surface in the adaptive regression method.

In [AsC09], the "Hyper-Fit" circle regression is defined with the constraint matrix in Equation 4.4:

$$\mathbf{N_H} = \begin{bmatrix} 8\overline{w} & 4\overline{x} & 4\overline{y} & 2 \\ 4\overline{x} & 1 & 0 & 0 \\ 4\overline{y} & 0 & 1 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix}, \tag{4.4}$$

where $\overline{x}$ is the sample mean $\frac{1}{n}\sum_{i=0}^{n} x_i$ and $w = x^2 + y^2$. Using this matrix $\mathbf{N_H}$ in the constraint in equation 2.13 results in equation 4.5:

$$8A^2\overline{w} + 8AB\overline{x} + 8AC\overline{y} + B^2 + C^2 + 4AD = 1 \tag{4.5}$$

Extending it to 3D results in equation 4.6:

$$8A^2\overline{w} + 8AB\overline{x} + 8AC\overline{y} + 8AD\overline{z} + B^2 + C^2 + D^2 + 4AE = 1 \tag{4.6}$$

This extension is based on similar relations between circular and spherical fits in [Pra87]. The constraint 4.6 together with the objective function in Equation 2.9 translates to the minimization problem in Equation 4.7 in the "natural" parametrization of spheres:

$$F_h = \frac{\sum_{i=1}^{n}[(x_i - c_x)^2 + (y_i - c_y)^2 + (z_i - c_z)^2 - r^2]^2}{\sum_{i=1}^{n}[2(x_i - c_x)^2 + 2(y_i - c_y)^2 + 2(z_i - c_z)^2 - r^2]}. \tag{4.7}$$

The "Hyper-Fit" is a drop-in replacement for any other algebraic sphere-fitting method, with superior statistical properties. The constraint 4.6 can be written in matrix form in Equation 4.8:

$$\mathbf{N} = \begin{bmatrix} 8\overline{w} & 4\overline{x} & 4\overline{y} & 4\overline{z} & 2 \\ 4\overline{x} & 1 & 0 & 0 & 0 \\ 4\overline{y} & 0 & 1 & 0 & 0 \\ 4\overline{z} & 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.8}$$

This matrix $\mathbf{N}$ can be used in equation 2.16 to solve the regression as an eigenvalue problem. In addition to the standard spherical regression, the process can be extended with a weighting scheme. This is useful if the input points for the regression should not have the same influence on the resulting sphere. Section 4.1 (specifically equation 4.1) proposes a weighting scheme used in the noise sphere generation.

In our tests, the inclusion of the weights into the minimization process improves the quality of the resampling and the normal computation. Equation 4.9 describes the calculation of the weighted matrix of moments $\mathbf{M_w}$ with the weight matrix W, which is a diagonal matrix containing the weights w_{ii} for each point p_i. $\mathbf{Z}$ is defined in equation 2.11 and contains the positional information of the points.

$$\mathbf{M_w} = \mathbf{Z^T W Z} \tag{4.9}$$

Profiling of the sphere regression process indicated that about 85% of the computation is spent calculating the eigenvectors. In the general formulation (equation 2.16), an asymmetric eigenvalue problem has to be solved. It is possible to transform the equation into a symmetric eigenvalue problem and transform the computed eigenvectors back into the original problem domain. This reformulation was already published for a similar setting by [STN88].

The constraint matrix $\mathbf{N}$ (see definition 4.8) is symmetric by definition, therefore $\mathbf{N^{-1}}$ is also symmetric. $\mathbf{M} = \mathbf{Z^T Z}$ is also symmetric and positive definite [AsC09]. Thus the

Cholesky-Decomposition is defined for $\mathbf{M}$: $\mathbf{R^T R} = \mathbf{M}$. $\mathbf{R}$ is a right upper triangular matrix. Substituting $\mathbf{M}$ with $\mathbf{R^T R}$ in equation 2.16 leads to:

$$(\mathbf{N^{-1} R^T R})\mathbf{P} = d\mathbf{P} \tag{4.10}$$

Multiplying the equation with R from the left produces a different eigenvalue-problem:

$$(\mathbf{R N^{-1} R^T})(\mathbf{RP}) = d(\mathbf{RP}) \tag{4.11}$$

The matrix $\mathbf{RN^{-1}R^T}$ is symmetric by definition, and $\mathbf{RP}$ are the corresponding eigenvectors. The eigenvalues are the same as for $\mathbf{N^{-1}M}$. To obtain the eigenvectors for the original problem, the resulting eigenvectors have to be multiplied with the inverse of $\mathbf{R}$.

$$\mathbf{P} = \mathbf{R^{-1}}(\mathbf{RP}) \tag{4.12}$$

In general, the eigenvectors have to be normalized to unit length, since $\mathbf{R^{-1}}$ is not an orthogonal matrix and therefore not length preserving. In the case of algebraic spheres, this does not matter: The algebraic representation is invariant under a scalar multiplication:

$$\mathbf{P}u = (Au, Bu, Cu, Du, Eu) \Rightarrow c_x = \frac{Bu}{-2Au} = \frac{B}{-2A} \tag{4.13}$$

$$r^2 = \frac{(Bu)^2 + (Cu)^2 + (Du)^2 - 4AuEu}{4(Au)^2} = \frac{u^2[B^2 + C^2 + D^2 - 4AE]}{u^2 4A^2} \tag{4.14}$$

This reformulation leads to a more efficient calculation of the eigenvectors and the corresponding sphere (see Algorithm 4.3).

Algorithm 4.3: Fast Algebraic Sphere Regression

Input: Symmetric inverse constraint Matrix $\mathbf{N}_{\text{inv}}$ and symmetric and positive definite matrix of moments $\mathbf{M}$

Output: Sphere defined by center $\mathbf{C}$ and squared radius r^2

1 $\mathbf{R}$ = CholeskyDecomposition($\mathbf{M}$) ;
2 $(\mathbf{V}, \mathbf{D})$ = SymmetricEVD($\mathbf{R N_{\text{inv}} R^T}$) ;
3 $i = \arg\min\{D > 0\}$;
4 $\mathbf{P} = \mathbf{R^{-1} V_i}$;
5 $\mathbf{C} = \frac{1}{-2\mathbf{P_1}} \mathbf{P_{2:4}}$;
6 $r^2 = \frac{1}{4\mathbf{P_1}^2} \mathbf{P_{2:4}^T P_{2:4}} - 4\mathbf{P_1 P_5}$;
7 **return** $(\mathbf{C}, r^2)$;

Notes: $\mathbf{P_1}$ is the first coordinate of vector $\mathbf{P}$, $\mathbf{P_{2:4}}$ is a 3-dimensional sub-vector of $\mathbf{P}$ of coordinates $2, 3, 4$. "SymmetricEVD" returns the eigenvalues $\mathbf{D}$ and the eigenvectors

in the columns of $\mathbf{V}$. If the eigenvectors are sorted corresponding to the ascending eigenvalues, $i = 2$ for the regression method discussed in Section 4.3. The inversion of $\mathbf{R}$ can be performed with back-substitution because it is a triangular matrix.

In cases where the input points of the spherical regression are approximately on a plane or if only few points (between 4 and 8) are used, the matrix $\mathbf{M}$ can be badly conditioned. In this case the matrix is not positive definite and the solution is not defined. This can be checked during computation of the Cholesky decomposition. If the matrix is not positive definite, the number of points has to be increased or a different regression model has to be applied (e.g., planar regression with principal component analysis). In our case, the neighborhood – i.e., the set of points for the spherical regression – is increased if $\mathbf{M}$ behaves badly. Section 4.2 describes the process of selecting further points for the neighborhood.

A different way of computing the spherical regression is also suggested in [AsC09]. It uses the singular value decomposition, which is numerically more stable and covers the case of a badly conditioned matrix, too. This implementation was not used here because it is significantly slower, as it requires an additional SVD step for the data matrix $\mathbf{Z}$ of equation 2.11.

The transformation of an asymmetric to a symmetric eigenproblem of Algorithm 4.3 requires more steps but is faster. A CPU implementation, including the Cholesky decomposition and the back-transformation of the eigenvectors, is about twice as fast as the general asymmetric eigenproblem. The GPU-implementation for the general QR-algorithm (asymmetric eigenvalue solver) in double precision is not feasible on current hardware. One reason for this could be that the asymmetric algorithm needs more registers for the computation. If there are no more free registers, the local variables are spilled into global memory, which significantly slows down the computation.

The symmetric problem can be implemented on the GPU. Section 6.1.1 gives an outline thereof. Figure 7.14 compares the three different implementations for solving the $5x5$ eigenvector and algebraic sphere computation.

CHAPTER 5

Globally Consistent Normal Orientation

Chapter 4 deals with a local problem: How can the the local surface of a point be approximated? This estimated surface has a position and a direction – the normal vector. Apart from their spatial relation, the single local surfaces are independent from each other – there is no defined inside and outside consistent with all surface patches. In order to obtain a globally oriented model, the orientation of the normal vectors has to be determined. Global mesh-reconstruction methods (e.g., Poisson reconstruction [KBH06],[KH13], RBF [CGGS13]) often require a set of oriented normals in addition to the unorganized point cloud.

This chapter first describes the problem of normal orientation in the domain of mesh reconstruction. Two classes of algorithms for solving this problem are identified, and a brief overview of the state-of-the-art of one approach is given in Section 5.1. Section 5.2 improves state-of-the-art techniques regarding performance and quality (robustness) of the consistent normal orientation. The neighborhood information generated during the local reconstruction process (Section 4) is reused in this stage to achieve better results.

5.1 Problem definition

Various techniques exist to estimate the normal direction of the surface, based on a surface model (e.g., [HDD+92], [MN03], [GG07], and Section 4 of this thesis). A local surface S_l (based on a certain model) is fitted to the spatial positions of the input points (or a subset thereof). Regardless of the surface model, the normal n of each point on the modeled surface S_l can be computed. This vector n may be explicitly defined for the local surface S_l, but its orientation on the global surface is not defined – at least not by S_l. In a local context, there is no inside or outside of the object. The local neighborhood represents a topological disc, whereas a closed mesh has a different topology. The union

of all local surfaces has to be set into a global context and the outside, and the inside have to be defined.

Figure 5.1 illustrates the state after the local normal computations. The normal direction is computed for each point, but the global orientation is not yet defined. Therefore, each point has a normal direction but no specific orientation. The problem definition is illustrated with an example in Figure 5.2: The normal vectors of the model are calculated locally, which results in a non-consistent global orientation (left sub-image of Figure 5.2). After the orientation pass, the normals point from the inside to the outside of the model.

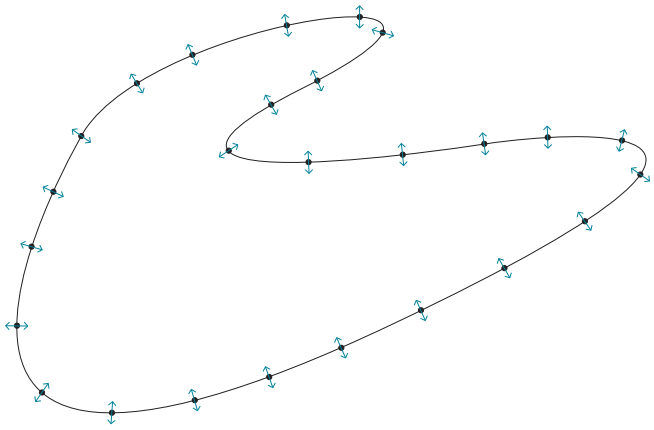


Figure 5.1: Normal computation estimates the normal directions, not the global orientations.

This chapter deals with the problem of finding a consistent global normal orientation for the whole point cloud. In general, there are two types of algorithms for this problem domain: Volumetric methods and so-called "surficial" propagation methods [KG09].

Volumetric methods classify the space around the sampled surface into an inside and an outside region. If a correct labeling was established, the normals are flipped in a way such that they point from the inside of the object to the outside. A major drawback of this class of methods is that they need closed surfaces in general (see [KG09]). The raw point cloud of scanned objects rarely represents a closed surface. Therefore, these methods are not well suited for our purposes. Another disadvantage of volumetric methods is that they require a spatial data structure (e.g., octree), which introduces more complexity.

Surficial methods build a global connectivity structure of the points (with their respective normals) and propagate the normal orientation of one or many seed points across the whole surface. A comprehensive discussion and a state-of-the-art report is available in [KG09].

The fundamental principle of surficial orientation propagation methods was introduced by Hoppe et al [HDD+92]. The basis for most surficial methods is the Euclidean minimum spanning tree (EMST). The EMST can be constructed by forming a complete

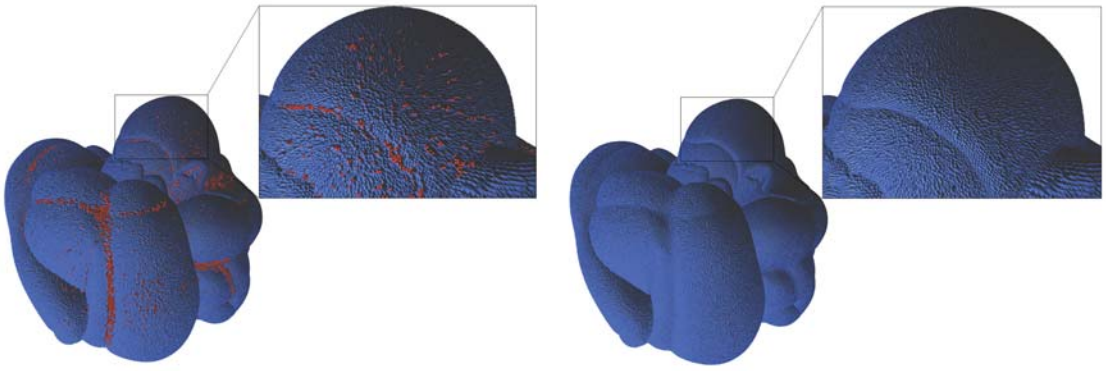


Figure 5.2: After the local normal computation the normal vectors have no consistent global orientation. The left image shows the normal vector of the *omotondo* model. Blue vectors have the correct orientation, red vectors point into the opposite direction. The right image shows the same model after flipping the normal vectors to obtain a global consistent orientation.

graph of the point set (all points are connected to all other points) and assigning the Euclidean distance between the connecting points to each edge. Then the EMST is the minimum spanning tree (MST) of this weighted complete graph. This is not an efficient calculation of the EMST since it relies on $O(n^2)$ (where n is the number of points) distance computations. A more efficient way is to generate the Delaunay triangulation of the edge set and calculate the MST of the edge set of the triangulation. This works because the Delaunay triangulation is a superset of the EMST [HDD+92].

Hoppe et al. then use the so-called "Riemannian Graph" G_R as the underlying data structure. This is the union of the edge set of the EMST and auxiliary edges based on the k-nearest neighbors of the points. The EMST ensures that the graph itself is connected, otherwise a global propagation would be interrupted. The auxiliary edges add connections to the graph, which increases the number of possibilities to propagate the global orientation through the graph. As an improvement, the APSS-framework [GG07] uses the BSP-neighborhood (see Section 4.1.2) instead of the k-nearest neighbors for enriching G_R.

Finally, the minimum spanning tree (MST) of G_R is calculated, and the orientation is propagated in depth-first search (DFS) order. The critical parts of methods using the surficial approach are the weight function for the graph, and the decision function that decides if a normal vector has to be flipped.

In [HDD+92], a zero- (or small) curvature assumption is used: The difference between adjacent normals in G_R is assumed to be minimal (near zero). The weight function 5.1 determines the weight of the edges of G_R:

$$w(e_{ij}) = 1 - |(n_i * n_j)|. \tag{5.1}$$

This weighting penalizes edges that connect normals with different normal directions. Orthogonal normals get the weight 1, and 0 is assigned to parallel normals. The absolute value is taken so that the orientation of the vectors does not influence the weight.

The DFS of the MST of G_R then is the path through the vertices of G_R with minimum variation of the (adjacent) normals. This works well for very densely sampled or low-curvature regions, since adjacent normals will always have little deviations from their neighbors. Adjacent normals in high-curvature parts of a point cloud tend to differ more and violate the zero-curvature assumption. (see Figure 5.1)

To overcome this shortcoming, [XWH+03] proposed a different weighting scheme. They make a constant curvature assumption: The curvature between adjacent normals should be approximately constant. Instead of directly comparing the two normals, one normal is reflected on the bisector of the edge e_{ij} connecting the two points. Then the weight function 5.1 of [HDD+92] is used to calculate the weight of the edge. The reflection is defined by equation 5.2:

$$P(n_i, j) = n_i - 2(e'_{ij} * n_i)e'_{ij}, \tag{5.2}$$

where e'_{ij} is the normalized vector from point p_j to p_i.

This reflection is visualized in Figure 5.3. If a constant curvature is assumed, both points (with their normals) lie on a circle (in 2D). Formula 5.2 is then a measure for the deviation from the constant curvature assumption.

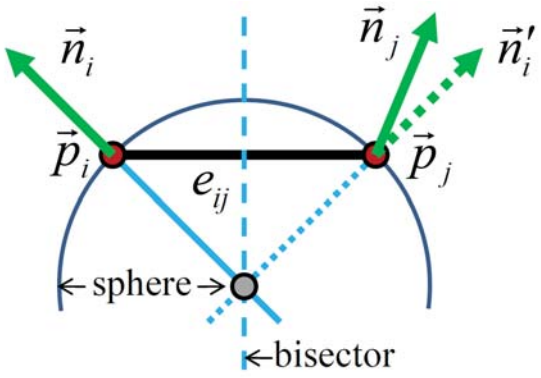


Figure 5.3: Constant curvature assumption: The normal of the neighboring point is reflected by the bisector of the edge between the points and then compared to the other normal. (Illustration from [KG09])

This more elaborated weight function performs better on high-curvature surfaces than the method of [HDD+92].

The actual propagation is bootstrapped at one or more seed points. A possible choice is the point of minimum z-value. From this seed point, the orientation of the

corresponding normal is propagated along the DFS of G_R. At each step in the graph, a decision has to be made on each edge if the next normal has to be flipped. The decision function is related to the weight function: In [HDD+92], a normal is flipped if the condition $n_i * n_j < 0$ is true. The method proposed by [XWH+03] uses the condition $P(n_i) * n_j < 0$ ($P()$ defined in Equation 5.2).

5.2 Improvements using local neighborhoods

In this section, it is shown how the neighborhood structures described in 4.1 can be used to speed up the generation of the *Riemannian Graph* G_R. An additional local operation is used to make the whole process more robust.

The problem of consistent normal orientation across the whole mesh is global per definition. Therefore, the data structure G_R is computed in a global context. To ensure connectedness of all nodes of the graph, the EMST is the base of G_R. This MST can be computed by first constructing the 3-dimensional Delaunay Triangulation of the point set and adding all edges with the corresponding edge lengths to a weighted graph. The EMST then equals the MST of the resulting weighted graph. The expected asymptotic runtime of the 3D Delaunay Triangulation is near linear, although it has a worst-case complexity of $O(n^2)$ [ABL03] (where n is the number of points).

Instead of constructing the EMST and enriching the graph with k-nearest neighbors [HDD+92] or the BSP-Neighborhood [GG07], we propose to use N-Ring neighborhoods (Section 4.1) instead. G_R is replaced by G_{lR}, the "Local Riemannian Graph". It is the union of the adaptive neighborhoods that were used to reconstruct the local surfaces and the corresponding noise spheres (see Section 4.2). This approximation is not guaranteed to be connected. Since the connectivity is determined based on local criteria, it cannot be guaranteed (for all possible input point clouds) that the union of all local neighbors results in a globally connected graph. Analysis of G_{lR} shows that most ($> 95\%$) points are connected and form a big connected component. The other connected components usually contain < 5 vertices. A simple heuristic to construct a globally connected graph G_{lR} is described in Algorithm 5.1:

The graph **G** is formed by generating the union of all points and all edges of the local neighborhoods, such that it does not contain any duplicates. Then the connected components of **G** are extracted and the biggest connected component *main_comp* identified. All other components are connected to *main_comp* by finding the $k = 3$ nearest neigbors in *main_comp*. The edges connecting *main_comp* with the point in the other component is added to **G**.

The complexity of obtaining the connected components of a graph is $O(|V| + |E|)$, for constructing the Kd-Tree of a set of 3D points it is $O(n \log(n))$. Finding the nearest neighbors in a Kd-Tree has the complexity of $O(\log(n))$. Since the number of connected components is small (< 200 for many point sets) and contain only few (< 5) vertices, finding the edges to make the graph connected is a fast operation. The number k of nearest neighbors was set to 3 in our experiments.

Algorithm 5.1: Heuristic to generate a connected graph from the N-Ring neighborhoods of the point set. **Nhoods** contains the set of neighbors that were used to approximate the local surface and calculate the noise spheres in Section 4.2.

Input: Set $\mathbf{V}$ of all points, set $\mathbf{Nhoods} = \{nh_i\}$ contains neighborhoods $\forall p \in \mathbf{V}$, $\|\mathbf{V}\| = n$

Output: Set $\mathbf{E}$ of edges of G_{lR}, a connected graph with n vertices

```
1  E = {} ;
2  for nh_i in Nhoods do
3  |   E = E ∪ edges(nh_i) ;
4  end
5  G = Graph(V, E) ;
6  C = connected_components(G) ;
7  main_comp = max_size{C} ;
8  T = KdTree(main_comp) ;
9  for comp in {C \ main_comp} do
10 |   for i = 1 : k do
11 |   |   p = comp[i];
12 |   |   nn = kNearestNeighbors(p, T, 1) ;
13 |   |   E = E ∪ edges(p, nn) ;
14 |   end
15 end
16 return E ;
```

In some cases, the heuristic performs suboptimal. If the variance of the noise level is very high, the assumption that there exists one main connected component (containing most points) is false. In this case both the quality of the resulting graph, as well as the actual runtime of the computation, suffer. This was not the case for the tested models. One solution to this problem would be to detect the case of too many connected components (percentage of vertices in the biggest connected component) and to fall back to an EMST. Another failure case for Algorithm 5.1 occurs when there are other disconnected components between the main component and other components. This can happen for models with long and thin structures. In those cases, G_{lR} offers only a mediocre approximation to the *Riemannian Graph*.

It is rather hard to evaluate the methods for global orientation propagation. The whole process is quite unstable: One bad decision (wrong flip) can influence many other normals and degrade the overall quality of the normal orientation. Tiny differences of the input point cloud can cause big differences of the result. The reason for this behavior is that all propagation methods are approximations. The underlying problem of finding the best orientation for all points can be reduced to the MAX-CUT problem, which is NP-complete ([HDD+92]).

Increasing the complexity of the propagation has a decreasing return of investment. [KG09] propose a rather complex method (which is also based on [HDD+92]). The

runtime is approximately 10 times higher than the methods of [HDD+92] or [XWH+03]. Still, it fails for some point sets and is only marginally better than [XWH+03]. For some man-made objects, however, it outperforms the other methods regarding quality.

We propose an additional preprocessing step for the orientation propagation technique to increase the robustness of the solution. It is a local pass before the MST calculation and uses the N-Ring neighborhood structures of Section 4.1. The idea behind this method is to analyze the local consensus of the orientations. The N-Ring neighborhood of a point is only valid for itself, therefore an edge $e_{ij} \in nh_i$ is a valid relation between points p_i, p_j for point p_i. The same edge may not be present in neighborhood nh_j of point p_j. In a first step, the bijective neighborhood $\mathbf{bi_{nh}}$ (equation 5.3) is generated to filter out edges that are not valid for both points.

$$\mathbf{bi_{nh}} = \{e_{ij} : p_j \in nh_i \wedge p_i \in nh_j\} \quad \forall e_{ij} \in \mathbf{NH} \tag{5.3}$$

Algorithm 5.2 describes the iterative local process to find a local consensus and assess its confidence.

Algorithm 5.2: Find the consensus orientation of each point and calculate a confidence value.

Input: Set $\mathbf{N}$ of all normals, set $\mathbf{bi_{nh}}$ of all bijective neighborhoods
Output: Array $\mathbf{C}$ of confidence values for each point, updated set $\mathbf{N}$

```
1  C = {} ;
2  flipped = |N| ;
3  while flipped ≠ 0 do
4  |  flipped = 0 ;
5  |  for i = 1to|N| do
6  |  |  same_orientation = 0 ;
7  |  |  for nb_ind ∈ bi_nh_i do
8  |  |  |  same_orientation += N(i) * N(nb_ind) > 0 ;
9  |  |  end
10 |  |  agreement = same_orientation / |bi_nh_i| ;
11 |  |  if agreement < 0.5 then
12 |  |  |  flip(N(i)) ;
13 |  |  |  agreement = 1 − agreement ;
14 |  |  |  flipped + + ;
15 |  |  end
16 |  |  C(i) = agreement ;
17 |  end
18 end
19 return N, C ;
```

In each iteration, the algorithm checks for every point whether the corresponding normal points in the same direction as at least more than half of the bijective neighbors.

In case it differs from the "consensus" orientation, it is flipped. This is done as long as normal flips are executed.

In the rare case that a point p_i has an empty bijective neighborhood bi_{nh_i}, the agreement is set to 0. This fits with the rationale that only points with a high local agreement are well-suited for propagating the orientation. Points that lack bijective neighbors have no reliable partners to exchange the orientation.

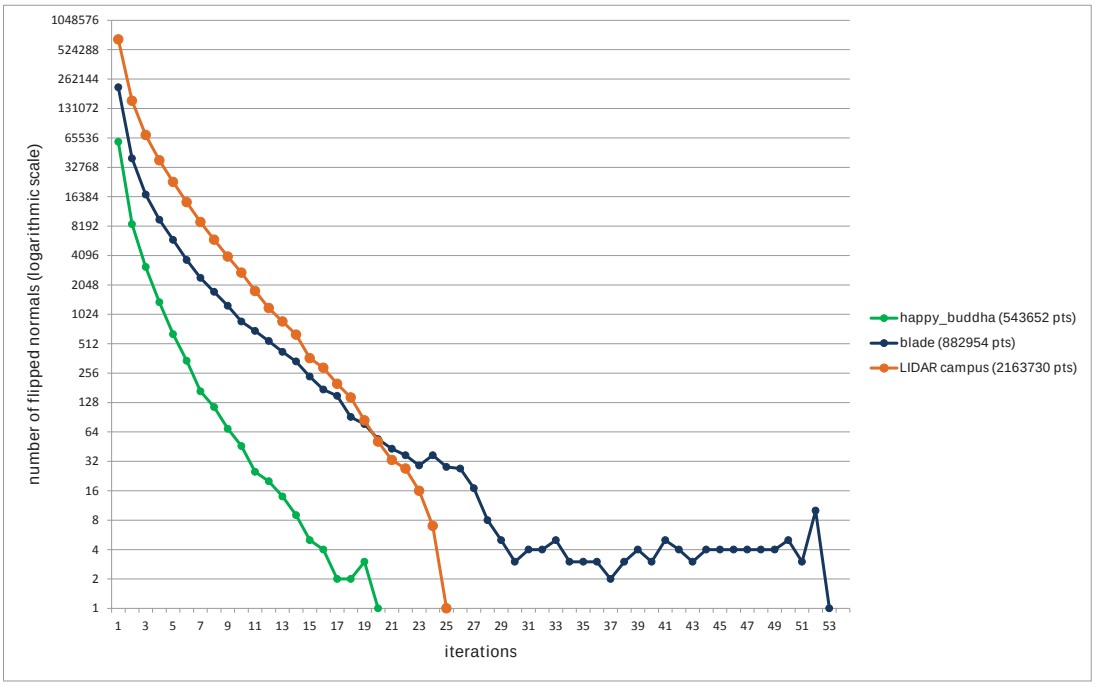


Figure 5.4: The flipping Algorithm 5.2 converges quickly for different models. (y-axis is scaled logarithmically) Models: LIDAR and blade point cloud, Buddha statue (Figure 7.4)

The local agreement and flipping algorithm converges fast for every tested point cloud. Figure 5.4 gives an overview of the convergence for three different models. The graph also shows that the method does not converge monotonically. It decreases approximately logarithmically for all tested models. In some cases, the function alternates between lower values before finally converging to 0. In our implementation, we limited the number of iterations to 40, since the solution does not change much after this iteration count. Figure 5.5 visualizes the result of the local agreement iterations: Blue encodes the maximum agreement ($= 1.0$) whereas red denotes 0.5. The pattern emerges from the dynamic behavior of the algorithm: Areas of non-optimal agreements (< 1.0) shrink and move around the surface until they converge.

The local agreement, which is computed with Algorithm 5.2 and stored in array $\mathbf{C}$,

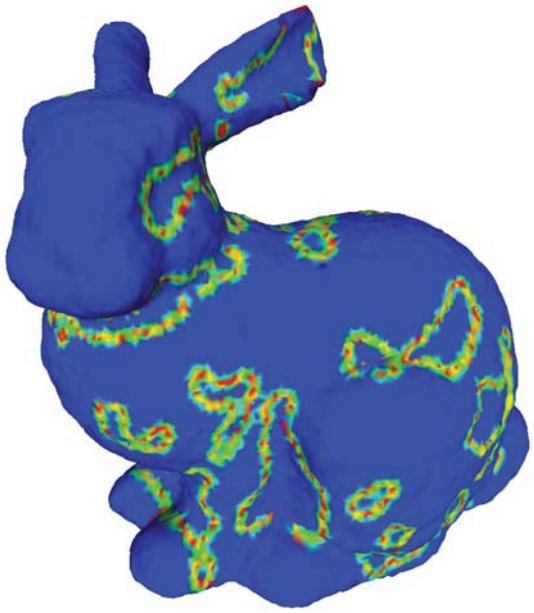


Figure 5.5: Visualized local agreement after convergence: Blue = 1.0, red = 0.5.

is then integrated into the edge weight of G_{lR} (equation 5.4):

$$\text{weight}(e_{ij}) = w_{ij} * (1 - \min[\mathbf{C}(i), \mathbf{C}(j)]), \tag{5.4}$$

where w_{ij} denotes the weight assigned by the metric – either w_h by [HDD+92] or w_x by [XWH+03] (see definitions in Section 5.1). This penalizes edges where at least one node has a bad local agreement. The expected behavior of this approach is that edges connecting "unreliable" vertices are in the leaf nodes of the DFS-tree during the orientation propagation.

In summary, our global normal-orientation algorithm performs the following steps:

- Generate G_{lR} with Algorithm 5.1
- Execute Algorithm 5.2 to obtain the agreement values
- Compute weights for the edges of G_{lR} with equation 5.4
- Propagate orientations like in [XWH+03] or [HDD+92]

The MST and DFS calculations, as well as the actual orientation propagation, are the same as in other state-of-the-art propagation techniques (e.g., [XWH+03]). A single-threaded CPU implementation of this method is about 2-3 times faster than the standard "Riemannian Graph" approach. Note that this can be further accelerated by computing the bijective neighborhood structure and the local agreement in parallel. The DFS has

linear time complexity ($O(|V|+|E|)$), whereas the MST can be computed in linearithmic time $O(|E|log|V|)$ using Prim's algorithm [CSRL01]. Section 7.2 presents results of our tests and compares the performance between our local method and the global approach.

CHAPTER 6

Implementation Details

This chapter describes the implementation of the methods presented in Chapter 4 and 5. The implementation is divided into the following parts:

- Calculation of the noise spheres
- Optional sphere filtering pass
- Consistent Normal Orientation
- Experimental Triangulation

6.1 Regression Sphere Calculation

Two different implementations were created: One is executed purely on the CPU, the second makes use of the GPU for performance-critical sections (see Section 6.1.1). On the CPU, a straightforward implementation of Algorithm 4.2 is possible. It works independently on each input point. Therefore, the order of execution is not determined and the method is trivially parallelizable. The outer loop can be split up between arbitrarily many threads on a CPU without any synchronization mechanisms.

The following optimizations and reformulations accelerate the implementation:

The implementation of spherical regression can be divided into two parts: Building the matrices and calculating the Eigenproblem (see Section 4.3). The central element is the matrix of moments, $\mathbf{M} = \mathbf{Z}^\mathbf{T}\mathbf{Z}$ (see equation 2.11). Instead of first constructing matrix $\mathbf{Z}$, it is faster and uses less memory to compute $\mathbf{M}$ directly (see equation 6.1), particularly for increasing number of points n.

$$\mathbf{M} = \begin{bmatrix} \overline{ww} & \overline{wx} & \overline{wy} & \overline{wz} & \overline{w} \\ \overline{xw} & \overline{xx} & \overline{xy} & \overline{xz} & \overline{x} \\ \overline{yw} & \overline{yx} & \overline{yy} & \overline{yz} & \overline{y} \\ \overline{zw} & \overline{zx} & \overline{zy} & \overline{zz} & \overline{z} \\ \overline{w} & \overline{x} & \overline{y} & \overline{z} & \overline{1} \end{bmatrix}, \tag{6.1}$$

where $\overline{xw} = \frac{1}{n}\sum_{i=1}^{n} xw$. In order to avoid unnecessary matrix inversions, N^{-1} can be constructed directly (equation 6.2).

$$\mathbf{N^{-1}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0.5 \\ 0 & 1 & 0 & 0 & -2\overline{x} \\ 0 & 0 & 1 & 0 & -2\overline{y} \\ 0 & 0 & 0 & 1 & -2\overline{z} \\ 0.5 & -2\overline{x} & -2\overline{y} & -2\overline{z} & -2\overline{w} + 4\overline{x}^2 + 4\overline{y}^2 + 4\overline{z}^2 \end{bmatrix} \tag{6.2}$$

6.1.1 GPU-Implementation of Spherical Regression

The most expensive part of the local surface fitting (see Section 4.2) is the spherical regression. This is due to the fact that the regression function is invoked several times for each input point (see Algorithm 4.2). A solution to this problem is transferring these computations to the GPU. Since the problem of finding a regression sphere to each neighborhood is local and independent, it can be executed in parallel.

A general asymmetric eigenvalue computation for $5x5$ matrices in double precision is infeasible on currently available graphics hardware (e.g., Geforce GTX770). The reason is that the QR algorithm for solving the asymmetric case requires too much memory for local variables. One reason for this is that the asymmetric problem has complex eigenvalues in general, and therefore the computations are done using complex numbers. In contrast to the CPU, the GPU cores do not have a powerful cache infrastructure, which would alleviate the problem. Instead, all local variables that do not fit into the registers are spilled into global memory. Therefore, the whole computation is slowed down by several orders of magnitude. The symmetric eigenproblem, however, can be computed on the GPU. Algorithm 4.3 was implemented in CUDA, using subroutines from the *JAMA Matrix Package* (we ported it from Java to CUDA).

To reduce the traffic between the CPU and the GPU, all matrices of all points (and their respective neighborhoods) are collected for each step of the iterative fitting process (Algorithm 4.2). Then, all matrices are sent to the GPU, on which Algorithm 4.3 is executed. The resulting spheres are then analyzed on the CPU. In the next iteration, only those matrices are collected that require additional regression steps.

This approach succeeds in calculating 2.8 million spheres per second on a Geforce GTX770. That is about 20 times faster than the same algorithm on a single CPU core. Figure 7.14 gives a comparison of the calculation of the 5x5 eigenvalue problem.

The present implementations (both on the GPU and CPU) have further potential for optimization. Ideas for a more efficient noise-sphere calculation can be found in Section 8.2.

6.2 Sphere Filtering

For the current approach in Section 4, a dense but noisy sampling may contain redundancy. Consider a sample p_j that is within the noise sphere of a sample p_i. Since p_j is within the (estimated) noise extent of p_i, it does not add additional information for the underlying surface. An additional pass reduces the number of noise spheres with Algorithm 6.1. One aim of this stage is to reduce the number of samples used in subsequent stages of the processing pipeline (e.g., consistent normal orientation) in order to improve the runtime performance. This method should also reduce those noise spheres that overestimate the actual noise level. Note that this is an experimental algorithm that was not used for producing the results in this thesis.

Algorithm 6.1: Noise sphere filtering

Input: Set $\mathbf{S}$ of noise spheres
Output: Set $\mathbf{S_f}$ of filtered noise spheres

```
1 I = intersections(S) ;
2 G_I = build_graph(I) ;
3 sort_edges(G_I) ;
4 while has_edges(G_I) do
5 |   s = biggest_sphere(G_I) ;
6 |   remove_edges(s, G_I) ;
7 end
8 S_f = nodes(G_I) ;
9 return S_f ;
```

Computing the intersections between all spheres (Line 1) has a run time of $O(n \log^3 n + k)$, where n is the number of spheres and k is the number of actual intersections. It is implemented by first determining the intersections of the bounding boxes of the spheres using CGAL's [CGA] box intersection algorithm. The actual sphere intersections are then calculated by checking the bounding box intersections. The nodes of the undirected intersection graph G_I represent the noise spheres, edges e_{ij} denote that sphere s_i intersects with s_j.

Algorithm 6.1 removes the biggest noise sphere until there are no more intersections between the spheres. The noise is often over-estimated, which produces too large spheres (which intersect many neighboring spheres). Removing the biggest spheres first reduces the number of deleted spheres. Figure 6.1 shows the effect of noise-sphere filtering. The input of about 240k points (and their noise spheres) is filtered to 60k in 4 seconds (example with much overlap).

We did not include the sphere filtering process in our tests for better comparability with other methods. The filtering itself is only reasonable with very dense sampling and high noise extent. On many test data sets, this method removes only a small number of samples and has no positive effect on the reconstruction of the surface. One problem is that the radius of the noise sphere is not a very accurate estimate of the actual noise

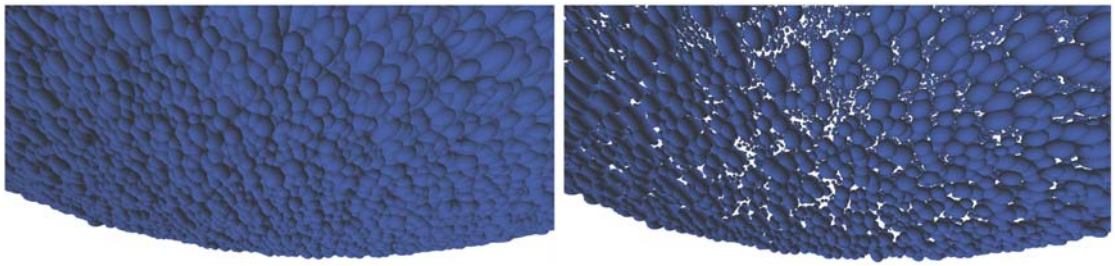


Figure 6.1: Left: dense overlapping noise spheres. Right: Filtering reduces overlap and reduces the overall number of noise spheres.

level – see Figure 4.9. Therefore, the removal procedure falsely deletes many samples. A better approach incorporates all samples and explicitly models the noise. For example, in CLOP [PMA+14] points are not removed but clustered for the mixture of Gaussians approach. This reduces the number of points and therefore also the computational effort of the subsequent steps, while including the information of all samples. Another disadvantage of this noise-sphere filtering is that it is a global approach that cannot be executed in parallel easily.

6.3 Consistent Normal Orientation

The orientation of the normals is fixed with the method described in Chapter 5. The bottleneck of this step is the calculation of the MST of the Riemannian graph. Currently, a single-threaded implementation of Prim's algorithm is used. It is preferred over Kruskal's MST algorithm because of the better asymptotic complexity: Prim has $O(|E| \log |V|)$, whereas Kruskal has $O(|E| \log |E|)$ (where $|V|, |E|$ are the number of vertices and edges, respectively). The constant factor of Prim's algorithm is larger and it is therefore slower for sparse graphs. In our case, the input graph is rather dense (> 6 edges per vertex), so Prim's algorithm has better performance: Kruskal's MST algorithm takes about twice the time for a Riemannian graph with 500k nodes.

6.4 Local Triangulation

For fast preview, an approximated local triangulation is implemented. It works similarly to the method described in [KA08] (see also Section 2.2.1): The nearest neighbors of each point p_i are projected onto a regression plane and sorted according to their angle with respect to p_i (which is in the center of the neighbors). Then a heuristic is used to choose those vertices from the projected neighbors s.t. the length of the maximum edge is minimized. This generates an umbrella – a triangle fan centered at p_i – for each point. The left sub-image of Figure 6.2 shows the result of computing and displaying the local umbrellas. It is sufficient for a fast preview of the triangulation but has clear

disadvantages: It is not water-tight and has several triangles that intersect other triangles and many non-manifold triangles.

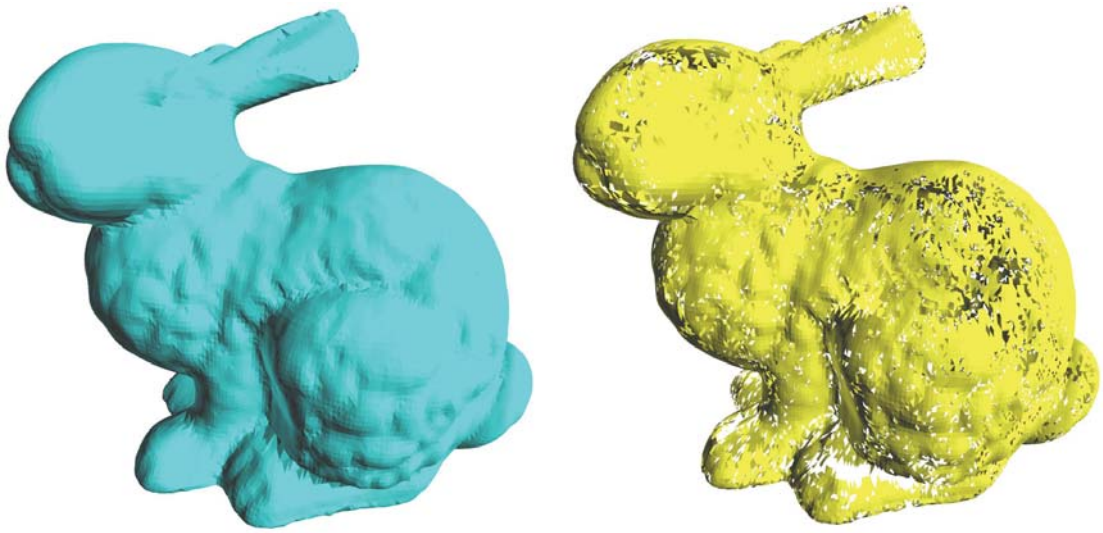


Figure 6.2: Left: Local umbrellas (94408 triangles) Right: 54146 consensus triangles

An additional global pass determines the *consensus triangles*. For each triangle, the number of references is counted: If a triangle is referenced by all its 3 points, it is a consensus triangle. The right sub-image of Figure 6.2 shows the consensus triangles of the bunny. The umbrella generation is done on the GPU and is very fast. For example, computing 3.5 million triangles (umbrella triangles) and 500k consensus triangles takes 0.3 seconds. Under certain sampling conditions (see [OMW13]) the consensus triangles are sufficient to obtain a (near) water-tight triangulation of a point cloud. In practice these sampling requirements are too strict and a further processing of the incomplete triangulation (hole filling) has to be done.

CHAPTER 7

Results

In this chapter the results of the methods of Chapters 4 and 5 are presented. We show that using isotropically fair neighborhoods can be beneficial for the reconstruction of the local surface and therefore for the resampling and normal computation. Our presented method reduces the noise of the input point cloud and generates a set of oriented normal vectors for the underlying surface. The latter is an essential requirement for surface reconstruction methods like [KH13]. Particularly non-uniform point clouds benefit from our technique, since it does not assume any noise level but adapts at each point, without requiring any user-defined parameters.

There is a large amount of techniques in the literature that aims at sharp edges and features. We specifically target non man-made objects. Our method adapts locally to reconstruct these sensing artifacts well and with some additional computational expense. Nevertheless, the algorithm works only on local data and can be executed in parallel. The additional data structures that are used for the noise-sphere calculation can then be reused for the normal orientation step. Our normal-orientation approach with local computations, which can be executed in parallel, results in a $2-3\times$ speedup while offering comparable quality and robustness.

We mainly compare our method to APSS [GG07], which also uses a spherical regression approach. Since the underlying approach is the same, this comparison is well suited to highlight the effect of isotropically fair neighborhoods and the adaptive regression. Two implementations for APSS are available: The reference implementation [GG07] requires an oriented point cloud (point cloud with oriented normals) and generates a triangular mesh using marching cubes. In this thesis, it is referred to as "reference APSS". The second implementation is provided by Meshlab [CCR08], whose APSS resampling method also requires normal vectors but does not calculate a triangulation (only the resampling). They can be computed by Meshlab, which uses an approach similar to [HDD$^+$92]. The combination of normal computation with Meshlab and its APSS implementation is referred to as "Meshlab APSS". Our method is also compared to "CLOP"

[PMA+14]. For this thesis, the author of [PMA+14] provided the results using their publicly available implementation.

7.1 Noise-Reduction Quality

In this section, the quality of the reconstruction of noisy point clouds is assessed. The Hausdorff distance is a metric that can be used to determine the distance between two surfaces. It is defined as:

$$\mathrm{d}_H(X,Y) = \max\left\{ \sup_{x\in X}\inf_{y\in Y} \mathrm{d}(x,y), \sup_{y\in Y}\inf_{x\in X} \mathrm{d}(x,y) \right\}, \tag{7.1}$$

where X, Y are non-empty subsets and d is a metric between two points. It is the maximum of the two one-sided Hausdorff-distances, which ensures that d_H is symmetric. This is required so that d_H is actually a metric. The Hausdorff distance can be approximated for point clouds and triangular meshes, which is described in [CRS96]: Both meshes are sampled, and the nearest distances between the samples on mesh A to samples on mesh B are stored. The maximum of those values is the one-sided Hausdorff distance $d(A,B)$. This algorithm is implemented in *Meshlab* [CCR08], which is used in this chapter for the quality evaluations.

For our assessments, the distance from the resampled points to the original mesh surface is calculated, and the one-sided Hausdorff distance and the root mean square error (RMS) is compared. This way, the error of each resampled point is evaluated. The RMS is defined as $RMS = \sqrt{\frac{\sum_{i=1}^{n}(p_i - p_t)^2}{n}}$ (where p_i are the resampled points and p_t the corresponding nearest points of the ground truth). The RMS has the advantage that it is less sensitive to outliers than the one-sided Hausdorff-distance.

7.1.1 Resampling

In this section, the quality of the resampling of Section 4 is evaluated. The input for a resampling method is a point cloud, the output is the set of denoised samples. The virtual scanner framework by Berger et al [BLN+13] was not used for testing the resampling quality since it mainly measures the quality of the output mesh. Our method does not output a triangular mesh but denoised points. The virtual scanner also requires a triangulated reference model, which was not available for all tested point clouds (e.g., range images, Horse).

In order to determine the resampling quality, the following procedure is used: The vertex data of a mesh is first extracted and perturbed with uniform isotropically distributed Gaussian noise. The resulting point cloud P_{input} is then fed into our algorithm, described in Section 4, to obtain a smoothed point cloud with oriented normal vectors P_{our}. The same point cloud P_{input} is processed by *Meshlab APSS* and *CLOP*. Afterwards, the distance between the resulting point clouds P_{our}, P_{APSS}, P_{CLOP} and the original mesh surface is computed. The root mean square error (RMS) and maximum error (H_{max})

	APSS [GG07]		CLOP [PMA+14]		our method	
	H_{max}	RMS	H_{max}	RMS	H_{max}	RMS
armadillo (0.00196)	0.8212	0.2111	1.2964	0.1875	1.5496	0.2221
armadillo (0.00328)	1.5703	0.3900	2.5830	0.2602	1.9191	0.3342
armadillo (0.0052)	2.6483	0.7587	3.117	0.358	2.1535	0.4826
blade (0.0004)	0.5529	0.0586	0.744	0.0879	1.3943	0.0792
blade (0.0016)	1.3305	0.2075	2.5478	0.185	1.5839	0.2050
blade (0.0031)	2.0449	0.5572	4.0604	0.251	1.7948	0.3453
buddha (0.0014)	3.7415	0.1964	3.846	0.2356	2.1344	0.2357
buddha (0.0034)	4.1494	0.5099	6.3544	0.44	2.7690	0.4666
buddha (0.0061)	4.9111	1.2711	7.166	0.759	3.6357	0.7323
bunny (0.0074)	7.0795	0.8101	7.0976	0.619	3.6464	0.7533
bunny (0.013)	9.3724	1.7300	13.372	0.853	6.3418	1.1436
bunny (0.0186)	11.6478	2.8843	12.813	1.201	8.4630	1.5101
dragon (0.0008)	0.5994	0.0974	0.597	0.168	1.5774	0.1049
dragon (0.002)	1.2132	0.2621	2.9002	0.19	1.3530	0.2323
dragon (0.0035)	2.3620	0.6054	5.1603	0.282	2.1741	0.3711
horse (0.0053)	3.2889	1.2722	3.662	0.368	3.3739	1.1759
horse (0.0074)	4.5887	1.7168	6.0798	0.746	4.8001	1.5816
horse (0.0105)	6.5765	2.3683	8.4267	1.021	6.2890	2.2368

Table 7.1: Comparison of different meshes with different noise levels (number in parentheses). The numbers in the table are scaled by a factor of 1000: H_{max} denotes the one sided Hausdorff distance (maximum) between the resampled points and the original vertices. The RMS is the root mean square error of all distances between the resampled points and the original surface. The points were resampled with *Meshlab APSS* [GG07] [CCR08] and with a reference implementation of *CLOP* [PMA+14]. The parameters of CLOP are manually adjusted for each model and noise level. Figure 7.1 visualizes the RMS of this table.

are used for evaluation. The values are normalized by the diagonal of the bounding box of the point cloud.

Table 7.1 gives an overview of different point clouds and compares our technique to *Meshlab APSS* and *CLOP*. Each model has different average feature sizes so the noise level is determined for each model individually. For each model 3 different noise levels (low, medium, high) are chosen. The actual noise level (which corresponds to the variance σ in equation 1.1) is determined by the formula $ext = f * \text{median}_{\|e\| \in \mathbf{T}}$, where e are the edges of the triangulation T of the model. The factor f is chosen visually to create the three noise levels for each model.

Figure 7.1 visually compares the RMS values of Table 7.1.

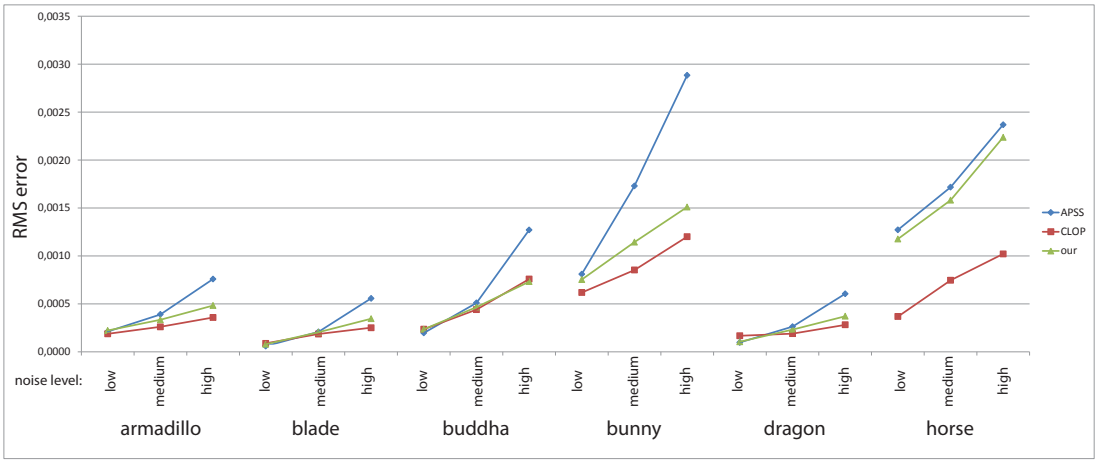


Figure 7.1: Visualization of the RMS of Table 7.1. Compares the RMS error of *Meshlab APSS* [GG07] and *CLOP* [PMA+14] with our method.

Unlike conventional kernel-based projective methods that require setting an individual filter kernel (and possibly several other parameters) for an optimal reconstruction result, our method automatically determines the optimal neighborhood for the primitive fitting, and is thus more flexible for point sets of varying density or noise. CLOP achieves a higher reconstruction quality than our method, but uses an L1 approach and also requires to provide an optimal bandwidth parameter for each model. Our method does not match the accuracy of complex L1 reconstruction methods like CLOP, but does not require any parameter. Furthermore, our method performs considerably better than APSS, which it is more related to.

In Figure 7.2, the distance between the resampled points and the original mesh is visualized for the *horse* model of Table 7.1. The first row is resampled with *Meshlab APSS*, the middle row with *CLOP* and the bottom row with our method. The noise levels increase from left to right. Parameter settings for the techniques and the noise levels are the same as in Table 7.1. Our method succeeds in adapting to the different sampling conditions and produces better results. In the examples with CLOP, the adaptation is done manually. The far better numerical values of CLOP result from the lower parts of the horse, where the sampling is very irregular. Therefore, the visual difference in Figure 7.2, which shows the head of the horse, is not as dramatic as the numbers of Table 7.1 would suggest.

7.1.2 Reconstruction

In this section, the quality of the reconstruction of a raw point cloud (without normals) is evaluated. This determines the quality of both the normal vector calculation, the normal orientation and the resampling (denoising of the point cloud). Since the refer-

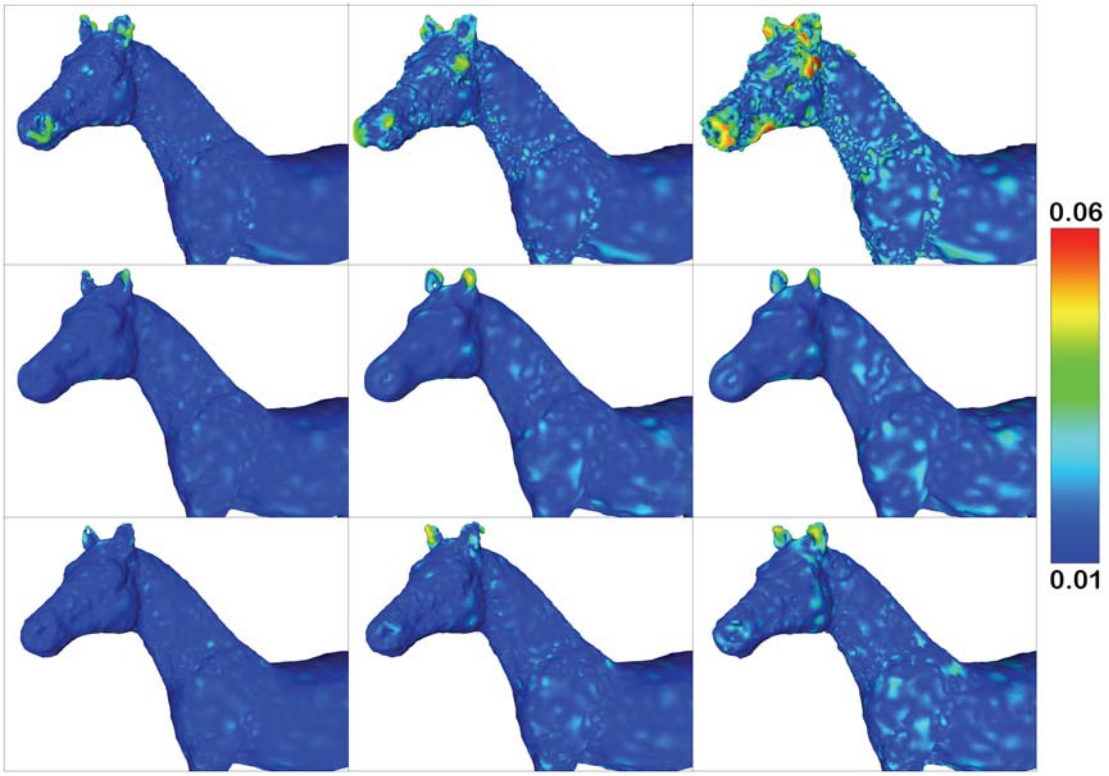


Figure 7.2: Visual example of the *Horse* of Table 7.1. The distance between the resampled points and the original surface is color-coded. The first row is resampled with *Meshlab APSS* [GG07], the second row with *CLOP* [PMA+14] and the bottom row with our method. The columns correspond to the noise levels of the input meshes as in Table 7.1.

ence implementations of APSS and CLOP do not support this whole step, we compare different reconstruction pipelines.

Besides resampling the vertices of the point set, our method reconstructs the normal vector for each point, which means that it recovers the orientation of the local surface. In order to compare the quality of both the normal vectors and the resampled points, Screened Poisson surface reconstruction [KH13] is used to obtain a mesh of the the point cloud. (In this thesis, the reference implementation published by the original authors is used.) This surface reconstruction method, like many others (see Section 2), relies on good normal vectors to recover the surface of a model. One advantage of this reconstruction method is that it is flexible: By setting the point weight to a high value (32 is used in this thesis), the resulting surface is nearly an interpolation of the data set. This is needed as it does not alter the result by additional smoothing. We compare this pipeline (resampling and normal calculation with our method + Screened Poisson) with

reference APSS [GG07]. Since Meshlab uses a different normal computation method, we cannot use this tool to compare the reconstruction performance of APSS. Unfortunately for our comparison, the reference implementation of APSS does not generate the normals from the point set; instead, the normals are obtained from the triangulation of the input mesh. That means, the APSS implementation does not compute the normals based on the unorganized point cloud but derives them from the connectivity information provided by the triangulated mesh. APSS then uses a marching cubes technique to find the surficial mesh of the implicit APSS representation of the resampled surface.

Figure 7.3 compares the two resulting meshes: The left sub-image is the input model (without / with noise), the middle sub-image is the APSS-reconstructed mesh, and the right sub-image is the result of our method.

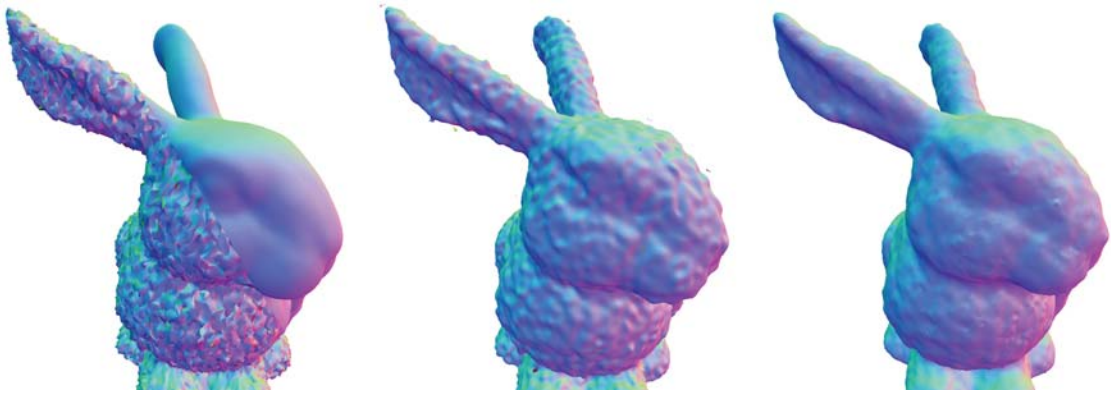


Figure 7.3: Comparison of the reconstruction quality: Left: input model *bunny* (without and with noise; noise level: 0.013, see Table 7.1). Middle: reconstruction with *reference APSS* [GG07] (reference implementation with marching cubes). Right: local reconstruction (positions+normals) with our method and Screened Poisson [KH13] to generate the mesh.

Our technique reduces the noise better than APSS, while still preserving the features of the input mesh. Figure 7.4 shows the result of the same reconstruction pipelines for the Buddha model. Note how the reconstructed head is much smoother with our method (right sub-image).

Figure 7.5 illustrates the difference in reconstruction quality between *reference APSS* and our method with Screened Poisson reconstruction (in interpolating mode). In this case, the input mesh is subjected to a big artificial Gaussian noise (0.0061, see Table 7.1). The left part is generated with *reference APSS* [GG07], the right part of the image shows the mesh created with our local reconstruction and screened Poisson reconstruction [KH13] (for the mesh generation). The colors encode the distance to the surface of the original model. It can be seen that our method adapts better to the high noise level than the APSS method. The overall RMS for this noise level is 39% less for our method than with *reference APSS*.

Figure 7.4: Left: input model *buddha* (noise: 0.0034, see Table 7.1). Middle: reconstruction with *reference APSS* [GG07] (reference implementation with marching cubes). Right: local reconstruction (positions+normals) with our method and Screened Poisson [KH13] to generate the mesh.

The local, iterative characteristic of the method presented in this thesis is particularly well suited for non-uniformly sampled point clouds. Figures 7.6 and 7.7 give a comparison with a non-uniformly sampled artificial object. It is basically an ellipsoid with small features on the surface. This test object is well suited for assessing the ability of the resampling method to preserve small features in the presence of noise. This point cloud is resampled with *Meshlab APSS* and our technique. The resampled points are then reconstructed with screened Poisson [KH13]. (The reason why the reference APSS implementation with marching cubes is not used is that not all tested models provide connectivity information (triangular mesh), which is needed by *reference APSS*). In Figure 7.6, the middle image shows the result of the APSS resampling, the lower image is the result of our method. Note how in the lower image, the small features are recovered, whereas the spherical structure of the features is destroyed in the middle sub-image.

In Figure 7.7, a difficult area is shown: The poles of the ellipsoid are densely sampled but highly non-uniformly and noisy. The normal computation algorithm of Meshlab fails to compute the correct vectors because it uses a fixed-size neighborhood. As a result, the reconstruction method (screened Poisson reconstruction [KH13]) produces artifacts that significantly degrade the quality of the surface. This results in a 4.02 times larger Hausdorff distance (maximum) and a 2.26 times higher RMS than with our method.

Another case of non-uniform sampling is shown in Figure 7.8: The *horse*, provided by [KH13], is a point set of 100k points that cover the surface of the horse irregularly. Figure 7.8 shows the effect of extreme noise on the reconstruction: The non-uniformly sampled point cloud is resampled one time using *Meshlab APSS* [GG07] [CCR08] (left sub-image) and one time using our method (right sub-image). Both are then reconstructed using

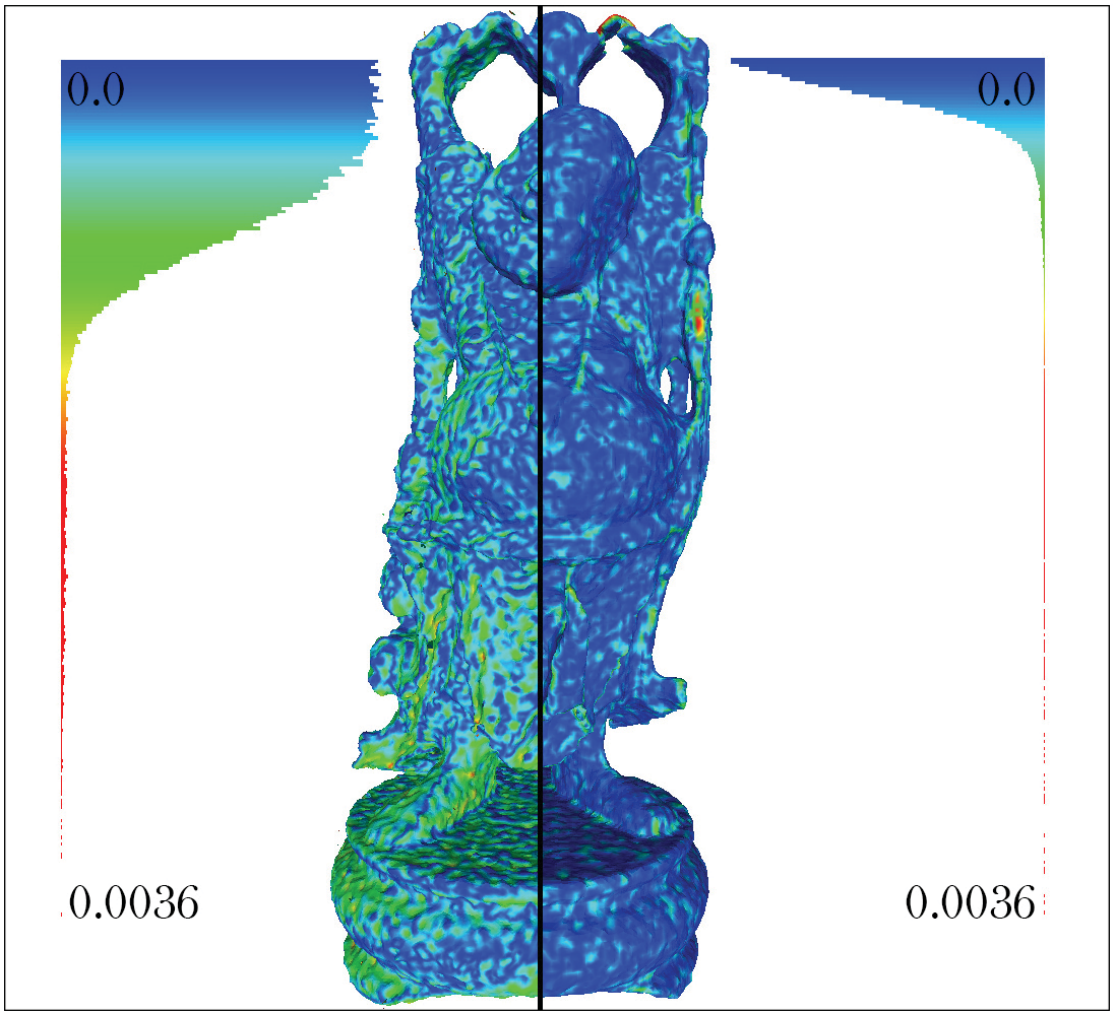


Figure 7.5: Comparison of the distance between the reconstructed mesh and the original *buddha* mesh (added noise: 0.0061, see Table 7.1). Blue encodes small error, red the maximum error. Left: reconstructed with *reference APSS* [GG07]. Right: resampled + normal calculation with our method + mesh generation with screened Poisson reconstruction [KH13]. The histograms show the distributions of the error values: the values are scaled from 0.0 to 0.0036 and mapped to a color scale.

screened Poisson reconstruction [KH13]. Even in this difficult situation, our technique is able to reconstruct the rough geometry and the major features of the point cloud. Since the other method cannot adapt to the very high noise level, the faulty normal vectors produce artifacts during the Poisson reconstruction.

Raw point clouds produced by 3D scanners often suffer from non-uniform noise. In contrast to state-of-the-art techniques for resampling the surface and calculating normal

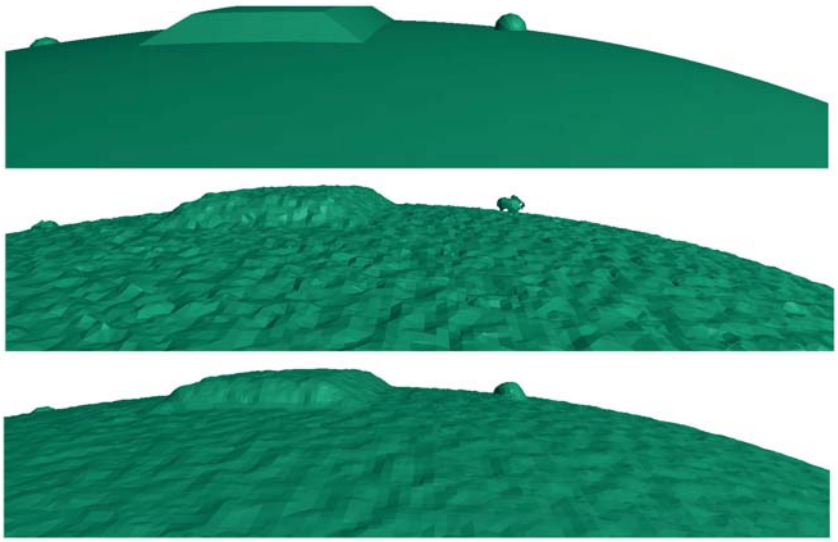


Figure 7.6: Artificial ellipsoid with extended features (upper image) is sampled non-uniformly, noise-extent: 0.0031. Middle: reconstruction with *Meshlab APSS*. Bottom: reconstruction with our method. The small extruded features are preserved with our method.

vectors, our method does not assume a global noise level. Figure 7.9 demonstrates the comparison of our method to APSS resampling [GG07] and the state of the normal reconstruction of Meshlab. The virtual scanner [BLN+13] is used to produce a point cloud by simulating the scanning process of the dragon model (Figure 7.9). The result shows that our technique performs better: The RMS of the Hausdorff distance is 13% better with our algorithm. The visualized Hausdorff distance in Figure 7.9 demonstrates the improvement of the adaptive isotropically fair neighborhoods against the fixed isotropically biased neighborhood definition.

The raw range images of 3D scanners are even more challenging for reconstruction. Although the sampling may be regular in screen space (of the scanner), the samples are generally distributed unevenly among the geometry of the surface. ([ABCO+01], [Bol10]) Surface patches that are perpendicular to the scanning device are sampled differently than surfaces with a steep angle towards the scanner. The noise level is also not uniform for all points: depending on the 3D scanner, the accuracy may not be constant in all directions and is additionally dependent on the geometry and the material of the scanned object.

The Microsoft Kinect™ is a low-cost consumer device capable of generating range images. Figure 7.10 shows the result of the reconstruction of two range images. The left sub-images are reconstructed using Meshlab's normal reconstruction, the right sub-images are generated by our method. The scanned object in both images is an open

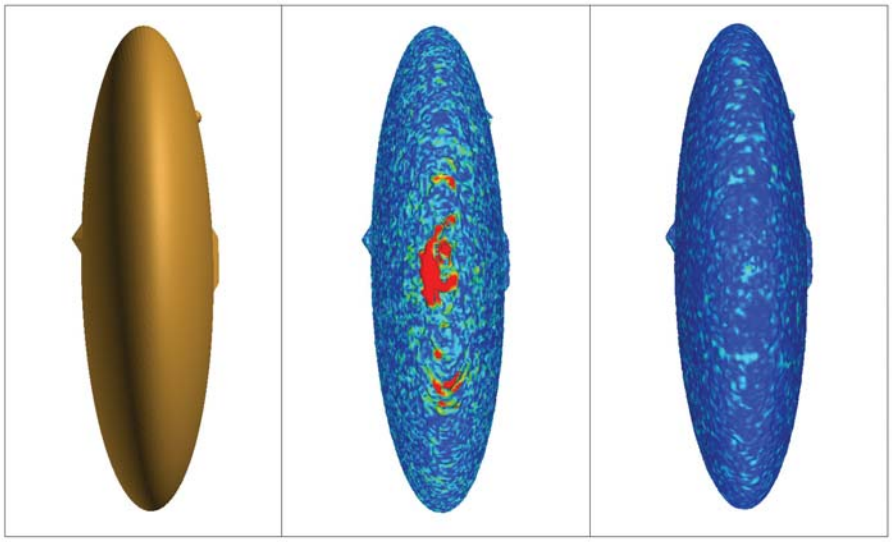


Figure 7.7: Visualization of the distance between the resampled vertices and the original surface. Object is a densely non-uniformly sampled ellipsoid with extruded features, noise-extent: 0.0031 (left image). Our method (right image) adapts to the different sampling condition and produces better normal vectors. This produces a significantly better surface reconstruction. Middle: *Meshlab APSS*

Figure 7.8: Left image: original model without noise. Reconstruction of the vertices with high noise (noise extent: 0.015). Middle: *Meshlab APSS* [GG07] and [CCR08], right image: our method. The resampled point clouds of both methods are then fed into the screened Poisson reconstruction to obtain the final mesh. The blob-like artifacts are caused by wrong normal vectors.

computer case, with small details like cables and large flat surfaces. The blob-like artifacts are produced by incorrect normal computation: If the normals are wrong, the

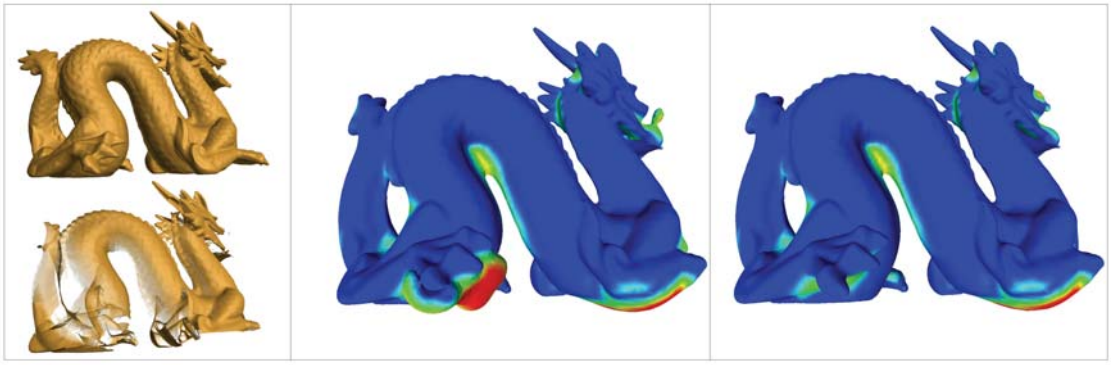


Figure 7.9: Left sub-image: Dragon is scanned by the virtual scanner by [BLN+13]. The middle image shows the resampling with *Meshlab-APSS* [GG07], the right image was resampled with our method. Our method produces less artifacts, particularly at the back foot and the mouth of the dragon.

reconstruction method [KH13] generates these artifacts in order to ensure a closed surface. Notice how our method is able to significantly reduce the artifacts in both cases while preserving fine structures.

7.2 Consistent Normal Orientation

This section will evaluate the performance of our method (see Section 5.2) for making normals globally consistent. Our solution avoids building global data structures to calculate the Riemannian graph, which is common in state-of-the-art methods (see Chapter 5 for details) Nevertheless, the results show that our method has comparable quality while being significantly faster (about 2-3 times).

Table 7.2 depicts the results of the most interesting test cases. Figure 7.11 visualizes this data in a diagram: It shows that our technique can compete with the slower global algorithm. In some cases it even produces better quality. The table also shows that the local agreement scheme (see Section 5.2) improves the robustness and quality of the local approach. The numbers without the local agreement scheme are written in brackets.

One disadvantage of our technique is the unpredictable state of the convergence of the global agreement. The final state (after convergence) of the local agreement is visualized for the Bunny model in Figure 5.5. If the regions of smaller agreement (near 0.5, visualized as red color) completely encircle bigger regions with exact agreement, the orientation cannot be transferred reliably between the encircled blue region and the outside. Thus, one patch which is separated by an enclosing region of small agreement may have the wrong normal orientation. This case occurs for example for the ellipsoid in Table 7.2: Although the method manages to reduce the number of wrong normals for the 0.0 noise model, it performs worse in relation to the global method for other noise

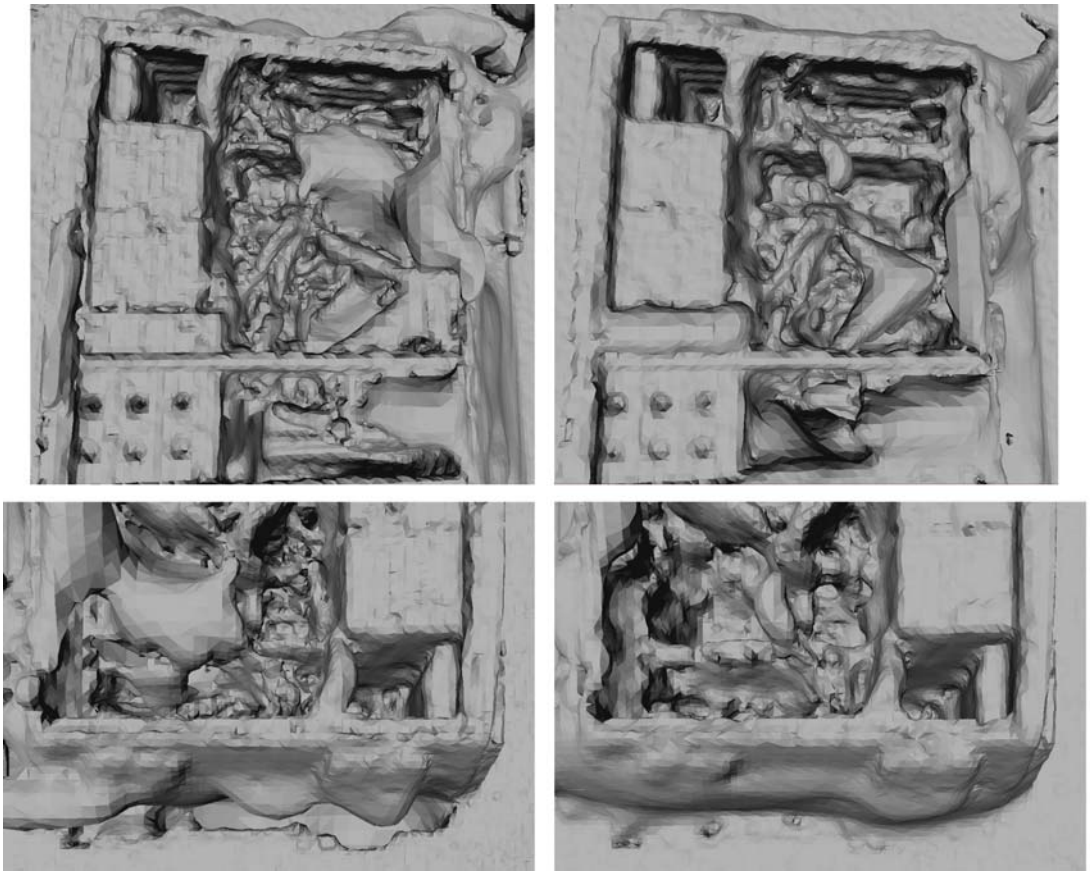


Figure 7.10: Reconstruction of Kinect^{TM} range scans of an open computer case. Left images are generated by *Meshlab APSS*, right images are the result of our method. The normal computation routine of Meshlab produces more faulty normal vectors because of the non-uniform sampling and noise level. This produces the blob-like artifacts during the surface reconstruction. Our method (right images) succeeds in producing more correct oriented normal vectors and thereby reducing these artifacts.

levels. Section 8.2 gives an outlook on how to improve the local approach to alleviate this problem.

A second drawback of the local technique is that the assumptions that are the base of the heuristic for building a connected graph are sometimes violated. The resulting graph then connects vertices that are not in the same vicinity on the actual surface. In these cases, the affected normal is flipped in the wrong direction. However, such mistakes only affect one or only a small number of vertices, since the local consensus prohibits the propagation of the wrong orientation.

A particular failure case concerns elongated, thin structures. If the sampling is not regular and the noise level exceeds a certain limit, the points form several connected

	global graph technique		local graph technique	
	wrong normals	time (s)	wrong normals	time
bunny (0.0)	0 (0.0%)	1.16	2 [2] (0.0056%)	0.543
bunny (0.0019)	1 (0.0028%)	1.162	2 [2] (0.0056%)	0.492
bunny (0.0092)	432 (1.201%)	1.062	4 [4] (0.0111%)	0.501
bunny (0.0167)	239 (0.665%)	0.88	9 [23] (0.025%)	0.655
ellipsoid (0.0)	342 (0.3%)	3.42	75 [351] (0.06%)	1.38
ellipsoid (0.0007)	10 (0.009%)	3.38	93 [164] (0.08%)	1.52
ellipsoid (0.005)	13 (0.014%)	3.08	82 [97] (0.09%)	2.61
omotondo (0.0)	111 (0.022%)	16.6	111 [130] (0.022%)	5.9
omotondo (0.00002)	112 (0.023%)	17.2	113 [116] (0.023%)	5.7
omotondo (0.00006)	112 (0.023%)	16.9	112 [114] (0.023%)	5.9

Table 7.2: Comparison of the normal orientation using global vs. local data structures for building the Riemannian graph (see Chapter 5). Different models with various noise levels (in parentheses) are processed with the global and the local method. The time denotes the total runtime of the orientation algorithm: building the data structures and flipping the wrong normals. The number in square brackets [] is the number of wrong normals using the local technique without the local agreement.

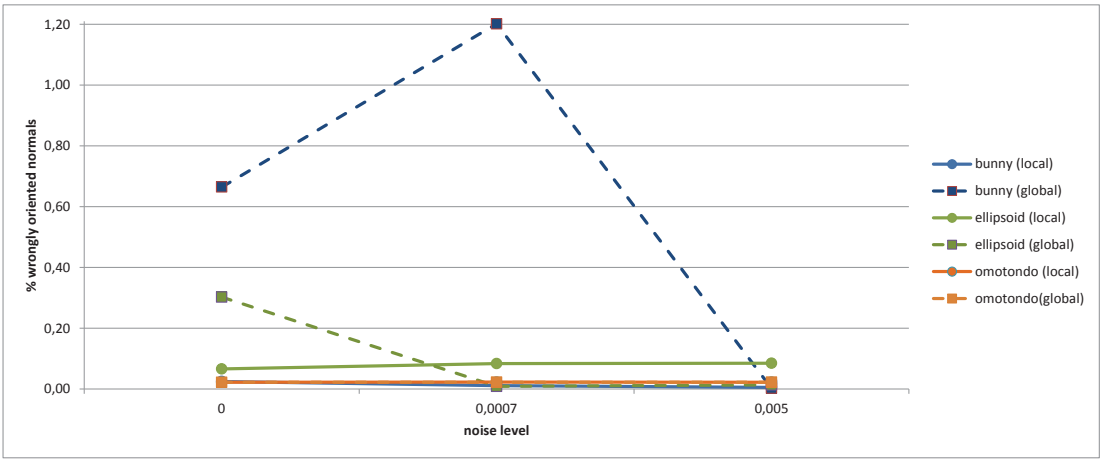


Figure 7.11: Diagram of Table 7.2: The colors encode the model (bunny: blue, ellipsoid with features: green, omotondo: orange), dotted lines are the results of the global method, continuous lines refer to our local technique. For these models our technique is more stable than the method using a global EMST.

components. The simple heuristic used in Section 5.2 cannot combine them to the biggest connected component without introducing edges that intersect other connected

components.

Figure 7.12 visually compares our local method with the standard global approach without local agreement. Since the propagation works by traversing a tree in a depth-first order, one wrong flipping decision can have a huge impact on the successive nodes in the graph. The weighting scheme of [HDD+92] should minimize the probability that a wrongly flipped normal (which is likely to have a big weight) has many successors and that it will be a leaf node. Nevertheless, it is not guaranteed and can deteriorate the result. Our local agreement scheme further reduces the probability of the propagation of wrongly flipped normals. In the left sub-image of Figure 7.12, the failure case of the global method without local agreement is shown. The right image shows a detail view of the same area after being oriented by our local technique.

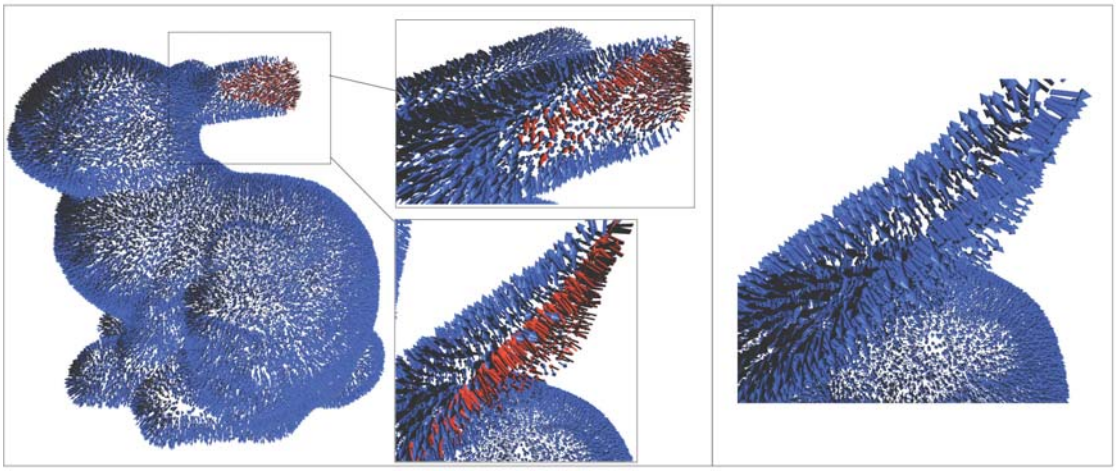


Figure 7.12: The normal vectors of the bunny model are shown: Blue vectors have the correct orientation, red arrows depict normal vectors with a wrong global orientation. Left sub-image: result of the global normal propagation after [HDD+92]. The ear of the bunny, where the wrong orientation occurs, is shown in two detail images. Right sub-image: ear of the same point cloud, where the normal orientation was determined with our technique with local agreement.

7.3 Performance

In this section, the performance and the timings of the discussed algorithms are presented. In general, the computational effort for generating the isotropically fair neighborhoods is non-negligible and higher than if an isotropically biased neighborhood (e.g., k-nearest neighbors) had been used. The effort is not only higher during initialization of the data structure but also during the other stages of the algorithm: bigger memory consump-

tion and more complex memory access leads to a slightly slower execution speed of the regression algorithm.

Table 7.3 lists the run-times of the different stages of our local reconstruction pipeline implemented on the CPU. Figure 7.13 visualizes these times. The Armadillo model is processed at three different noise levels. The majority of the run-time is spent in local functions. Building the neighborhood structure, calculating the noise spheres and the orientation agreement are computed independently and can be parallelized. Building the kd-Tree and searching for the nearest neighbors at the beginning, building the graph for normal orientation and flipping the wrong normals are currently done sequentially. In our tests, the kd-tree of ANN [MA] is used to process the nearest-neighbor search. Executing the neighborhood search in parallel would improve the performance of the regression.

The main bottleneck of the computation is the calculation of the noise spheres. The higher the noise level, the more computationally intensive is the task of generating the noise spheres. The reason for this behavior is that the growing neighborhood scheme 4.2 takes more iterations to find the best suitable neighborhood for each input point.

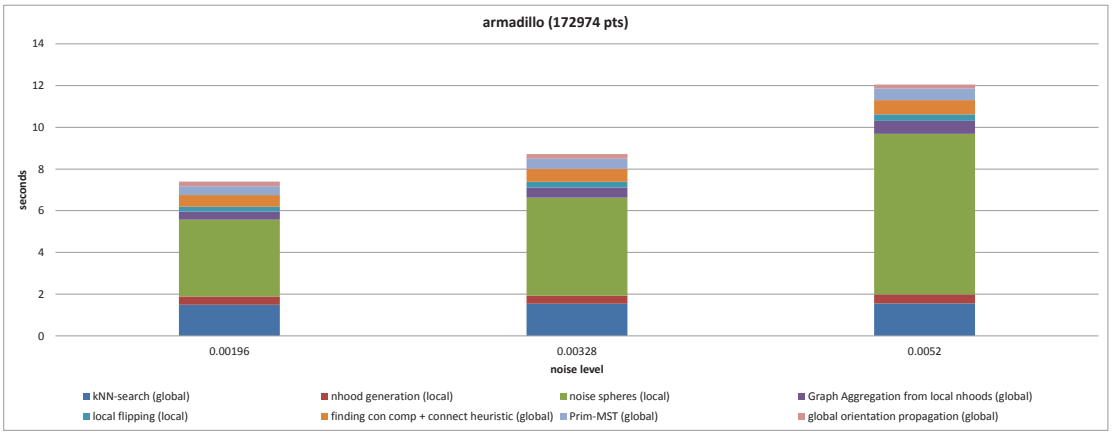


Figure 7.13: CPU performance of the resampling and normal computation and orientation for the *armadillo* point cloud. See Table 7.3.

The advantage of our method (see Chapter 4) is that the expensive part is a pure local function. That means that each noise sphere can be computed independently from all other spheres, only the input points are needed and are not altered by the iterative algorithm. Therefore, the critical part can be parallelized easily. The straight-forward CPU implementation yielded an acceleration of $2-3\times$ on an Intel Core 2 Quad using 4 threads.

The most expensive part of the noise-sphere calculation (about 85%) is the eigenvector computation of the regression. Section 6.1.1 describes an approach how to execute these calculations on the GPU. The actual eigenvalue problem is thereby accelerated by

armadillo (172974 pts)	noise level:		
	0.00196	0.00328	0.0052
kNN-search (global)	1.49809	1.53809	1.55809
nhood generation (local)	0.389	0.395	0.428
noise spheres (local)	3.68	4.7	7.7
Graph Aggregation from local nhoods (global)	0.39	0.475	0.628
local flipping (local)	0.235	0.273	0.303
finding con comp + connect heuristic (global)	0.563	0.632	0.68
Prim-MST (global)	0.414	0.483	0.56
global orientation propagation (global)	0,234	0,2254	0,193

Table 7.3: Run-time of various stages of our local reconstruction pipeline. Compares the run-times in regard to increasing noise level. Point cloud is from the armadillo model. Figure 7.13 illustrates this table.

size:	16283	60955	138799	187948	264051	372479	435545	498905	595358	882954
CPU (full)	0.226	0.82	1.908	2.257	2.972	5.1733	6.14	7.212	8.3	12.0357
CPU (symmetric)	0.122	0.45	1.035	1.227	1.61	2.77	3.18	3.64	4.42	6.38
GPU (symmetric)	0.008	0.024	0.053	0.063	0.084	0.143	0.54	0.366	0.228	0.556

Table 7.4: Run-times of different 5×5 eigenvalue solvers: CPU (full): asymmetric eigenproblem, CPU (symmetric) and GPU (symmetric) solve the transformed symmetric problem (Section 6.1.1).

a factor of 20. The graphs in Figure 7.14 compare different algorithms for the eigenvector computation of 5×5 matrices. The naive implementation (green) executes the general eigenvalue algorithm on the matrices. The reformulation of Section 4.3 has the benefit of only requiring the eigenvalues for a symmetric matrix, which has only real eigenvalues. The performance of the CPU implementation of the symmetric approach (red) has a speedup of about 1.88. The same method implemented on a GPU (NVidia GTX770) has a speedup of about 18.4 compared with the symmetric CPU implementation (blue). This speedup already includes the transfer times between the host memory and the device memory.

Although the eigenvalues can be calculated quickly, the overall performance gain over the single threaded CPU implementation is slightly less than the multi-core CPU implementation (on an Intel Core2 Quad). The whole pipeline of spherical regression is changed (see Section 6.1.1 for details). This results in an efficient regression sphere calculation, but the whole iterative function is slower. Handling the neighborhood (generation, enlargement) and calculating the matrices from the neighborhoods is still done in a serial way on the CPU. Additionally, the data has to be transferred between the CPU and the GPU. A pure GPU implementation, which would also handle the neighborhoods, would not suffer from these shortcomings and would be significantly faster.

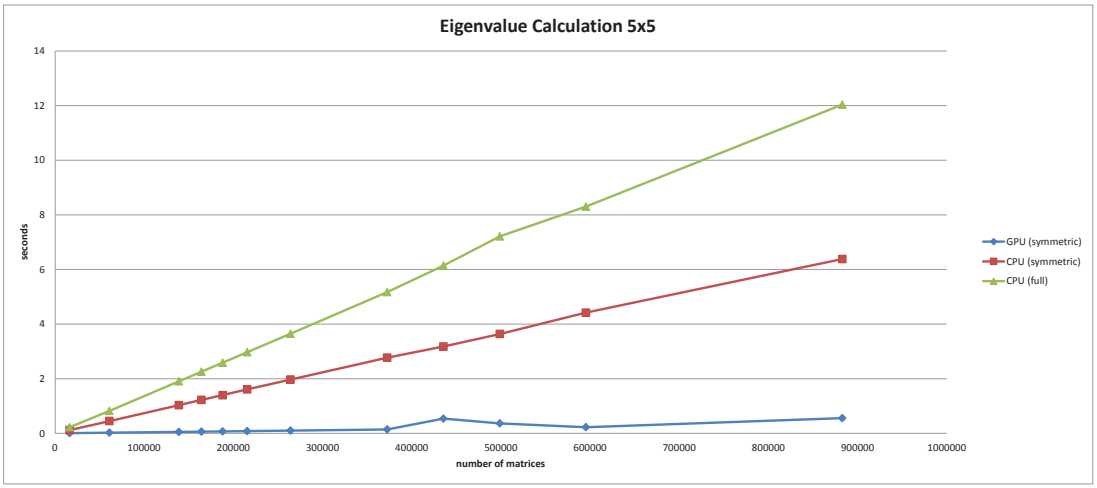


Figure 7.14: Comparison of eigenvector calculations for 5×5 matrices: The CPU implementation of the symmetric algorithm (red) takes nearly $20\times$ the time of the GPU implementation (blue). The CPU implementation of the general eigenvalue algorithm is nearly twice as slow as the symmetric CPU implementation. Also see Table 7.4

	APSS [GG07]	CLOP [PMA$^+$14]	Our (CPU quad)	Our (CPU single)	Our (GPU)
bunny (35947 pts)	0.51	0.179	1.59	4.05	1.39
horse (99998 pts)	19.60	0.377	10.79	30.26	10.56
buddha (144647 pts)	8.53	0.351	3.48	7.69	3.51
armadillo (172974 pts)	18.33	0.374	7.80	21.91	6.71
dragon (435545 pts)	7.30	0.543	11.25	27.05	10.17
omotondo (498938 pts)	63.40	0.78	10.83	26.62	11.22
blade (882954 pts)	131.55	1.195	16.28	33.41	16.69

Table 7.5: Comparison of the run-times of different resampling implementations: The normal computation and APSS implementation of Meshlab[CCR08], GPU implementation of CLOP [PMA$^+$14] and our method: single and multi-threaded CPU implementation and a GPU implementation. See also Figure 7.15

Section 8.2 gives an outlook on how to further improve the resampling performance.

Figure 7.15 and Table 7.5 compare the runtime of resampling implementations: Our method (CPU single-threaded, multi-threaded and GPU assisted) is compared to the Meshlab implementation of APSS [GG07] (and normal vector computation) and a highly optimized GPU implementation of CLOP [PMA$^+$14]. The CPU implementations are executed on an Intel Core2 Quad with 3GHz, the GPU assisted implementation of our

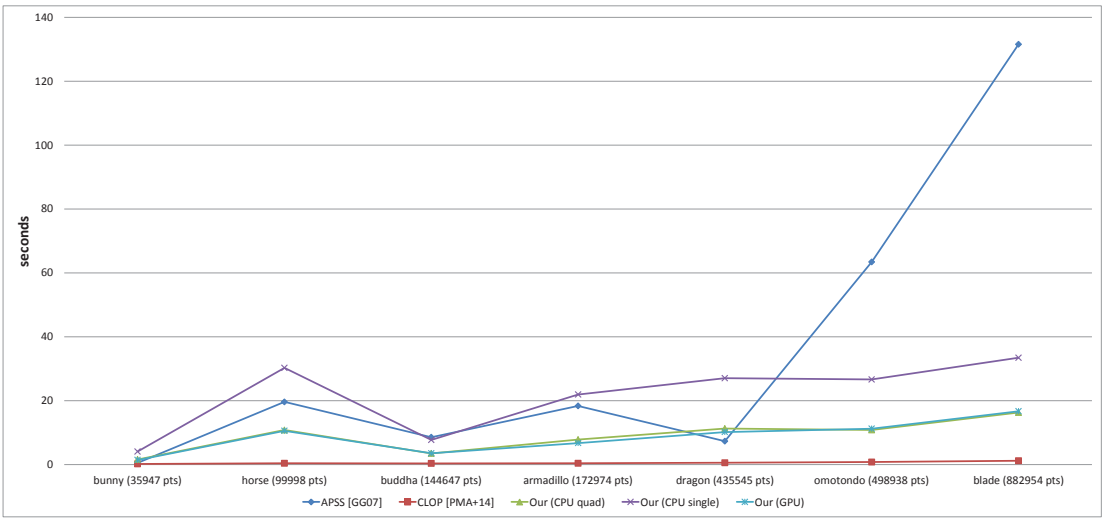


Figure 7.15: Run-time of various resampling implementations: *Meshlab APSS*: Using the normal computation method + the APSS [GG07] projection. *CLOP*: Highly optimized GPU implementation [PMA$^+$14]; parameters are determined manually for each point cloud. Our method is implemented in a single core, multi-threaded and GPU assisted implementation.

method is run on an NVidia GTX770 GPU and CLOP is executed on an NVidia GTX TITAN GPU. Concerning the run-times of the algorithms, the CLOP implementation outperforms APSS and our method by one order of magnitude. However, this does not account for the time spent for adjusting the parameters to gain a reasonable resampling. Because *Meshlab APSS* is composed of two methods (calculating the normals + APSS projection), the kd-tree is built twice.

The speed comparison in Figure 7.15 shows that our method has overall better performance than *Meshlab APSS*. A direct resampling performance comparison with *reference APSS* is not feasible, because the reference implementation of APSS [GG07] reconstructs the global surface using marching cubes.

The global normal orientation procedure heavily relies on global data structures. Usually, an Euclidean minimum spanning tree is required to ensure global connectedness of the input points. The results of our tests show that in many cases, this global connectivity approach is not necessary. Instead, the aggregation of the local neighborhoods is sufficient to model the global connectivity. The presented heuristic in Section 5.2 ensures that all points are connected and yields a better performance, especially for bigger point clouds ($> 100k$ points). Table 7.2 shows the timings of different models and shows a median speedup of 2.24 between the traditional global approach and the local method.

Figure 7.16 and Table 7.6 compare the performance of our method with the global,

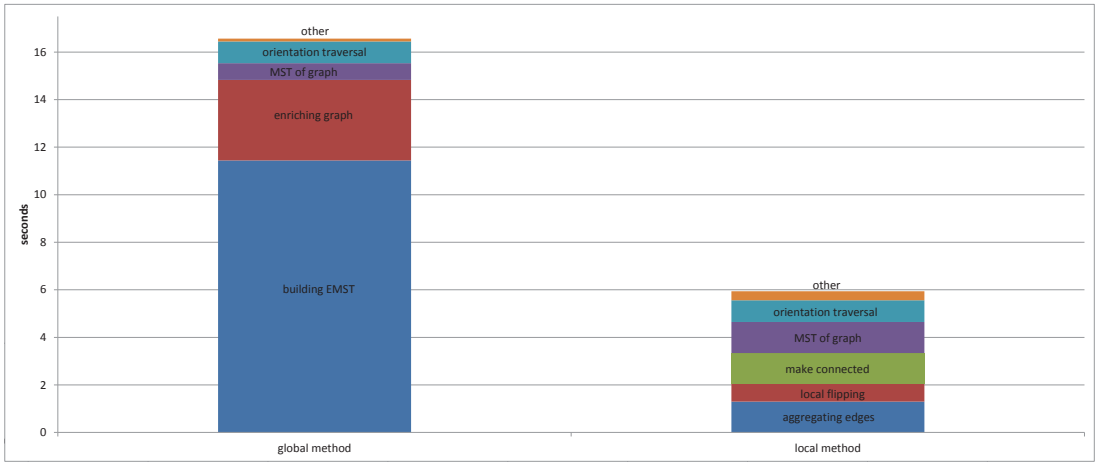


Figure 7.16: Runtime comparison between the global technique (approach after [HDD+92], [XWH+03]) and our technique (Section 5.2). The input point cloud is the *omotondo*, noise: 0.0 (see Table 7.2), with 498905 vertices. Global: building the EMST for generating the Riemannian graph, enriching it with further edges, calculating the MST of the Riemannian graph and traversing the MST to orient the normals. Local: Aggregate the edges of the local neighborhoods, determine the local agreement, heuristic to ensure that the resulting graph is connected, build the MST of this graph and traverse it to orient the normals. *other* are operations like determining the weights of the graph and handling data structures.

EMST based approach ([HDD+92], [XWH+03]). The most expensive part of the global technique is building the EMST (which is derived from a Delaunay triangulation in this implementation). Enriching the graph – finding additional edges for the EMST – is also a non-trivial part of the overall runtime.

Our technique's runtime is primarily determined by the aggregation of the edges to form a global set of unique edges, making the graph connected with our heuristic and building the MST of the resulting graph. Those are inherently global operations that cannot be accelerated easily by parallelization. The local flipping can be further improved by executing it in parallel for each point.

global		local	
step	seconds	step	seconds
building EMST	11.4400	aggregating edges	1.3000
enriching graph:	3.3862	local flipping	0.7420
MST of graph	0.7090	make connected	1.3000
other	0.1227	MST of graph	1.3040
orientation traversal	0.9091	other	0.3773
		orientation traversal	0.9121

Table 7.6: Listing of the run times of the single steps of the normal orientation of the textitomotondo point cloud, noise: 0.0. *global* uses an approach after[HDD+92],[XWH+03], *local* denotes our technique using the local neighborhoods. Also see Figure 7.16

CHAPTER 8

Conclusion

8.1 Synopsis

The use of isotropically fair neighborhoods enhances the reconstruction quality of noisy point clouds. Previous techniques use only the positions and the distances of neighboring points. The proposed *N-ring* neighborhood (Section 4.1) succeeds in gathering additional information about the spatial relation between the vertices of the unorganized point cloud. This leads to improved surface reconstruction, particularly in non-uniform sampling situations in which isotropically biased methods perform worse.

Our parameter-free regression method can roughly estimate the noise level at each input point and adapts to the noise extent. This results in good reconstruction quality without the need for user interaction or other additional data. This also makes the method insensitive to non-uniform sampling conditions and varying noise levels across a point cloud.

The local nature of the method allows for a fast parallel implementation. As the most critical step, we have implemented the sphere regression (which is a 5×5 eigenvalue problem) on the GPU. The data structures built for the isotropically fair neighborhoods can be reused in later stages of the surface reconstruction. Determining a globally consistent normal orientation requires a good approximation of the global connectivity of the underlying surface. The union of all local isotropically fair neighborhoods is a suitable and more efficient replacement for other data structures which have to be built with global algorithms. Together with a local normal-orientation scheme, this results in a faster generation of an oriented point cloud than with previous methods.

8.2 Future Work

The presented N-ring neighborhood in Section 4.1 is only one possible isotropically fair neighborhood. A Voronoi-based neighborhood would be an interesting alternative: For

each point, a local Voronoi diagram (for example used in [DG03]) is calculated and the points are then additionally weighted by the size of the corresponding Voronoi cell. This would also include additional information about the spatial relation between the adjoining points to the neighborhood.

Currently, our method treats input-points as samples which are subject to uniform Gaussian noise. Modeling the surface as a continuous probability density function (similar to the mixture of Gaussians approach in [PMA+14]) may increase the quality and flexibility of the reconstruction. Information about the sensor noise (provided by the 3D scanner) can then be incorporated into the local regression, which may produce a better approximation for the underlying surface – especially for range images. Normals provided by the scanner – if available – may also be included in the regression. In [GG07], the normal vector of a point is used as a normal constraint, which has the benefit of turning the sphere regression into a system of linear equations (instead of an eigenvalue problem). Future work could extend this method to include the normals as fuzzy constraints. This could increase the accuracy of the reconstruction as well as the performance (solving a system of linear equation instead of an eigenvalue-problem).

One drawback of the adaptive spherical regression method is that it is not well suited for sharp features. An extension of the current method could tag sharp features by using a local l1 kernel. The sharp features – edges and corners – could then be handled separately for better reconstruction. A similar approach is suggested in [GG07].

Currently, the implementation does not use the GPU optimally. The generation of the neighborhood structure, which is now done on the CPU, is also a local operation that can be executed on the GPU. Furthermore, the whole adaptive regression can be implemented on the GPU in parallel. This would exploit the local nature of the algorithm and could significantly increase the performance of the method. The adaptive resampling method 4.2 can be combined with an interpolating reconstruction method to complete the reconstruction pipeline (from unorganized points to triangular mesh): creating local consensus triangles (see Figure 6.2) out of the resampled points recovers large portions of the surface. A post-processing step with an interpolating method similar to [OMW13] would fill the holes and would create the final triangulation.

The local agreement approach for the normal-orientation step (Section 5.2) can be used to segment the normals and generate connected patches: The normals are currently locally flipped until convergence. Afterwards, the normals could be clustered into patches with perfect agreement and regions with smaller agreement. Figure 5.5 visualizes the converged agreement values. A global method then would find a consistent orientation between all patches. This would significantly reduce the work of the global method because the corresponding graph has only the size of the number of patches.

Bibliography

[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.

[ABL03] Dominique Attali, Jean-Daniel Boissonnat, and André Lieutier. Complexity of the delaunay triangulation of points on surfaces the smooth case. In *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*, SCG '03, pages 201–210, New York, NY, USA, 2003. ACM.

[AGJ02] Udo Adamy, Joachim Giesen, and Matthias John. Surface reconstruction using umbrella filters. *Computational Geometry*, 21(1-2):63 – 86, 2002. Sixteenth European Workshop on Computational Geometry - EUROCG-2000.

[AS13] James Andrews and Carlo H. Séquin. Type-constrained direct fitting of quadric surfaces. *Computer Aided Design and Applications*, 2013. To appear. Will be presented at CAD'13.

[AsC09] A. Al-sharadqah and N. Chernov. Error analysis for circle fitting algorithms. *Electronic Journal of Statistics*, pages 886–911, 2009.

[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.

[Bol10] Matthew Grant Bolitho. *The Reconstruction of Large Three-Dimensional Meshes*. PhD thesis, The Johns Hopkins University, 2010.

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

[CCR08] Paolo Cignoni, Massimiliano Corsini, and Guido Ranzuglia. Meshlab: an open-source 3d mesh processing system. *ERCIM News*, (73):45–46, April 2008.

[CGA] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

[CGGS13] S. Cuomo, A. Galletti, G. Giunta, and A. Starace. Surface reconstruction from scattered point via rbf interpolation on gpu. In *Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on*, pages 433–440, Sept 2013.

[CL05] N. Chernov and C. Lesort. Least squares fitting of circles. *J. Math. Imaging Vis.*, 23(3):239–252, November 2005.

[CRS96] Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. Metro: Measuring error on simplified surfaces. Technical report, Centre National de la Recherche Scientifique, Paris, France, France, 1996.

[CSRL01] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[DG03] Tamal K. Dey and Samrat Goswami. Tight cocone: A water-tight surface reconstructor. In *Proceedings of the Eighth ACM Symposium on Solid Modeling and Applications*, SM '03, pages 127–134, New York, NY, USA, 2003. ACM.

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

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

[GKS01] M Gopi, Shankar Krishnan, and Claudio T. Silva. Surface reconstruction based on lower dimensional localized delaunay triangulation. *Computer Graphics Forum*, 19(3):467–478, 2001.

[GR04] David Gisch and Jason M Ribando. Apollonius' problem: A study of solutions and their connections. *American Journal of Undergraduate Research*, 3(1), 2004.

[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 Inter-*

active Techniques, SIGGRAPH '92, pages 71–78, New York, NY, USA, 1992. ACM.

[HLZ+09] Hui Huang, Dan Li, Hao Zhang, Uri Ascher, and Daniel Cohen-Or. Consolidation of unorganized point clouds for surface reconstruction. In *ACM SIGGRAPH Asia 2009 Papers*, SIGGRAPH Asia '09, pages 176:1–176:7, New York, NY, USA, 2009. ACM.

[KA08] Yong Joo Kil and Nina Amenta. Gpu-assisted surface reconstruction on locally-uniform samples. In *Proceedings of the 17th International Meshing Roundtable, Pittsburgh, Pennsylvania, USA, October 12-15, 2008.*, pages 369–385, 2008.

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

[KG09] Sören König and Stefan Gumhold. Consistent propagation of normal orientations in point clouds. In Marcus A. Magnor, Bodo Rosenhahn, and Holger Theisel, editors, *VMV*, pages 83–92. DNB, 2009.

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

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

[Lev01] David Levin. Mesh-independent surface interpolation. In *Advances in Computational Mathematics*. in press, 2001.

[MA] David M. Mount and Sunil Arya. ANN: A library for approximate nearest neighbor searching. https://www.cs.umd.edu/ mount/ANN/.

[MN03] Niloy J. Mitra and An Nguyen. Estimating surface normals in noisy point cloud data. In *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*, SCG '03, pages 322–328, New York, NY, USA, 2003. ACM.

[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.

[OMW13] Stefan Ohrhallinger, Sudhir Mudur, and Michael Wimmer. Minimizing edge length to connect sparsely sampled unorganized point sets. *Computers & Graphics (Proceedings of Shape Modeling International 2013)*, 37(6):645–658, Oct 2013.

[PMA+14] Reinhold Preiner, Oliver Mattausch, Murat Arikan, Renato Pajarola, and Michael Wimmer. Continuous projection for fast l1 reconstruction. *ACM Transactions on Graphics (Proc. of ACM SIGGRAPH 2014)*, 33(4):47:1–47:13, August 2014.

[Pra87] Vaughan Pratt. Direct least-squares fitting of algebraic surfaces. *SIGGRAPH Comput. Graph.*, 21(4):145–152, August 1987.

[STN88] Knut Stamnes, Sfchee Tsay, and Teruyuki Nakajima. Computation of eigenvalues and eigenvectors for the discrete ordinate and matrix operator methods in radiative transfer, 1988.

[XWH+03] Hui Xie, J. Wang, Jing Hua, Hong Qin, and A. Kaufman. Piecewise c1 continuous surface reconstruction of noisy point clouds via local implicit quadric regression. In *Visualization, 2003. VIS 2003. IEEE*, pages 91–98, Oct 2003.