METHODS FOR DETECTION AND RECONSTRUCTION OF SHARP FEATURES IN POINT CLOUD DATA

vom Fachbereich Informatik der

Technischen Universität Kaiserslautern

zur Verleihung des akademischen Grades

Doktor der Naturwissenschaften (Dr. rer. nat.)

genehmigte Dissertation

von

Dipl.-Inf. Christopher Dominik Weber

Erster Berichterstatter:
Prof. Dr. Hans Hagen
Zweiter Berichterstatter:
Prof. Dr. Stefanie Hahmann
Vorsitzender der Prüfungskommision:
Prof. Dr. Klaus Schneider

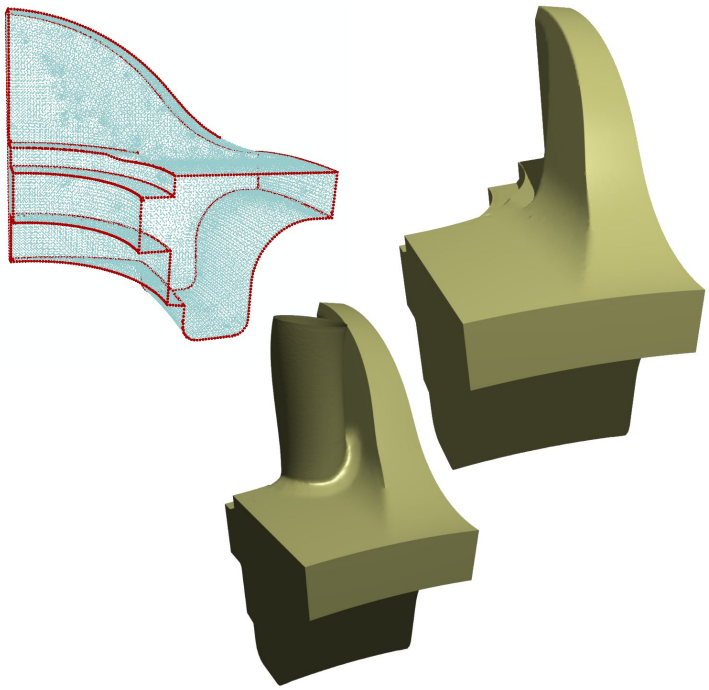
Dekan des Fachbereichs:
Prof. Dr. Arnd Poetzsch-Heffter

Datum der wissenschaftlichen Aussprache: 23. August 2011

D 386

METHODS FOR DETECTION AND RECONSTRUCTION OF SHARP FEATURES IN POINT CLOUD DATA

CHRISTOPHER DOMINIK WEBER



Computer Graphics and HCI Group
Department of Computer Science
Technische Universität Kaiserslautern

Advisors:
Prof. Dr. Hans Hagen
Prof. Dr. Stefanie Hahmann

April 2011

D 386

ACKNOWLEDGMENTS

I would like to thank all the people around me that made it possible to write this thesis with their help and support. Above all, I would like to thank my family, especially my parents and sister. They all gave me the support and backing I needed during the research and writing of this thesis.

This thesis would not have been possible without the help, support and patience of my supervisors Prof. Dr. Hans Hagen [1] and Prof. Dr. Stefanie Hahmann [2]. They were always an invaluable source of advice and knowledge for which I am extremely grateful. I would also like to acknowledge the financial, academic and technical support of the IRTG 1131 of the German Research Foundation and the Technische Universität Kaiserslautern and its staff as well as my friends and colleagues at the Computer Graphics and HCI Group and the IRTG. Our open meetings and dicussions were always a huge inspiration for the solution of upcoming problems.

Last but not least I would like to mention and thank the sources for the examples and datasets I used in this thesis. The vase model is provided courtesy of INRIA, the fandisk, the ocata-flower and the trim-star are provided courtesy of MPII, all by the AIM@SHAPE Shape Repository. The drill model is courtesy of Sergei Azernikov, Siemens Corporate Research, Princeton.

1 Technische Universität Kaiserslautern, Computer Graphics and HCI Group, Germany
2 Université de Grenoble, Laboratoire Jean Kuntzmann, France

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Part I

INTRODUCTION

1

INTRODUCTION

Many applications rely on the quality of the geometric models representing the relevant object. They vary from special effects in movies to stress analysis during earthquakes, from medical applications to 3D printers in rapid prototyping. A large branch of Computer Science is engaged with the efficient acquisition, presentation, manipulation, analysis and reconstruction of 3-dimensional objects in the computer. This opens the doors for many applications covering a wide area from CAD/CAM, multimedia and entertainment, to scientific visualization and medical imaging.
There exist many different ways for the representation of those often complex surfaces, for example using geometric primitives like points or polygons, subdivision surfaces, or implicit functions.
Today, polygonal models occur everywhere in graphical applications, since they are easy to render and to compute and a very huge set of tools are existing for generation and manipulation of polygonal data. But modern scanning devices that allow a high quality and large scale acquisition of complex real world models often deliver a large set of points as resulting data structure of the scanned surface. A direct triangulation of those point clouds does not always result in good models. They often contain problems like holes, self-intersections and non manifold structures. Also one often looses important surface structures like sharp corners and edges during a usual surface reconstruction. So it is suitable to stay a little longer in the point based world to analyze the point cloud data with respect to such features and apply a surface reconstruction method afterwards that is known to construct continuous and smooth surfaces and extend it to reconstruct sharp features.

In this thesis we present such a method and additional algorithms for analysis, surface reconstruction and also combination of designed and scanned data. Most data used will be point cloud data as it is received from scanning devices. So all methods and algorithms presented in this thesis will be point based.

The main part of this thesis is divided in three chapters.
The first main chapter (Chapter 3) contains a new method for recognition and identification of sharp features in a point cloud sampled from a scanning device. The new method introduced does not rely on additional information like normals or connectivity, but only the spatial locations of the data points. As a complete new approach we use the gaussian sphere in combination with a clustering algorithm to detect the sharp features. In addition an iterative refinement of the method is introduced, which makes the method complete automatic without need for user interaction. Moreover, it improves the results of the global and user controlled method using local and optimized parameters. To allow the method to be effective on larger scale data, it is based on local neighborhoods. For each point in the point set, the local neighborhood around the point is used to determine a sharp feature. Normals of local approximations are projected onto the gaussian sphere around the actual point. After this a clustering algorithm analyses the patterns in this projection and rate these patterns to decide whether the point lies on a sharp feature or not.

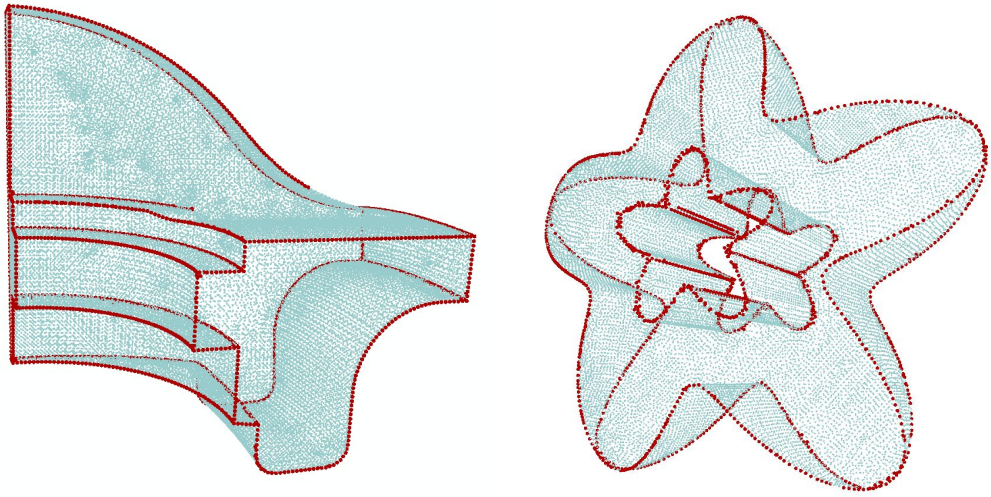


Figure 1: Examples for sharp feature detection

The second main chapter (Chapter 4) is about the reconstruction of surfaces based on a point cloud and especially the reconstruction of the sharp features identified in Chapter 3. Here a modification of the popular moving least squares (MLS) method is used. As the usual MLS is not capable of producing sharp features we decided to modify and adapt the method to allow a sharp feature reconstruction. We use the property of

MLS being a combination of local approximations. In a first step we check if the actual region contains sharp features. We can just do a normal MLS if the regions turns out to contains no sharp features. Otherwise, we use the modified approach to reconstruct the sharp feature.

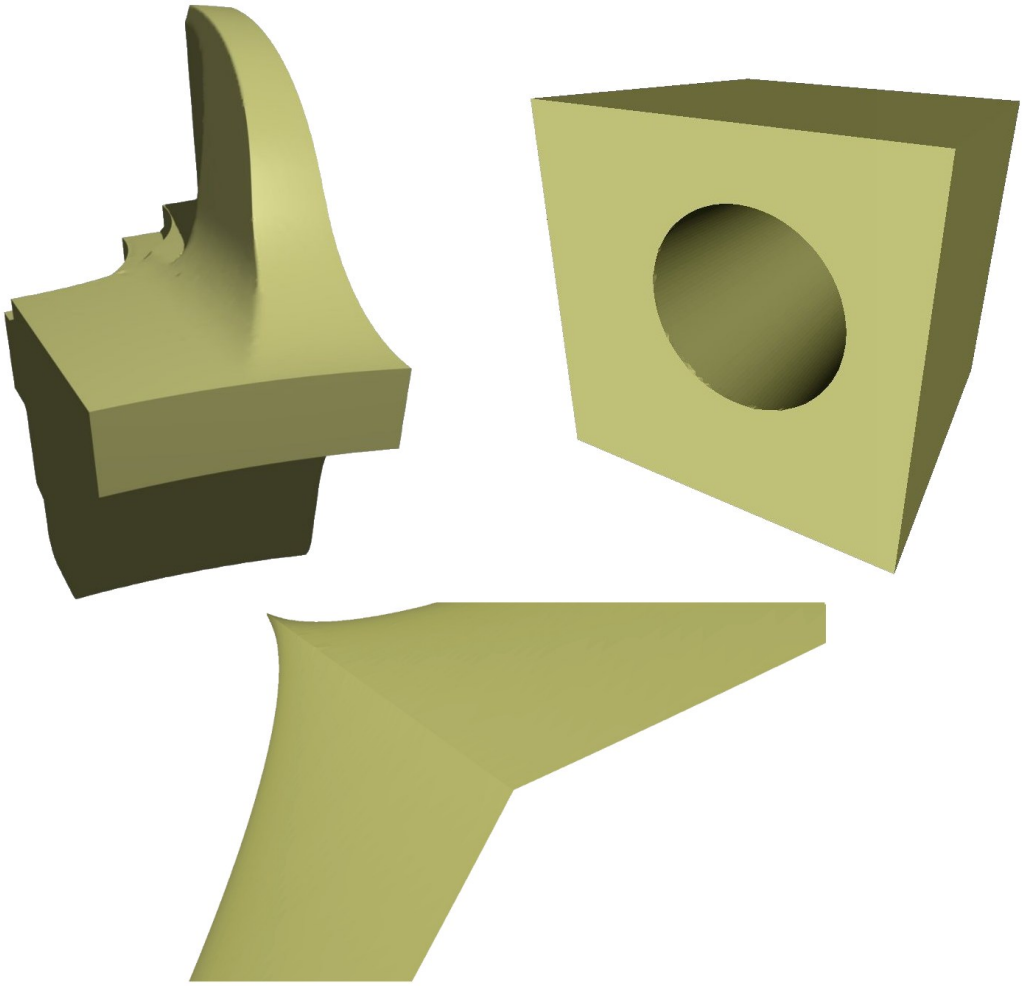


Figure 2: Examples for the sharp feature MLS-reconstruction

For the final sharp feature reconstruction the most important step is the modification of the neighborhood that we use for the local least squares approximation. Based on the known positions of the sharp features, the neighborhood can be modified in a way that the MLS does not blend the sharp feature. During the MLS reconstruction we first check for the actual local point set if it contains a sharp feature. If this is not the case, the usual MLS reconstruction is performed. Else the feature points in the local neighborhood

are used to construct a local feature line that divides the neighborhood along the sharp feature. Points on the wrong side of the feature line are discarded for the reconstruction. This way of modification of the local neighborhoods makes sharp feature reconstruction possible for simple point based datasets without any further information like normals or neighboring structure.

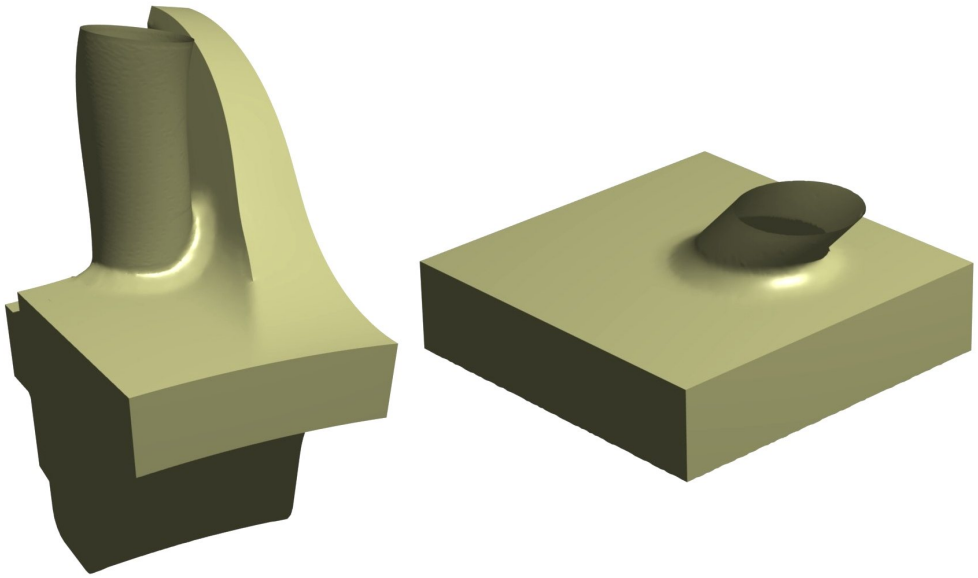


Figure 3: Examples for blending of a NURBS- and a MLS-surface

The third and last main chapter (Chapter 5) is about the combination of surfaces from the MLS reconstruction of point clouds and NURBS surfaces from e.g. a CAD design system. Here the sharp feature reconstruction from Chapter 4 will be used in a new context. The goal of this chapter is the construction of a smooth blending between the MLS surface and the NURBS surface. This combination of point based and algebraic surfaces is not very common and a challenging task since those two types of surface representation have very less in common. Since we wanted to stay in the world of point based methods, we sample the NURBS surface to construct a second point cloud as compatible surface type for the blending step. For the blending we generate a third point set representing the blending area. To offer flexibility in the design of this blending area, we use two parameters to control hight and width of the 'collar like' area constructed for the blending. During the construction of this blending area, we remove the parts of the original surfaces that overlap and generate offsets in both surfaces. After we have

three point sets which we can modify and combine. The original base point set of the MLS surface, a point set sampled from the NURBS surface and a point set constructed to blend the other two point sets in a smooth manner.

In the end, the three point sets are combined as global point set and the MLS reconstruction form Chapter 4 can be applied to generate a global surface for the combined data. This surface contains the combined NURBS and MLS surfaces blended in a smooth manner, but due to the characteristics of the sharp feature reconstruction used it will also conserve the sharp edges and corners of the original data sets.

The main contributions of the thesis are:

- A new method for the detection of sharp features in point cloud data based on gaussian clustering.
- An adaptive and iterative extension of the sharp feature detection which makes the method independent from user controlled parameters.
- A moving least squares based method for surface reconstruction of point cloud data that conserves sharp features.
- A method for blending of point based moving least squares surfaces and NURBS surfaces.

Part II

BASICS

2

BASICS

IIn this chapter we will give an overview and short introduction to concepts definitions and methods related to the subject of this thesis. We will start with an overview about point sets since all methods in this thesis are based on point sets. Than we will introduce Moving Least Squares, our method of choice for the surface reconstruction part in Chapter 4, and the last basics section will be about NURBS and parametric curves since we will use these types of curve and surface representation in Chapter 5.

2.1 POINT SETS

The methods presented in this thesis are all based on point set data. Point sets are a common source of data in computer graphics. Today with emerging scanning technologies, point set data can be found in a wide range of applications. Although point sets are over all a quite simple data structure, they demand for several problem solutions. One is the pure number of data points. Especially regarding the progress of the different scanning techniques the point sets that are produced become larger and larger. Point sets of several hundreds of thousands or several millions of data points are getting more and more common. This demands for special treatment in the used algorithms. Solutions working on a local basis are always a good way to deal with those large scale problems by cutting it down to multiple but better manageable problems. The methods used in this thesis will all be such local methods.
This leads to a second problem. To use a local method one needs to know the data in the local area of interest. Since most points cloud data is unsorted and not equipped with information about the structure and neighboring information of the underlying data one also needs well designed data structure to store and if necessary sort the data.

The data in the point sets, respectively the 'points' itself can vary in the kind of information contain. Some of these point types are shown in the next sections.

2.1.1 *Simple points*

The most basic type of point data and thus the data type that we apply to our algorithms. It contains only the pure information about the position in the given space. A point set of simple points consists only a simple list of coordinates for each space dimension. We work in the 3-Dimensional space, thus the simple point data for a point P is

$$P = \{x, y, z\} \text{ with } x,\ y,\ z\ \in \mathbb{R}.$$

2.1.2 *Oriented points*

Oriented points are just like simple points, but they contain additional information about the data they are representing. In the case of a point set representing a surface, an oriented point contains the normal vector of the given surface at its position.

$$P = \{x, y, z, \vec{n}\} \text{ with } x,\ y,\ z\ \in \mathbb{R},\ \vec{n} \in \mathbb{R}^3.$$

If the normal information is not available in the data set it can be approximated, for example like in the work of Alexa [3] or Dey [11]. For a good approximation, the density of the point cloud has to be high enough. Usually the approximation is done by an analysis of the neighborhood of the point.

2.1.3 *Splats*

Splats are the next extension of oriented points. Additional to the coordinates and the direction those objects also contain a radius, thus forming a disc approximating the local surface around this point. In a generalization, those discs can be defined as elliptical splats by two tangential axes and their respective radii.

Figure 4 from [28] shows the aforementioned basic types. The simple point, the oriented point with its additional normal vector and the two splat types with circular and elliptical disc.

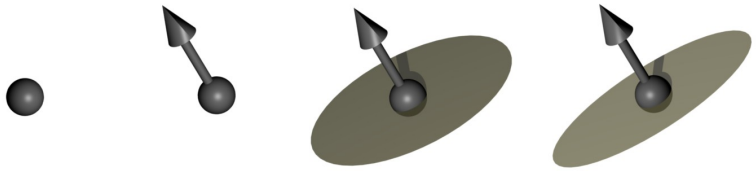


Figure 4: From left to right: simple point, oriented point, splat, elliptical splat

2.1.4 *Point set algorithms*

2.1.4.1 *Splatting*

Splatting is a common representation technique for point sets. It was proposed first by Zwicker et. al in 2001 [49]. To close the gaps between the points in the point cloud, the points are equipped with normal information and a radius, called splat. This generates circular disks around the point representing a local approximation the underlying surface. A further improvement of this technique uses not only circular, but elliptical splats, defined by two tangential axes and their respective radii. If these axes are aligned to the principal curvature directions of the underlying surface and the radii defined inverse proportionally to the minimum and maximum curvature, the approximation can be improved further. For the presentation of sharp edges, one needs to clip the involved splats against clipping lines in their specific local tangent frames. Often this is done using two splats sharing the same center but equipped with two normals from the two surfaces at the sharp edge.
From differential geometry one can conclude that a local ellipse is the best linear approximant of a smooth surface. They provide the same quadratic approximation order as triangle meshes. Figure 5 from [28] shows a comparison between rendering of a mesh and splat representation of an object

One advantage of splats over points or oriented points is, that the point density, necessary for a good surface representation can be easy adapted. For example it is better to adapt the density of points to the local properties, so that regions with a high curvature contain more and smaller splats than flat regions. In contrast, simple

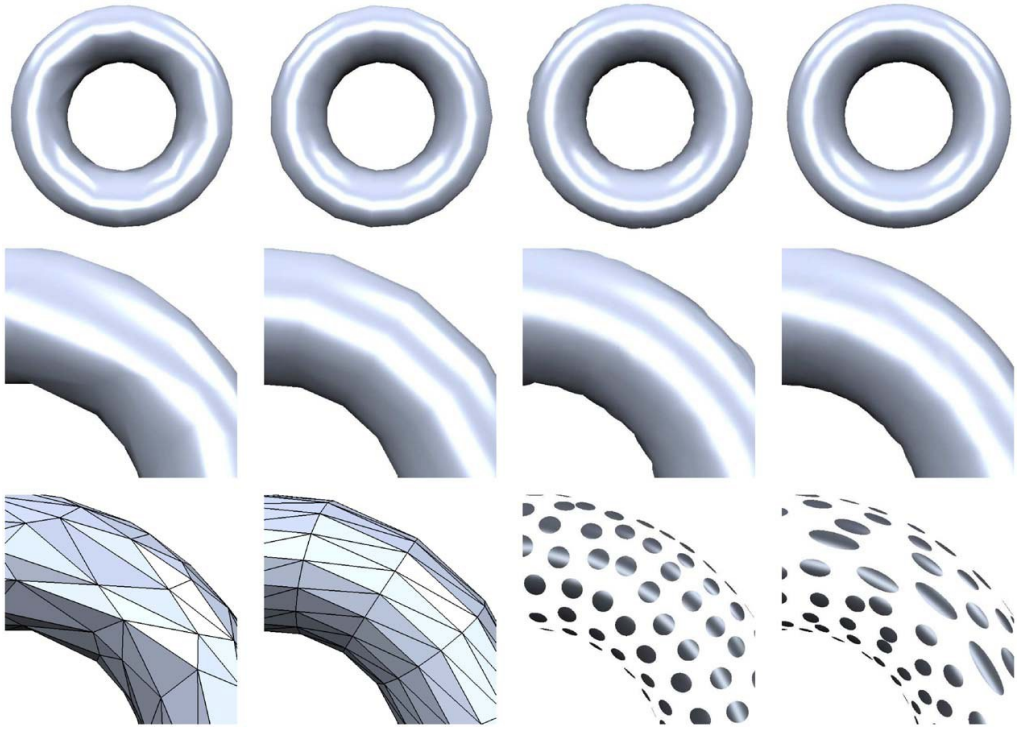


Figure 5: Different shape approximations from left to right: irregular triangles, regular triangles, circular splats, elliptical splats. Image taken from [28]

points or oriented points do not have a 'radius' that can be adapted to the local surface properties. Although, using splats, one can reduce the number of primitive elements significantly in comparison to point based rendering. Figure 6 shows an example using 30.000 splats instead of 350.000 points for the given data set. Some more techniques for the optimization of the splat representation were proposed by Pauly [38] and Kobbelt [46].

A drawback of splatting, as for most point based representations, is that most of the current software and hardware is designed for triangle and mesh based representation, and need to be adapted to the point based splatting approach.

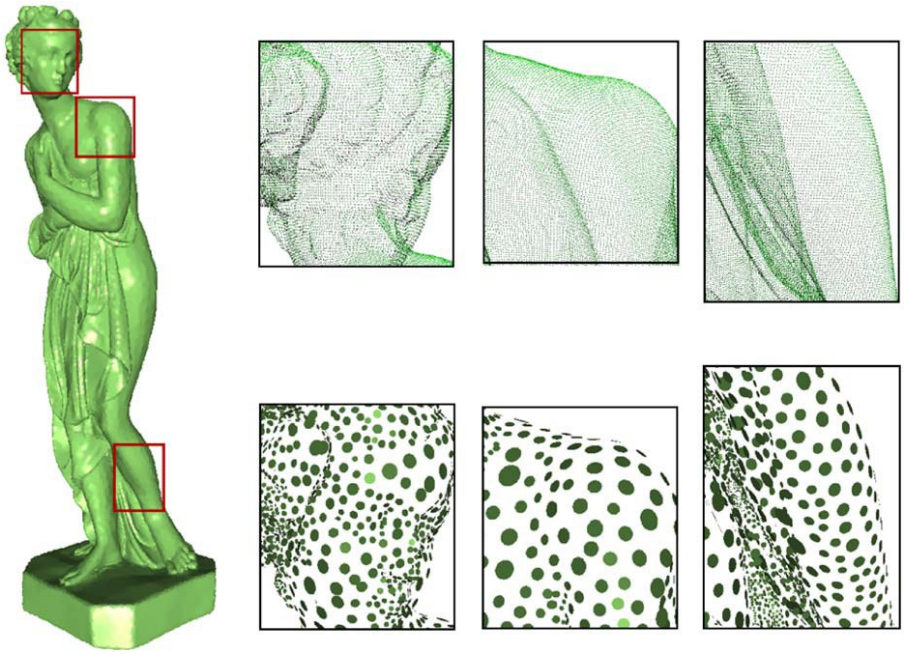


Figure 6: Comparison: splats and point set. Top row: 350k points, bottom row 30k circular splats, Image taken from [28]

2.1.5 *Generation of unorganized point sets*

In this section a short overview about the common generation of point sets will be given. In general, the point sets are produced by a device, scanning the surface and measuring the coordinates of multiple positions on a given surface. This can be achieved in several ways.

MECHANICAL SCANNERS: Scanners based on mechanical measurements usually consist of a needle like probe mounted at the end of a robot arm. During a scanning process, the probe is moved towards the object until the head of the scanning probe touches the surface. The exact position is computed from known properties of the robot arm like length of the arm segments and angles between the arm segments. The system than stores this position in the point set. Those techniques are only reasonable for small numbers of sample positions. Since only one point can be scanned at a single step it is quite time consuming. Although the results are very exact. This kind of measurement devices is for example used in the car manufacturing industries. One uses it for example to control the real positions of fixed features on parts of the car body and check if the produced part meets the

quality requirements. A exact reconstruction of a complete surface is usually not possible due to the relative low number of samples. Mechanical scanners may have problems with the scanning of concave objects, if the robot arm with the probe can not reach the concave elements. But they have no problems with reflective or glass like transparent surfaces like laser based systems.

NON-CONTACT SCANNERS: Scanners that are not 'touching' the surface usually use laser. Most of them are able to measure groups of points simultaneously. Laser scanners project rays of laser light on the surface. A sensor at the head of the scanning device measures the time until the laser is reflected. With this information, the system computes the distance between the laser probe and the surface.
By a projection of laser lines on the surface, laser scanners are able to scan groups of points at one step, making them much faster than mechanical probes. Unfortunately laser scanners have problems dealing with light reflecting or transparent objects like glass. For those object other methods have to be used, for example mechanical probes. As for the mechanical scanners, non-contact scanners may also have problems in scanning concave elements of a surface the laser cannot reach. Modern laser scanners are almost as accurate as mechanical probes.
There are also non contact scanning approaches that use digital cameras in several ways to scan a surface. One is done by the projection of a geometric pattern onto the surface. Through an analysis of the distortion of these pattern in the picture the three dimensional structure of the surface can be recomputed. Another approach to compute 3D coordinates using digital cameras on a surface is done using a whole set of digital cameras watching the scene from different angles. In a combination of the multiple pictures and the known positions of the cameras, it is possible to compute the three dimensional reconstruction of an object. The advantage of this method is that additional color or even texture data for the reconstructed object is available. This is not the case for the methods using patterns or laser.
A drawback of these image based reconstruction methods is that they are often not exact enough for an industrial application. Especially not in manufacturing industries and quality control. Here laser or mechanical scanners are a better choice.

In Chapter 4 we will present a surface reconstruction method based on Moving least squares. MLS is a common approximation method for scattered data. So, let us give a overview over some important scattered data interpolation and approximation methods in this section.
Scattered data interpolation and approximation methods, as the name says, deal with the reconstruction of an unknown function from given scattered data. The application areas are quite widespread. They vary from surface reconstruction, terrain modeling, numerical solutions of partial differential equations, kernel learning, fluid structure interaction to many more. Many of these applications are used in fields like mathematics, computer science, engineering, but also in geologies, biology and business studies.
The general problem in scattered data interpolation or approximation is the following: Given is a set X of $n \in \mathbb{N}$ data sites $X = x_1, x_2, ..., x_n$ and $x_i \in \mathbb{R}^d, (d > 1)$ with corresponding data values $f_i \in \mathbb{R}^{\overline{d}}, (\overline{d} = 1, 2, ...)$ and $f_i = f(x_i), i \in \{1, 2, ...n\}$ in case of interpolation and $f_i \approx f(x_i), i \in \{1, 2, ...n\}$ in case of approximation. The most often encountered cases are $d = 2, 3$ where the data sites are 2 or 3-dimensional, or 3D restricted to a surface and $\overline{d} = 1, 2, 3$. Based on the given data, the task is to find an unknown function f that interpolates or approximates the given data at the data sites.The approximation case is of special interest if the given data contains noise, since approximation allows us to smooth the function and reduce the effects of the noisy data. Let us now review two basic scattered data methods. A complete overview can be found in [17]

2.2.1 *Shepard's Method*

An algorithm known as Shepard's method [43], which is based on inverse distance weighting, was one of the first algorithms in this field. It defines a C^0-continuous interpolation function in form of weighted average of the data. The weights are inverse proportional to the distance. The equation used is as follows:

$$F(x) = \sum_{i=0}^{n} w_i(x) f_i \tag{2.1}$$

where $x \in \mathbb{R}^d$, n is the number of data points in the set, f_i are the given function values at the data points, and w_i are the weight functions assigned to each data point. The classical form of the weight function is:

$$w_i(x) = \frac{\sigma_i(x)}{\sum_{j=1}^{n} \sigma_j(x)} \tag{2.2}$$

where

$$\sigma_i(x) = \frac{1}{d_i(x)^{\mu_i}} \tag{2.3}$$

is a power of the Euclidean distance $d_i(x) = |x - x_i|$. The weighting exponent μ is, in the classical form recommended to be $\mu = 2$ since this eliminates the computation of the root and helps to speed up the process.

The advantages of Shepard's method is that there is no linear system to be solved as each function value can be evaluated as a weighted sum. Being a global method, Shepard's method suffers on problems from influences of points far away from the actual point. Huge data sets are also a big problem as for almost every method using a global approach due to the chance of numerical instabilities. Another disadvantage is, that every weight needs to be recomputed if a single data point is added, removed or modified.
Later Franke and Nielsons [18] modified Shepard's approach overcomes most of these problems. The modified quadric Shepard's method generates C^1-continuous interpolation functions.

2.2.2 *Radial Basis Functions*

Radial Basis Functions (RBF) are another popular scattered data interpolation technique. Their origins lies in the field of the neural network community.
The RBF method uses a set of radially symmetric basis functions. Each of the RBFs is centered at one of the data points x_i.

$$f(x) = \sum_{i=1}^{n} \alpha_i R(d_i(x)) + p_m(x), \quad p_m(x) = \sum_{j=1}^{m} \beta_i p_j(x) \tag{2.4}$$

The basis functions $R(d_i(x))$ are positive radial functions. Usually functions of the distance $d_i(x)$ of the point x to the interpolation point x_i are used. $\{p_j(x)\}$ is a set of

monomials with a maximal degree of m. The unknown coefficients for the basis functions are then computed by solving a linear system of equations, using the interpolation conditions $f(x_i) = f_i$ and the m side conditions

$$\sum_{j=1}^{n} \alpha_j p_i(x_j) = 0,\ i = 1, \cdots, n \tag{2.5}$$

During computation, the function is constrained to be zero at the data points, and non zero on other points, to avoid the construction of the trivial solution. For large data sets, the coefficient matrix becomes very large and often poor conditioned, making RBF difficult to handle for large data sets. A popular choice for the basis functions are Hardy's multiquadrics [23] due to its condition properties and good results. Hardy used rotation symmetric basis functions of the form:

$$R(r_i) = (r_i^2 + R_i^2)^{\frac{\mu_i}{2}},\ \mu_i \neq 0 \tag{2.6}$$

that have no polynomial precision with $m = 0$. R_i and r_i are free to be chosen, but the best choices seem to be $r_i = r(d_i) = d_i(x)$ and $R_i = R$. In this case the basis functions have rotational invariance and translation invariance and the coefficients matrix of the system of equations becomes symmetric.

Carr et al. [9] developed a multipole extension for polyharmonic functions, which overcomes some difficulties and lead to good results. But this solution is very complex and difficult to reproduce. Today RBFs are also used in pattern recognition [27] and statistical learning .

2.3 AN INTRODUCTION TO MOVING LEAST SQUARES

We use moving least squares as primary tool for the surface reconstruction. Moving least squares (MLS) creates an implicit surface. According to the parameters, basis- and weighting-functions used the resulting surface either interpolates or approximates the original point set. To recapture the basics of moving least squares we start with the standard least squares method that is the historical basis of MLS.

2.3.1 *Standard Least Squares*

Least Squares is a mathematical approximation method. It tries to find a function that approximates a series of given measured data points very close ('best fit'). It does this by

minimizing the sum of the squared differences respectively distance between the values of the function and the original data.
Assume to build a function that approximates the values $y_i \in \mathbb{R}^d$ at given points $x_i \in \mathbb{R}^d$, $i \in [1...N]$, $with\ f(x_i) = y_i$. To reach this goal, we first suppose the function to be of a certain form with some parameters, we need to determine. A linear function of the form $f(x) = c_0 + c_1 x$ has the parameters c_0 and c_1. According to the definition of least squares, we have to seek values for c_0 and c_1 that minimizes the sum of the squares of the differences $y_i - f(x_i)$.

$$\min(\sum (y_i - f(x_i))^2) \tag{2.7}$$

2.3.2 *Towards Moving Least Squares*

The standard least squares approach is based on the minimization of a sum of the squares of the distances at all data points. So the solution is the defined over the whole space leading to a global fit. In difference, moving least squares allows the fit to change locally, and also the solution changes with the value of x.
The original MLS approach was presented by Levin ([33],[32]). Over the time many variations of MLS where generated. But overall they can be classified in two categories: projection based MLS surfaces and implicit MLS surfaces.
The projection based surfaces use a two step projection procedure to project a set of points onto the original surface defined by the point set. The implicit surface in contrary is defined by the construction of a zero isosurface of a level set function. In this section we will give a short introduction to the implicit and the projection based forms of MLS. Our approach we present later in Chapter 4, is an variation of the projection based MLS.

2.3.2.1 *Implicit MLS*

Implicit MLS is for example used by Shen et al. [42] and Kolluri [30]. To achieve the 'moving' of the function, the idea is to give the points of the point set being approximated different contribution to the actual local fit. This is done using a distance weight function $w(r)$ for each difference $r = (\|x - x_i\|)$. This leads us to:

$$\min(\sum w(x - x_i)(y_i - f(x_i))^2). \tag{2.8}$$

For a linear fit using a linear basis function, $f(c_0, c_1) = c_0 + c_1 x$, this leads us to

$$\sum_{i=1}^{N} w(x - x_i) \left[y_i - (c_0 + c_1 x_i) \right]^2. \tag{2.9}$$

In the end it leads us to an system of equations

$$c_0 \sum_{i=1}^{N} w(x - x_i) + c_1 \sum_{i=1}^{N} w(x - x_i) x_i = \sum_{i=1}^{N} w(x - x_i) y_i \tag{2.10}$$

and

$$c_0 \sum_{i=1}^{N} w(x - x_i) x_i + c_1 \sum_{i=1}^{N} w(x - x_i) x_i^2 = \sum_{i=1}^{N} w(x - x_i) x_i y_i. \tag{2.11}$$

Transformed in matrix form, the coefficients c_0 and c_1 can be computed by inversion of a matrix. The linear case above has two coefficients and thus requires the inversion of a 2x2 matrix. A quadratic version would need a 3x3 matrix to be inverted. In general, implicit MLS with polynomial basis function of degree $M - 1$ results in a system with M unknown coefficients to solve and thus the inversion of a $(M + 1)x(M + 1)$ matrix to solve.

The Basis functions used are of big importance and have a huge influence on the result. This can be seen in Figures 7 and 8. Usually linear polynomial functions are used.

$$f(x) = c_0 + c_1 x + c_2 x^2 + ... + c_{M-1} x^{M-1} \tag{2.12}$$

This is an example of degree $M - 1$. In a generalized form this one uses:

$$f(x) = \sum_{i=0}^{M-1} c_i b_i(x) \tag{2.13}$$

with functions $b_0(x), b_1(x), ..., b_{M-1}(x)$, the basis functions. Usually quadratic $f(x) = c_0 + c_1 x + c_2 x^2$ or cubic functions $f(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3$ are used. But it is also possible to use a constant function, that means only $f(x) = c_0$ as basis. A linear basis function leads to the behavior shown in Figure 7.

The usage of higher orders for the basis functions is not always a benefit since the solving of the equation system needs is more expensive. A good trade off is the use of

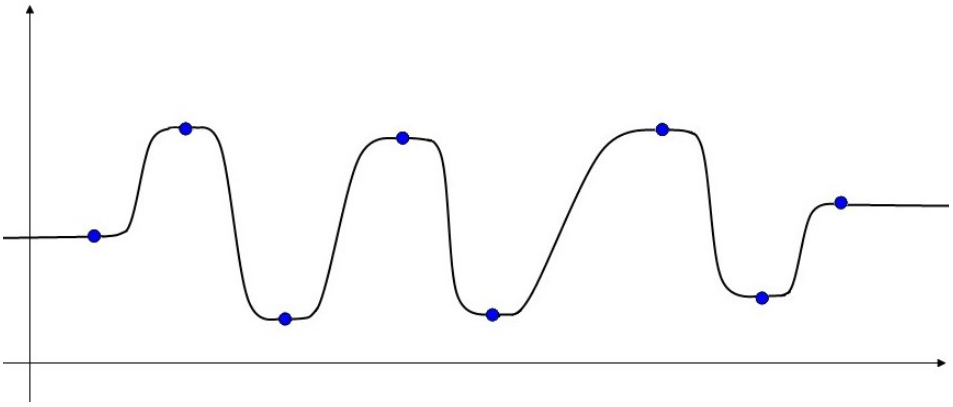


Figure 7: 2D example for an interpolating moving least squares fit with an linear basis function

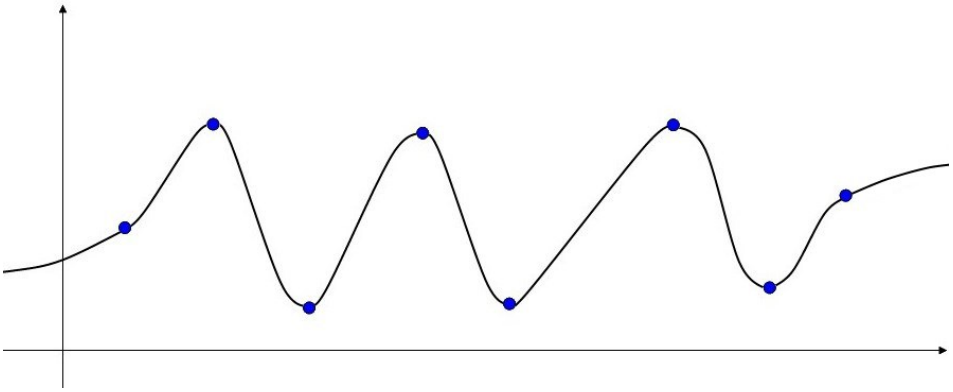


Figure 8: 2D example for an interpolating moving least squares fit with an quadratic basis function

cubic or quadratic basis functions. Figure 8 shows an example of a 2D interpolation with quadratic basis function.

But not only the basis function has an effect on the resulting approximation. The selection of the weighting function has a large influence on the behavior of the resulting fit too. The behavior can vary from interpolation to coarse approximation even with a low order weighting function. Using a weighting function that approaches $+\infty$ at zero will lead to interpolation of the original data.

Functions used as weighting functions often belong to the group of inverse distance functions. One example for such a weighting function is:

$$w(r) = \frac{1}{(r^2 + \epsilon^2)} \tag{2.14}$$

This function provides both interpolating and approximating behavior by adjusting parameter ϵ. An ϵ of zero will lead to an interpolation of the data points since in this situation, the weight w approaches $+\infty$ at the given data points, an non-zero value of ϵ leads to approximational behavior. In this case, a larger ϵ will provide a smoother approximation while a smaller ϵ will giver a closer approximation of the original data. This way, ϵ can be used to smooth the result in noisy data sets. The figures 9 and 10 show 2 dimensional examples for an interpolation and approximating MLS reconstruction.

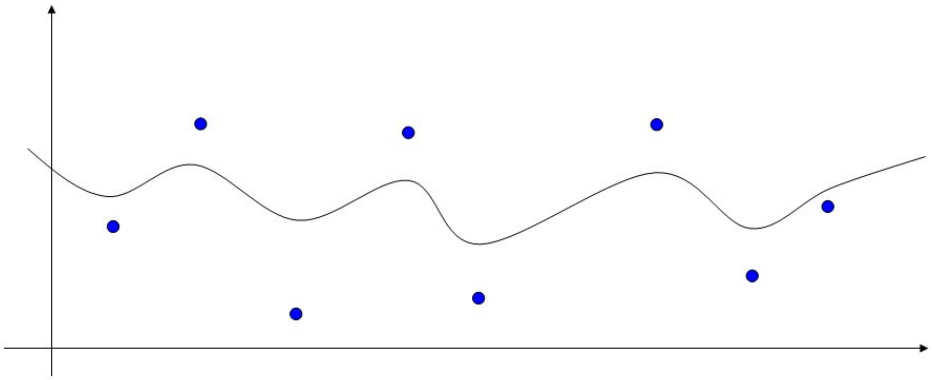


Figure 9: 2D example for a moving least squares fit with an approximating weight function, $\epsilon = 0.5$

An other group of functions used as weighting functions are Gaussian functions like:

$$w(r) = a * \exp^{-r^2/b^2} \tag{2.15}$$

Here the parameters $a > 0$ and b are used for the smoothness of the approximation. Gaussian weight functions do not approach $+\infty$ at zero. So they cannot provide a true interpolation of the data. But in practice a good combination of the parameters can achieve nearly interpolating behavior.

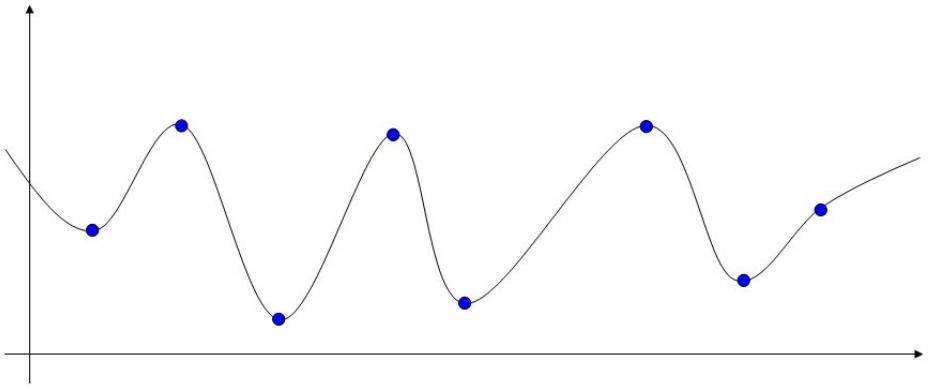


Figure 10: 2D example for a moving least squares fit with an interpolating weight function, $\epsilon = 0$

2.3.2.2 *Projection based MLS*

Levin [32] originally supposed an other way to compute the MLS-surface. He introduces the use of a projection procedure for this step. The main idea is to define a projection procedure in a way that a given point near the point set is projected onto the surface defined by the point set. Alexa [1] shows the projection procedure in detail.
For a point set $p_i \in \mathbb{R}^3,\ i \in \{1, ...N\}$ sampled from the original Surface S one projects a point $r \in \mathbb{R}^3$ near S onto a the Surface S_p defined by the point set. The surface S_p is then locally approximated. The projection procedure itself is divided in two steps. In the first step, one has to find a reference plane for the local polynomial approximation which is then executed in the second step. Figure 11 shows the projection procedure for a single point r projected onto the Surface defined by a set of points.

1. The local reference plane

In this first step a hyperplane $H = \{x | \langle a, x\rangle - D = 0,\ x \in \mathbb{R}^3\},\ a \in \mathbb{R}^3,\ \|a\| = 1$, with origin D and normal a, referring to a point $r \in \mathbb{R}^3$ is computed. H is computed by minimization of the local weighted sum of the squared distances of given points p_i to the plane. The weights belonging to p_i are a function of the distance of p_i to the projection of r onto the hyperplane H. Let q be the projection of r on H. So H is then found by minimizing

$$\sum_{i=1}^{N} (\langle a, p_i\rangle - D)^2 \, w(\|p_i - q\|) \tag{2.16}$$

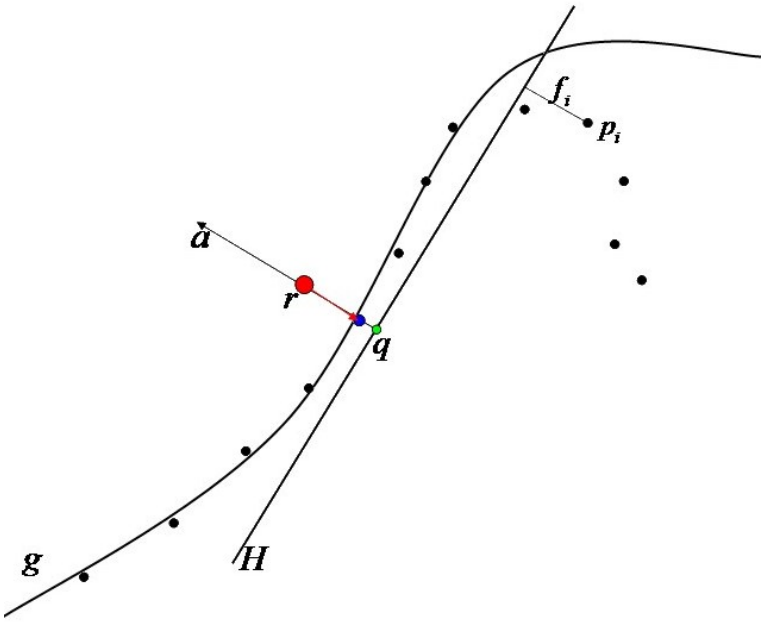


Figure 11: The basis MLS projection procedure: First, a local reference domain H for the red point r is generated. The projection of r onto H defines its origin q (green). After this, the local polynomial approximation g to the heights f_i of points p_i over H is computed. The blue point finally is the projection of r onto g.

with w as a smooth monotone decreasing function in $\mathbb{R}^+$. As w decreases with the growing distance of the points, the local reference plane approximates a tangent plane to S near r.

If we define $q = r + tn$ for a $t \in \mathbb{R}$, 2.16 can be rewritten as

$$\sum_{i=1}^{N} \langle a, p_i - r - tn \rangle^2 \, w\left(\|p_i - r - tn\|\right) \tag{2.17}$$

Operator $Q(r) = q = r + tn$ is defined as local minimum of 2.17 with smallest t and the local tangent plane H near r accordingly. q can than be used as origin of a local orthogonal coordinate system on H that forms the local reference domain.

2. The MLS projection

The local coordinate system on H with origin q from step1 is now used to compute a

local polynomial approximation of the surface in the neighborhood of r. The coefficients of the polynomial approximation are computed by minimizing

$$\sum_{i=1}^{N} (g(x_i, y_i) - f_i)^2 \, w(\|p_i - q\|) \tag{2.18}$$

In this formula (x_i, y_i) are the representation of q_i in the local coordinate system, while q_i is the projection of p_i on H, and f_i the height of p_i over H. Once more the distances are used for the weighting function.

In Chapter 4 we modify this projection based approach and introduce a surface reconstruction method that does not smooth the sharp features in the data set.

2.4 INTRODUCTION TO NURBS

In Chapter 5 we will show a method to combine and blend the point based moving least squares surfaces with algebraic defined surfaces. We used the popular NURBS surfaces in our method. Therefore let us give a short introduction into the basics of algebraic and especially NURBS surfaces at this point.
Being developed in the 1950s and 1960s in the beginnings of CAD in airplane and car design, these kind of curve and surface representations are today well known and established. One useful overview of these classical curve and surface methods in CAD is for example given in [6]. NURBS are a well known and popular approach in computer graphic and CAD based on parametric functions. For the definition of curves or surfaces it is common to use sets of parametric functions. That means, one can use a set of functions and one or more parameters to define a curve. A more detailed definition of parametric function can be found in [12]. Thus there exists plenty of possible definitions of curves or surfaces. For example, the coordinates of the points on an curve can be given as a set of polynomial functions:

$$x = X(t), \; y = Y(t), \; z = Z(t) \tag{2.19}$$

using polynomial functions X, Y, Z and a parameter t. A simple example for parametric functions is the definition of a circle in the 2 dimensional case. Here one can use two functions for the coordinates (x, y) of the curve with $x = X(t)$, $y = Y(t)$ and

$$\begin{aligned} X(t) &= r\cos(t) \\ Y(t) &= r\sin(t) \end{aligned} \tag{2.20}$$

with $t \in [0, 2\pi]$.
Over time many different approaches where developed, e.g. Bézier curves, Hermite curves or B-splines, to name a few. See Farin [15] for more details. NURBS-surfaces that we will use are based on Bézier curves and B-splines. Bézier curves, although being very popular due to their simplicity, suffer some limitations that B-splines can overcome. The most important are the non-local character of Bézier curves and the relationship between degree of the curve and the number of control points. In the global concept of Bézier curves each polynomial coefficient depends on every control point of the curve. This leads to the problem that the modification of a single control point changes the whole curve. Using B-Splines eliminates this problem. B-Splines consist of several spline segments, each defined by a reduced subset of control points. Changing a control point thus will only have an effect on the associated segment. B-splines also allow the generated curve to interpolate the endpoints of the control polygon.
A B-Spline curve of degree n is defined by a set of $m+1$ control points $\{P_0, P_1, ..., P_m\}, P_i \in \mathbb{R}^3$ $m \geqslant n$ consists of $m-(n-1)$ curve segments. Each segment is only affected by n control points. This means, changing one of the control points will only have an effect on the associated curve segment and not the global curve. This makes the design and manipulation of objects a lot more comfortable than using standard Bézier curves. In addition, a B-spline has a set of q knots, also called 'knot vector' that controls the distribution of the parameters for the segments in the interval $[0, 1[$. The number of knots related to the degree and the number of control points, and equals the number of control points plus the degree of the curve plus one $q = n + m + 1$. If the knots are distributes uniformly within this interval, the B-spline is said to be uniform, otherwise it is non-uniform. With non-uniform B-splines one can use the knot vector and a multiple use of single knots to interpolate for example the endpoints of the control polygon. If the first n knots of a curve are equal and the last n knots are equal, the curve will interpolate the two end control points of the curve. If one assumes the coordinates of a point (x, y, z) to be rational polynomials, like

$$x = \frac{X(t)}{W(t)},\ y = \frac{Y(t)}{W(t)},\ z = \frac{Z(t)}{W(t)} \tag{2.21}$$

with $W(t)$ interpreted as a weight added to the according control point. Such a B-spline is said to be rational, else it is non-rational.

Overall, this leads to 4 different types of B-Splines:

- Uniform Non-Rational B-Splines

- Non-uniform Non-Rational B-Splines
- Uniform Rational B-Splines
- Non-uniform Rational B-Splines

the last mentioned type is the NURBS. They are the most general case and thus probably the most powerful and most widely used B-Spline type, especially in CAD applications.

Mathematically a NURBS curve $Q(t)$ of degree n is defined as

$$Q(t) = \frac{\sum_{i=0}^{p} B_{i,n}(t) P_i w_i}{\sum_{i=0}^{p} B_{i,n}(t) w_i {}_{i=0}^{p}} \tag{2.22}$$

where $P_i \in \mathbb{R}^3$ are the control points, $w_i \in \mathbb{R}$ the associated weight, and $B_{i,n}$ a basis function defined by

$$\begin{aligned} B_{i,0}(t) &= \begin{cases} 1 \text{ if } t_i \leqslant t < t_i + 1 \\ 0 \text{ else} \end{cases} \\ \forall k > 0,\ B_{i,k}(t) &= \frac{t - t_i}{t_{i+k} - t_i} B_{i.k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(t). \end{aligned} \tag{2.23}$$

A NURBS-surface is constructed analog to a NURBS-curve, but with the use of a control mesh and two parameters u and v. Mathematically three dimensional parametric surfaces are generated from the Tensor product of two curves, thus we have the two parameters u and v. A NURBS surface $S(u, v)$ is thus defined by:

$$S(u, v) = \frac{\sum_{i=0}^{p} \sum_{j=0}^{q} B_{i,m}(u) B_{j,n}(v) P_{i,j} w_{i,j}}{\sum_{i=0}^{p} \sum_{j=0}^{q} B_{i,m}(u) B_{j,n}(v) w_{i,j}} \tag{2.24}$$

Some of their advantages and reasons why they are used widely are as follows.

- They offer a precise mathematical representation of free-form shapes and analytical shapes (like cones, spheres, etc.)
- The manipulation of the control points leads to a wide flexibility and variety of shapes

- NURBS are invariant under scaling, rotation and translation as well as perspective transformations
- It is easy to construct a NURBS with a desired continuity (C_0, C_1, C_2)

However, NURBS also have drawbacks. Some are mentioned by Piegl in [40]:

- They need extra storage to define some often used curves or surfaces. For example, the construction of a full circle needs seven control points and 10 knots. A traditional representation would need only the center, the radius and the normal vector to the plane of the circle.
- Some techniques, for example surface/surface intersection work better with traditional forms.
- Some useful algorithms get problems with numerical instabilities.
- A bad setting of the weights of control points can lead to a bad parameterization, which can lead to additional problems in subsequent surface constructions.

Part III

DETECTION OF SHARP FEATURES IN POINT CLOUDS

3

DETECTION OF SHARP FEATURES IN POINT CLOUDS

3.1 ABSTRACT

In this chapter, we are going to present a new method to identify sharp features in a simple point cloud data set without normal information.
Sharp features are of big importance especially in the manufacturing industries. During reverse engineering of mechanical components, for example for the evaluation of the manufacturing process, the possible loss of features like sharp edges is a problem to deal with. There are already existing concepts and algorithms to identify features in point clouds. But many concepts are based on creating a triangulation of the point cloud first and then searching for the features in this triangulation. Creating a triangulation first is a possible way. But the triangulation step requires additional computational costs and is itself not trivial.
Also, during the construction of the triangulation the sharp feature can be lost. So we decided to avoid these ways and worked out a method that searches and finds the interesting features only by using the information we have in our point cloud data. As input we use a simple set 3D point coordinates without further information about the surface (normals, neighbors).
During our feature detection we analyze the point cloud. We iterate through the point cloud and decide for every single point, if it belongs to a sharp feature or not. The result is an attributed point cloud showing the position of sharp features.

After the first promising results with the approach we refine the method and replace a global dataset-dependent parameter for sensitivity by a local one. This improves the results significantly as we will show later. The local parameter we are going to compute will be adapted iteratively to a local region depending on the characteristics of the region. This leads to a higher degree of automation of the feature detection and in addition improves the results in complex geometries. The following pictures in Figure 13 show the improvement through the adaptive parameter. The left side pictures show the results of the first detection of feature candidates, the right pictures show the candidates after the refinement procedure.

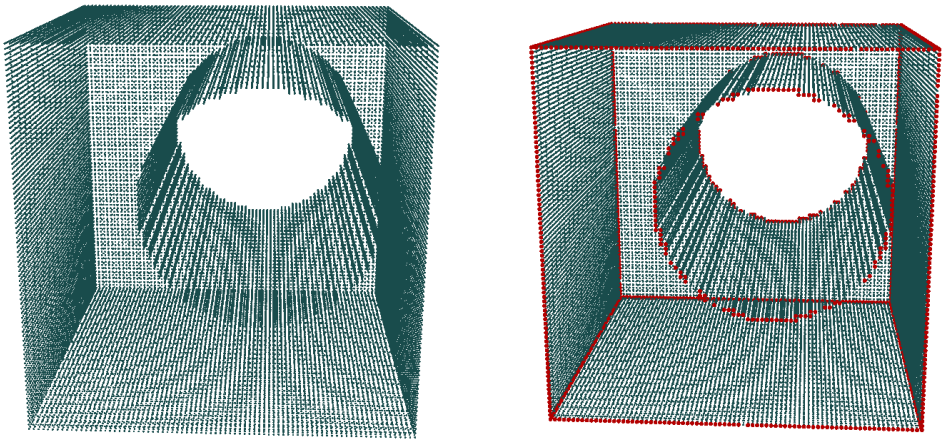


Figure 12: Tagged Point cloud: the left side of this figure shows the original point cloud data; The right shows the tagged data.

3.2 PROBLEM DESCRIPTION

Over the last years scanning technologies have become more and more affordable and accurate. This increased the availability and usage of these techniques in industry. The usage of scanning devices covers a wide broadband. During development process of a new product it can be used to approve and optimize the production process or to digitize manually designed prototypes. During production it might be used to support the quality control. In the car manufacturing industries the wear of a production tool can be measured by controlling the position of some fix points on the product. Today the number of control points used is relative small and only covers critical sections of the product. No complete reconstruction of the objects is used for the quality analysis. But with the development of faster scanning devices and reconstruction techniques also the reconstruction in detail or of the complete product might get useful for quality analysis. Common scanning devices usually collect data in form of sampling points either by a mechanical probe or by laser, see more details in Chapter 2 Section 2.1.5. In consequence the importance of processing these point clouds has increased.
For modeling applications or quality measurements the preservation of sharp features is of great concern. Even for different fields of computer graphics like simplification, smoothing or visualization the knowledge about the positions of sharp features can be of great help. There exist different approaches to reconstruct a surface from point cloud

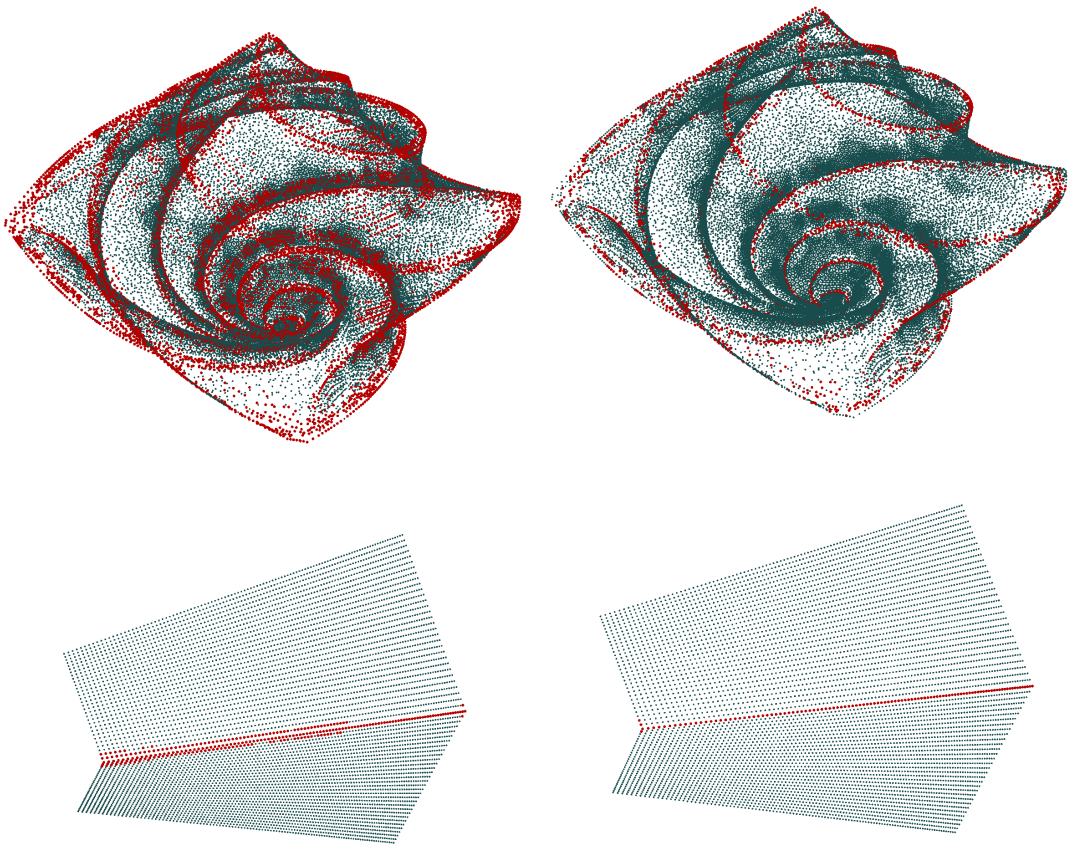


Figure 13: The left side shows examples without the adaption method; the right side examples use the adaptive method

data. But many of these techniques lead to a loss of detail and foremost a loss of sharp edges and features due to a smooth (C^2-continuous) reconstruction. Techniques allowing sharp features on the other side often don't identify sharp features automatically, and the user has to position or mark the sharp feature manually to reconstruct it properly. Since a user controlled operation is not always wanted, an automatic identification of sharp features in a point cloud is an interesting and important step to get better reconstructions of a given surface.

Many existing detection methods for sharp features or edges are based on a triangulation of the point cloud. We achieved to use only the point coordinates to detect the regions with sharp features in the point cloud. One usual approach is to use the Delaunay-triangulation and its triangle normals. Other approaches use a Neighboring-Graph based on the Riemannian graph. This graph contains the edges to the k-nearest neighbors of every data point. Then the analysis of the neighborhood of a given point delivers the probability of being a feature.
We differentiate between three groups of feature detection methods. Mesh based methods, point based methods and reconstruction based methods that detect sharp features during a surface reconstruction. In the next sections these methods will be shown in more detail.

3.3.1 *Mesh based feature detection*

There are multiple existing techniques for feature extraction, relying on polygonal meshes [26, 44, 29, 25, 45]. The following methods represent a group of different approaches. The list of methods is not intended to be exhaustive.
Hubeli and Gross [26] use a normal based multi-resolution framework and generate a set of edges with a normal-based classification operator. In a classification phase they assign a weight to every edge in the input mesh, proportional to the probability of belonging to a feature. The authors provide different types of operators for different mesh types like a 'second order difference' operator for very coarse data sets an 'extended second order difference' operator for finer meshes or a computational more expensive 'best for polynomial' operator which performs well on noisy points sets. After this, in a detection phase they reconstruct the features from the information gained in the classification phase. According to the weights, they produce piecewise linear curves from the collections of edges that are assumed to belong to a feature. For thresholding they use a hysteresis thresholding. An edge is added as feature if its weight is larger than an upper bound. If the weight is smaller but still larger than a lower bound the edge is also accepted if a neighboring edge is already selected as feature. All other edges are discarded as features. A thinning process then refines the edges to generate clear feature lines. For the thinning all patch-boundary edges are first inserted to a linked list. A first condition removes edges that are perpendicular to the mesh feature. A second condition makes sure, that an edge is only removed if the patch will not become disconnected. If an edge

is removed, new edges are inserted into the list and have to be analyzed. The process continues until the list is empty. They also present a multi resolution approach for their feature extraction to improve the quality further. The process is not fully automatic since a user has to choose the classification operator and some parameters for the detection phase.
Hildebrand et al. [25] use anisotropic filtering on third order derivatives of the surface mesh. The derivatives are approximated by discrete differential geometric approximations. This way, the authors compute discrete extremalities, which are then smoothed and used to trace feature lines in regular triangles. Singular triangles need a special treatment. In this case the adjacent triangles are used to determine the feature line intersections with the singular triangle. After the first feature line extraction a threshold filter is used to improve the stability of the feature extraction and to remove small ridges. The last step is an optional smoothing of the feature lines. Both methods [26, 25] use extreme triangles to build a set of sharp feature edges.
Watanabe and Belyaev [44] use the so called focal surfaces to detect extreme values of curvature on dense triangle meshes. If k_{max} and k_{min} are the largest and smallest principle curvature then the principle centers of curvature are points situated at the surface normal with a distance of $1/k_{max}$ and $1/k_{min}$ from the surface. These principle centers form the focal surface. It consists of two sheets, one for the minimal, and one for the maximal principal curvature. Watanabe now uses the property, that the singularities of the focal surfaces, called focal ribs, correspond to lines on the original surface where the principal curvature has an extreme value. They present a method for the estimation of the principle curvature on a dense triangle mesh and then show how the associated focal ribs can be used to identify the features. Also here a thinning process of the first results is necessary to construct a final feature line. The resulting lines show the regions of maximal curvature. The method is not specialized for the detection of sharp features.

All mesh based techniques use in the one or another way the connectivity information and normals associated with the underlying mesh. But often surface scanning devices do not deliver a mesh as raw data, but an unsorted set of point data representing the original surface. In this case, a mesh based method has to rely on the proper reconstruction of the features during the mesh generation. So, in order to use these methods one first needs a surface reconstruction that has preserved the sharp features. Since it is our goal to use the feature detection later in Chapter 4 for a surface reconstruction with sharp features these techniques are not of interest in our case. In the end, the negative point of

all mesh based feature detection methods is the fact that they all depend on the feature preserving abilities of the used mesh generation method.

3.3.2 *Point based feature detection*

Very few feature detection methods are dedicated to point-sampled geometry only. The major problem of these point based methods is the lack of knowledge concerning normal and connectivity information. This makes feature detection a more challenging task than in mesh based methods. The usual approach begins with the construction of a local neighborhood of a potential feature point. The different techniques than mostly differ in the analysis of these local neighborhoods.
Gumhold et al. [21] present a method that uses the Riemannian tree to build the connectivity information in the point cloud. The Riemannian tree contains all edges to the k nearest neighbors for every data point. The algorithm first analyzes the neighborhood of each point via a principal component analysis (PCA). The eigenvalues of the correlation matrix are then used to determine a probability of a point that belongs to a feature. The analysis of the ellipsoid formed by the three eigenvectors as basis vectors and their eigenvalues as associated length allows further conclusions about the underlying feature type. This way the algorithm can differentiate between line-type features, border and corner points. The result is a quite dense set of points covering all kinds of features independent if the feature is sharp or not. This set of points is then reduced by computing a minimal spanning tree followed by a branch cutting. This is an elegant way to get a sparse set of points representing the feature line.
Pauly et al. [39] extended the PCA approach with a multi scale analysis of the neighborhoods. Based on the eigenvalue analysis of the covariance matrix they compute a value for the surface variation in the local area around a sample point. For the multi scaling, they vary the size of these neighborhoods to receive additional information. A jump in the graph of the surface variation during the multi scaling shows the existence of new surface parts. Especially in noisy datasets, the multi scaling approach enhances the result of the usual PCA analysis. But since the method analyzes up to 200 neighborhood sizes for each point in the dataset, it is computationally more expensive. To handle these relative huge neighborhood sizes of over 200 neighbors, they also show a way to solve the problem of neighbors not belonging to the same connected region. To estimate that a neighborhood becomes too large and starts to include a near but unwanted regions, for example in close curves, they use a heuristic that looks for strong deviations in the

normal direction. The algorithm recognizes all kinds of visual eminent features including sharp features as well as most smooth features. But for the identification of only the sharp features inside the dataset, this method has to be modified. A way to adapt this approach to sharp features may be achieved with an adjustment of the thresholds used for the feature recognition. With well chosen lower and upper thresholds the method may be able to identify only sharp features. But this can lead to problems in case of sharp features with varying angle. This case would most likely require independent thresholds for each kind of sharp feature.
Demarsin et al. [10] also searched for sharp features in point cloud data. In contrast to our work their goal is to produce closed sharp feature lines. They choose a region growing method, to segment the point cloud into clusters and identify the regions of sharp features. Based on the analysis of the normals of the points, they segment the point cloud in clusters with equal normal behavior. From this clusters they build up a graph that connects the neighboring clusters. The edges in this graph are then used as indication for the existence of a sharp feature in the related area. The resulting set of points is a coarse representation of the area where the features are located. Similar to Gumhold and Pauly, they use a graph approach and construct a minimum spanning tree of these candidates. This gives them an initial reconstruction for the feature lines. A fixed parameter for the maximum branch length is then used to cut off the short branches of the tree. In the next step they close the feature lines. For each open endpoint in their graph they compute the n nearest neighbors among the other endpoints. The distance and the length of the paths of the neighboring endpoints is used to determine a good connection. After this they cut off the branches of the possibly remaining endpoints and smooth the graph to get their final closed feature lines.
Merigot et al. [35] estimate principal curvatures and normal directions of the underlying surface from a point cloud using a so-called Voronoï covariance measure and provide valuable theoretical guarantees. A convolved covariance matrix is computed of a union of Voronoï cells and returns principal curvatures and principal directions. It can be applied to feature detection in discrete data via a proposed algorithm that iteratively computes covariance matrices with varying neighborhoods. Similar to other PCA-based algorithms, the estimated features form a large band of points near the feature line.

The mentioned techniques for a feature detection in point clouds are mostly used as a preprocessing step for another processing step, e.g. a surface reconstruction with sharp features.

3.3.3 *Reconstruction based methods*

Several surface reconstruction methods aim to preserve sharp features during the constructing of a mesh from an unorganized point cloud.
Amenta et al. [4] reconstruct creases, corners and sharp features in a post processing step, if the user indicates that the model contains sharp edges. If not, the method smoothes the corner. They discard regions with sharp features, generating holes in the reconstruction. These holes are then closed by extending smooth surfaces linearly into the empty region until they meet. This results in a sharp angle.
Guy and Medioni [22] extract surfaces, feature lines and feature junctions by discretizing the space into a volume grid. After this, they compute surface votes from the data points for each of the cells. Saliency functions based on an eigenvalue analysis of the votes and the use of an adapted marching cubes algorithm allows the computation of the features.

There are also MLS-surface reconstruction methods that are trying to preserve sharp features in point clouds without a preprocessing step.
Fleischmann et al. [16] use robust statistics to identify sharp features and reconstruct a piecewise smooth surface. They are searching for outliers that are interpreted as sample points on another smooth part of the final surface. This way they classify regions of the point set as outlier free smooth regions. The identification of the sharp features is then done on an iterative refining process. Starting with a fit for a small subset of the point set, they increase the number of points for this fit, until the statistical analysis discovers an outlier. The fitting of the first surface part stops and a new fit on the remaining points starts. Thus this method generates a set of piecewise smooth surfaces. However, this method has problems with dense sampling and jagged edges.
Öztireli et al. [37] use kernel regression to extend the moving least squares surface reconstruction with sharp features. Their method increases the presentation of sharp features conserving the C^2-continuity of the MLS-surface. In some applications however it would be better to have a real sharp feature with a C^0-continuous surface.

3.3.4 *Our method*

In our method in contrary to most other methods, we try to use as less information as possible. That means, that the point cloud data we use only needs to contain the

coordinates of the point data without any additional information or triangulation. We only work on the local neighborhoods of the points. In the neighborhood of a point, we analyze the behavior of a sample point using a mapping onto the Gaussian sphere. We will show the details in the next section. Furthermore, we do not use any additional global and complex structure created from the dataset that might be hard to generate for large datasets. The only additional structure we use is a kd-tree as data structure for the point cloud that allows an efficient nearest neighbor search.

3.4 FEATURE EXTRACTION

In this section we present the details of our feature extraction method. It consists of three steps: We first build the data structure for our points set. In the second step we create local neighborhoods in the point set. In the third step we analyze these neighborhoods to identify sharp features. Let us first define *point cloud* and *sharp feature*.
A point cloud is a simple set of 3D point coordinates without any normal or connectivity information. The data points are unstructured, but supposed to belong to a 2-manifold surface. the point set P is defined as

$$P = \{P_1, P_2, ..., P_N\}, P_i \in \mathbb{R}^3.$$

Let $N = |P|$ be the cardinal of the point set.
A sharp feature we want to detect in the point cloud can be of different nature. For example, a sharp feature can be an edge between two surfaces or a corner where three or more surfaces meet.

3.4.1 *Data structure*

To detect the sharp features in this point cloud, we are going to determine for every point in the point cloud if it is part of sharp feature or not. Like other techniques, we analyze the neighborhood of each single point, to decide if it belongs to a sharp feature or not. As neighborhood we use the *k-nearest*. That means we take the k points in the point cloud with the shortest distance to the sample point. In a large, unsorted and unstructured point cloud even this step, the construction of the k-neighborhood, can be quite time consuming.
Let us first study the problem of searching the nearest neighbor before solving the problem of searching the k-nearest neighborhood. The neighbor search, especially in

a large dataset, depends much on the underlying data structure used. That's why we have to define an appropriate data structure first. A simple list based data structure for example is not a good choice since It would be necessary to check every single point in the list and its distance to the sample point. This will result in huge computational costs of $O(N)$ for a single point which results in $O(N^2)$ for the whole data set.
So it is important to reduce the number of necessary data accesses and comparisons. A space partitioning data structure is asymptotically more efficient in case of large 3D-datasets. These structures divide the data space in subsets. In this way huge groups of possible candidates get eliminated by every single comparison. Usually these data structures are tree structures. Most common are oct-, quad- and binary-trees, dividing the space in eight, four and two subspaces at every level. Oct- and quad-trees are of the same type, but apply to different dimensions. The quad-tree is usually used in 2D, whereas the oct-tree is used in 3D environments.

We finally choose the more flexible *kd-tree* [36] as underlying data structure to do an efficient k-nearest neighborhood search.
The kd-tree is a special case of the binary space partitioning tree (BSP-tree). Every node in the kd-tree is a k-dimensional point (in our case we work in three dimensions)[1]. The non-leaf nodes are used to create splitting planes which divide the space into two subspaces. The sub trees associated with the node now represent the subspaces on the left and right side of the splitting plane. In this way, the dataset is sorted with respect to spatial dimensions.
This allows an efficient search for the k-nearest neighborhood of a point. Since the algorithm of searching the single nearest neighbor can be easily upgraded to searching the k-nearest neighbor we will give a short overview of the single nearest neighbor search for a 3D-point $\boldsymbol{p}$. A useful feature of the kd-tree is that it is not essentially, that $\boldsymbol{p}$ is in the point set of the kd-tree.

SEARCHING THE NEAREST NEIGHBOR IN A KD TREE: To search the nearest neighbor of a point p, the method starts at the root of the tree. One moves down the left or right subtree, whether the splitting dimension of the actual level of p is bigger or smaller than that of the current node. Until now it is the same approach like inserting a new point to the kd-tree. Once the leaf-level is reached, the leaf node

1 please keep in mind, that the k of the kd-tree and the k of the neighborhood are unrelated, we use k in booth situations due to the historical naming of the methods

is used as first candidate for the nearest neighbor. After this the algorithm moves back and up the tree again. The next steps are performed at each node it traverses. If the current node is closer to p than the candidate, it becomes the new candidate. A hypersphere around the search node is than used to check if a point on the other side of the splitting plane might be closer than the current one. The radius of the hypersphere is the distance of the current candidate point to p. To determine if a point on the other side of the plane might be nearer it is enough to check if the hypersphere crosses the splitting plane. In this case the algorithm has to move down the other subtree and check it for nearer points as done before. If it does not intersect the splitting plane the algorithm continues moving up to the next branch. Thereby the whole subset of the point cloud in the branch not taken can be ignored as candidate without further testing. When the algorithm reaches the root node the remaining candidate is identified as the nearest neighbor of p.

Lee [31] has analyzed the worst case for a single nearest neighbor search in a kd-tree containing N nodes.

$$t_{worst} = O\left(k * N^{1-\frac{1}{k}}\right), \text{ with } k = 3 \text{ in our case.} \tag{3.1}$$

This leads to a bad behavior, if the number of points in the tree is only slightly higher than the number of dimensions. In our case with only three dimensions and $N >> 3$ the algorithm will not run into this problems and has a worst case of $O\left(3N^{\frac{1}{3}}\right)$ that is much better than the $O(N)$ of a list structure. Finding the nearest point in the case of randomly distributed points in average leads to $O(\log N)$.

It is simple to upgrade the algorithm for searching the nearest neighbor of a point to searching the *k-nearest neighbors*, by just maintaining the k current best candidates instead of just the last candidate. In this case, the radius of the hypersphere checking the neighboring branches has to be the distance of the worst of the current k candidates. Until k candidates are found, the distance is infinity.

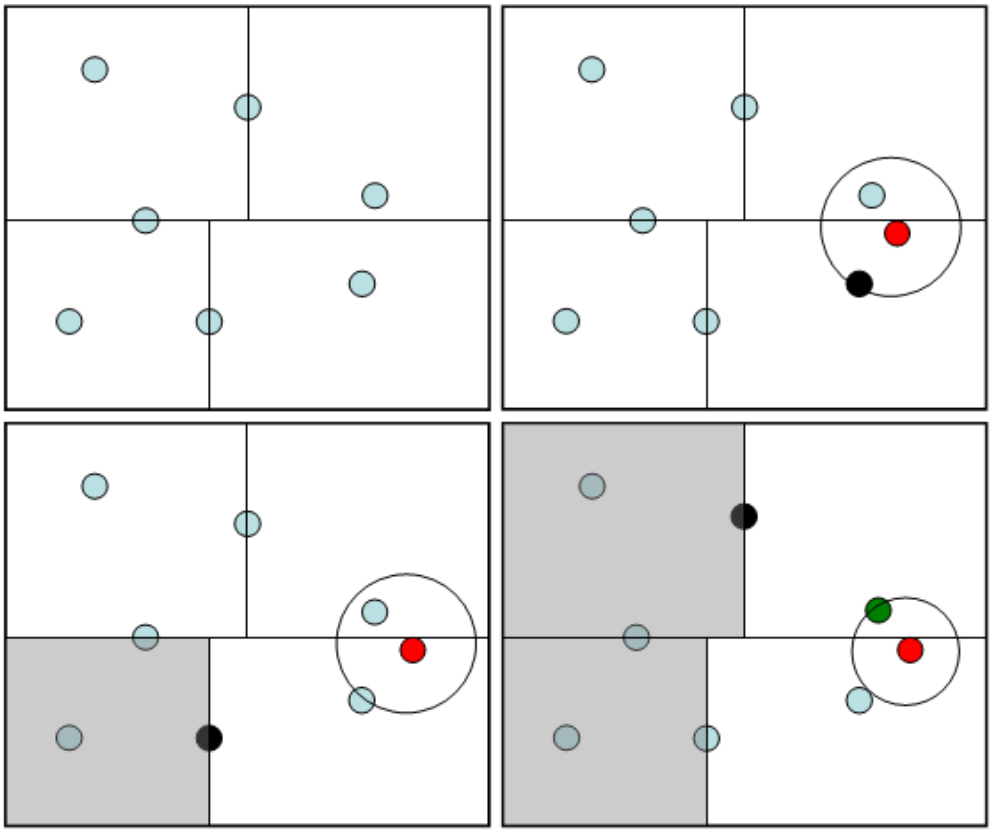


Figure 14: kd-tree nearest neighbor search: The upper left image shows the kd-tree at the beginning of a nearest neighbor search for a new point; The upper right image shows the new point *(red)* and its first nearest neighbor candidate *(black)* after moving down to the leaf level of the tree;
In the lower left image the parent of the current candidate *(black)* is shown. Since the sphere does not intersect the corresponding splitting plane, the other side can be ignored *(grey area)* and the algorithm moves up to the next branch;
In the lower right image the next parent is shown, another branch of the tree is eliminated, but the other side is intersected by the sphere, tested and a new currently best candidate is found *(green)*

3.4.2 *Neighborhood analysis*

Having the local neighborhood N_p constructed for a point $p \in P$, we now want to analyze it to decide if the sample point belongs to a sharp feature or not. To do this we apply a Gauss map clustering.

3.4.3 *Discrete Gauss map*

Let N_p be the neighborhood of p containing the k nearest neighbors, and I_p the index set of N_p. Let T be the set of all possible $k \cdot (k-1)$ triangulations of p with its neighborhood points

$$T = \{ \quad \Delta_{ij} = \Delta(p, p_i, p_j) \quad | \quad i \neq j, \quad i, j \in I_p \}.$$

The normal vector of the triangle Δ_{ij} is given by

$$n_{ij} = \overline{pp_i} \times \overline{pp_j}. \tag{3.2}$$

Note that $n_{ij} = -n_{ji}$.

The discrete *Gauss map* of the neighborhood of p can now be defined as the mapping of T onto the unit sphere S^2 centered at p as follows

$$\begin{aligned} G_p \quad &: T \to S^2 \\ \Delta_{ij} \quad &\mapsto x_{ij} := p + \tfrac{n_{ij}}{\|n_{ij}\|}. \end{aligned} \tag{3.3}$$

Feature detection is now performed in two steps. A first step discards all points belonging to a planar region with a simple flatness test. The remaining feature candidate points undergo than in a second step a more precise selection process, called Gauss map clustering. Gauss map clustering is usually used for other applications, such as segmenting a point sampled geometry into connected regions, grouping together points with same local curvature behavior [34].

FLATNESS TEST

These normal vectors from 3.2 allow us to draw some conclusions about the local surface behavior of the point cloud at p. If the lines passing through p and spanned by n_{ij} with $i < j$ are all almost parallel, then the underlying surface is nearly flat at p. Note that n_{ij} and n_{ji} span the same line. To compare them, we compute the angle between these lines.

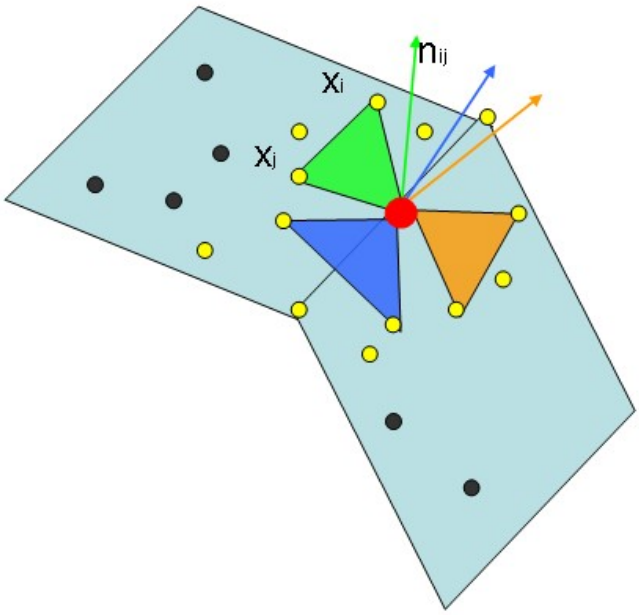


Figure 15: Computation of normal vectors used for feature identification in a local neighborhood

The flatness test consists now in computing the standard deviation of these angles, i.e. we check the distance that means the angle to the median value of the directions. If it is lower than a given threshold (we worked with 15%, which corresponds to an angle of about 13°) the surface in this neighborhood is assumed to be flat or nearly flat without having a sharp feature. But the inverse is not true. A high deviation at p does not imply a sharp feature. It can also appear in a high curvature region without a sharp feature.

GAUSS MAP CLUSTERING

That's why we also analyze p by computing the Gauss map G_p of the set $T = \{\Delta(p, p_i, p_j)\}$. Here we now check the clustering behavior of the normals from 3.2.

This idea is motivated by the fact that in the case of a smooth piecewise C^0 surface the Gauss map of the neighborhood of a surface point is different whether the point is flat, curved (elliptic, hyperbolic or parabolic) or tangent plane discontinuous. In case of a nearly flat point the Gauss map of neighbor points will represent one cluster of points on the sphere. In the case of a curved point (parabolic, hyperbolic, or elliptic) the

points on the sphere will be spread over a larger region. And in the case of a tangent plane discontinuity, the points of the sphere will build two distinct clusters. Figure 16 shows these three cases. A flat area on the left side results in one single, respectively two opposing clusters interpreted as a single one since we know that $n_{ij} = -n_{ji}$. The example with a smooth feature in the middle of Figure 16 shows no clustering at all. In the a case of a sharp feature, one can see that two clearly distinct clusters are produced.

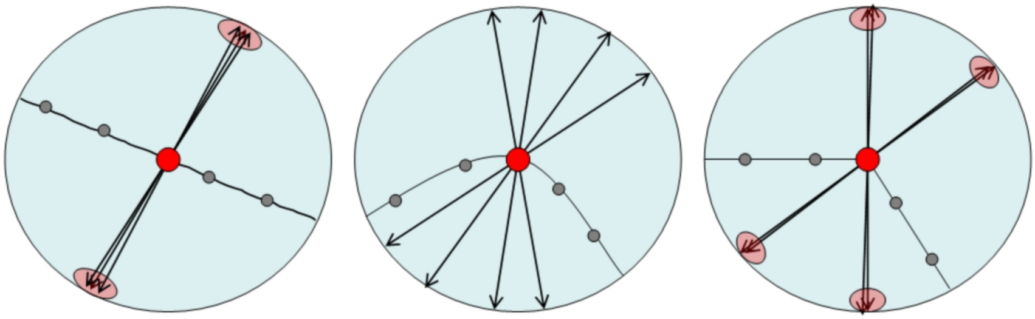


Figure 16: 2D examples for cases during feature detection

In the case of a point-sampled surface, the difficulty is that we don't know anything about the surface nearby. We don't have a local triangulation nor a normal vector associated to the neighbor points. For that reason we defined our Gauss map as the projection of the normals of all possible local triangulations, see Section 3.4.2.

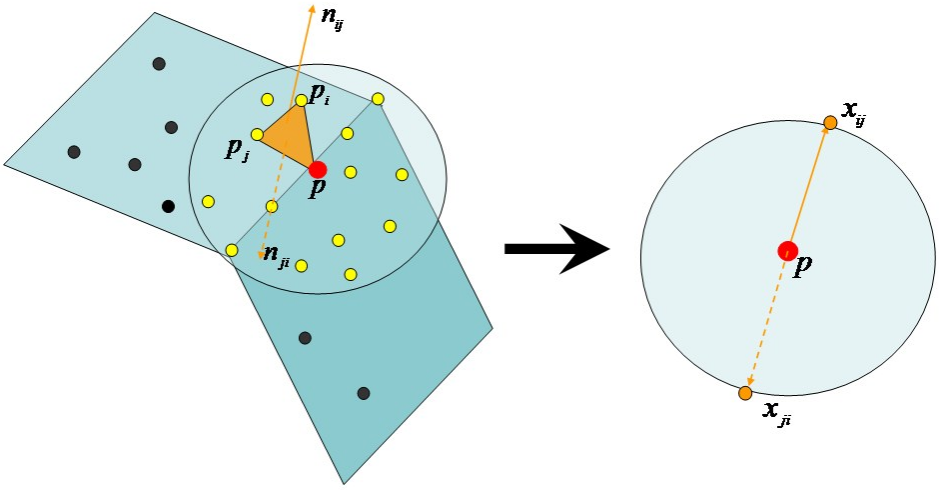


Figure 17: Projection onto the gaussian sphere

Figure 17 shows an exemplary projection of one of the normals n_{ij} onto the gaussian sphere. The figure also shows how the opposing clusters on the gaussian map are generated. The resulting Gauss map of a sharp feature point will thus contain some additional *'noisy' points* which correspond to the triangles that finally don't belong to the underlying surface. These noisy points however don't affect the general clustering behavior. In practice they will disappear when computing the clusters.

Let us illustrate these assumptions for the common case of two intersecting planes with p lying on the sharp edge. Half of the k neighborhood points are lying on each plane, see Figure 18 left. Computing the Gauss map will result in two (opposite) clusters of $O(k^2/4)$ identical points corresponding to the triangles lying entirely in one plane, and two clusters for the other plane. Note that n_{ij} and n_{ji} belong to opposite clusters. All other points on the sphere, the aforementioned 'noisy' points correspond to triangle where p_i belongs to one plane and p_j belongs to the other plane. These points are sparsely distributed over the sphere. In Figure 18 we show a real example implemented with Matlab. On the left, the planes and the 16 neighborhood points are shown. On the right, the Gauss map is shown, where the thick red and blue points represent the clusters.

Similar clustering behavior can be observed for the non-exhaustive list of sharp features whose profiles are shown in Figure 19 and Figure 20.

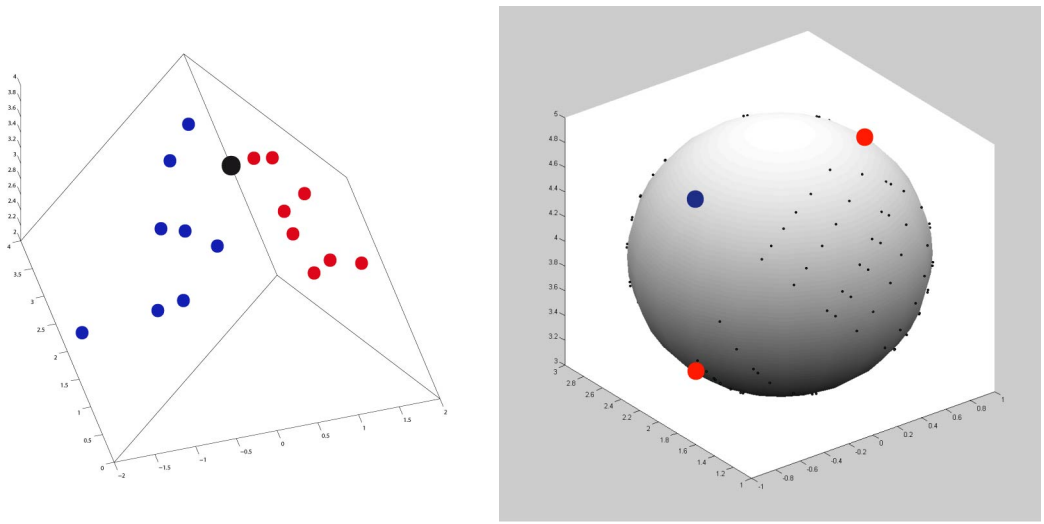


Figure 18: Computation of the Gauss map G_p for feature identification in a local neighborhood.

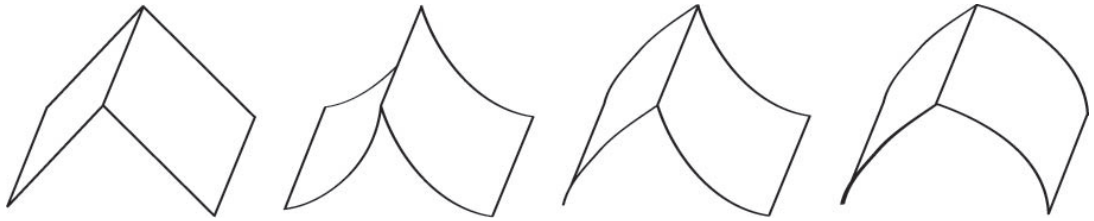


Figure 19: Some example of sharp feature profiles.

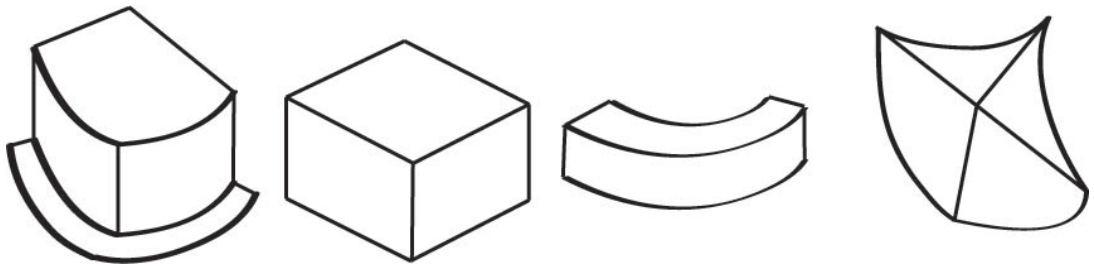


Figure 20: Examples of sharp corners of valence three and four.

We therefore use the clustering behavior of the projected points on the sphere, called Gauss map clustering, in order to determine whether the sample point p belongs to a sharp feature or not.

Regarding the clusters, we have to keep in mind that we can not be sure about the direction of our normals as mentioned above. That means, we don't have any information whether the normals points outside or inside of the surface. But for the identification of a sharp feature this does not matter at all. We use the fact that each point x_{ij} on the Gauss sphere has it's counterpart x_{ji} on the opposite side of the sphere, since $n_{ij} = -n_{ji}$. This can also be seen in Figure 17. The clustering behavior of one set of points is thus reproduced identically on the other hemisphere.

Distance measure. An important step in a clustering algorithm is to choose a distance measure, which decides how close two elements are. The present point set has two particularities: it is a spherical point set and it consists of pairs of symmetric points, i.e. points lying on opposite positions on the sphere. An appropriate distance measure has to take care of this. In order to preserve symmetry in the point set we choose as distance measure the angle between the lines through pairs of symmetric points. It can be shown that the angle between two normal vectors of our Gauss map is equivalent to the geodesic distance between two points on the sphere. The minimal angle between two

lines spanned by the normal vectors is thus an appropriate distance measure satisfying the particular requirements mentioned before:

$$d(x_{ij}, x_{jk}) = \min\{d_g(x_{ij}, x_{kl}), d_g(x_{ij}, x_{lk})\}, \tag{3.4}$$

where $i, j, k, l \in I_p$ and

$$d_g(x_{ij}, x_{kl}) = \arccos(< n_{ij}, n_{kl} >)$$

is the geodesic distance between two points x_{ij} and x_{kl} on the unit Gauss sphere.

Clustering. Many clustering algorithms require an a priori specification of the number of clusters to produce. Our aim however is not to cluster the total point set, but to distinguish some clusters from other sparsely distributed points in the Gauss map. We therefore use a hierarchical agglomerative ('bottom-up') clustering method [24]. It begins with each point as a separate cluster and merges them successively into larger clusters. As linkage criterion, which defines the distance between two clusters, we use the mean distance D_c between elements of each cluster

$$D_c(S_1, S_2) = \frac{1}{|S_1| \cdot |S_2|} \sum_{x \in S_1} \sum_{y \in S_2} d(x, y), \tag{3.5}$$

where S_1, S_2 are two clusters to be compared and d the distance measure of the Gauss map defined in (3.4). This is one of the most common criterion [24]. Each agglomeration increases the distance between clusters by merging the closest clusters. To find the two closest clusters we have to compute the distances between all clusters. After the merging of the two closest, we only have to recompute the distances to the new cluster. We stop the clustering algorithm when the distance between two clusters exceeds a certain threshold $\sigma \in [0, \frac{\pi}{2}]$.

Hierarchical clustering algorithms have complexity of $O(n^2)$ with n the number of elements to be clustered. In our case, the number of elements is $k * (k - 1)$. So in total, our clustering has complexity of $O(k^4)$. The complexity of the clustering thus strongly depends on the size k of the neighborhoods. Since we use not very huge k in our method (normally $k = 16$, at most $k = 32$), the method overall is still reasonably fast.

Analysis. All clusters containing only a few points are discarded, since they correspond to the noisy points. The remaining clusters are analyzed as follows. Opposite clusters on the sphere are considered as one cluster. If in the end a single cluster remains we decide that the current point does not belong to a feature. If two, three or four clusters remain, we decide that the point belongs to a sharp feature. If more than four clusters

remain, we decide that the current point does not belong to a feature. The number of four seems to be a good value, since most data sets generally don't have more than four sharp features meeting in one point, see Figure 20 for examples. Even the more complex vase in Figure 32 does not have sharp feature corners with more than three edges. However, it can be adapted in presence of a particular data set if necessary.

Let us discuss in more detail, how we choose the parameters and thresholds used during the clustering in the next section.

3.5 CHOICE OF PARAMETERS

The decision, to declare a sample point as sharp feature or not, is depending on two value that we can change: the size of the local neighborhood (k) and the sensitivity parameter (σ). The next sections will show the influence of these parameters on the result and behavior of the algorithm and give some advices how the parameters should be selected. After this, we will show how we could improve the method further.

3.5.1 *Size of neighborhood*

The size of the neighborhood is user controlled and fixed. It represents the region tested for a feature and has influences on the computation time. On one hand, a too small neighborhood will not deliver enough information for a reliable result but will be computed very fast. On the other hand, a too big neighborhood represents a huge region and thus even features far away from the sample point would influence the result. This may lead to false positives, i.e. finding sharp features in flat regions. In addition, the computation times for a large neighborhood are significantly longer. A too big neighborhood may also violate the requirement that all points of the neighborhood belong to the same connected region of the underlying surface. This is a theoretical problem, which does not occur with the relatively small number of neighbors we choose. However, this problem has been pointed out by Pauly et al. [39], since their algorithm loops on the size of the neighborhood varying from 1 to 200. A criterion is proposed to detect invalid neighborhood sizes.

So there is a trade off needed for the size of the neighborhood. During our tests a neighborhood with a size of $k = 16$ performs best. Detection methods for general features come to a very similar conclusion. [21] propose $k = 10$ to 16. [47] propose

$k = 12$. Sizes of 8 and 32 also deliver acceptable results but were kind of lower and upper bounds. Let us use Figure 21 to show and explain this. In the left image, the neighborhood size is $k = 8$. This generates clearly to not enough information for a good detection. In the middle with $k = 16$ the result is just fine, while in the right image at $k = 32$ the size of the neighborhood begins to become too large for a good detection using our method.

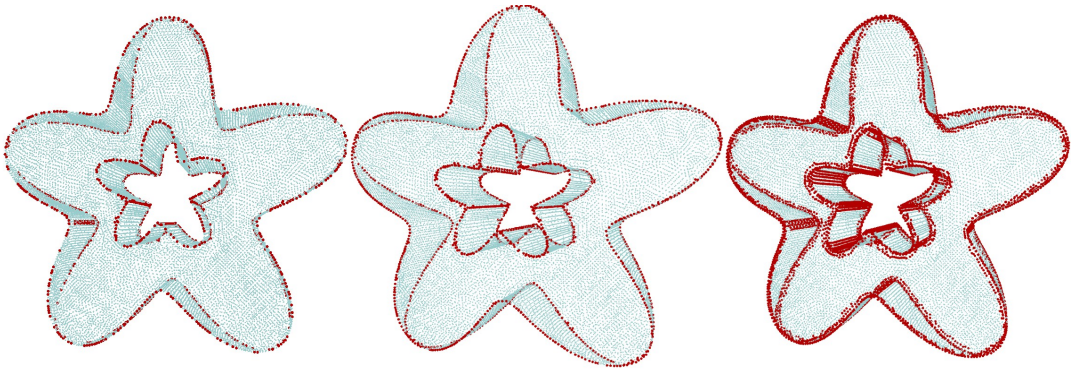


Figure 21: Feature detection on the trim star with a different neighborhood sizes and fixed sensitivity parameter. The neighborhood sizes vary are from left to right 8, 16 and 32.

3.5.2 *Sensitivity parameter for Gauss map clustering*

Remember that during clustering the distance D_c (3.5) between two clusters is compared to a threshold value. This threshold value is the sensitivity parameter σ. The clustering algorithm stops merging the clusters, when the distance between the clusters exceeds the threshold. σ corresponds therefore the *minimal distance* between all resulting clusters.

Let us explain the role of this parameter by first recalling the two main requirements to our method:

1. detection of all points lying on a sharp feature
2. no selection of points which are close to the feature, but not on it.

These requirements will lead our method to detect a relatively sparse set of points lying on the sharp feature or very close to it. Note that the second requirement is particular to our method, since in most cases, a sharp feature corresponds to a line. All previous methods are different in the sense that they aim to detect general features (not

sharp features). Such a feature often corresponds to surface region with high curvature variation. Consequently, they compute many points on the feature followed by a post-processing step, which reduces the number of points and which replaces them by an approximating line [21, 39].

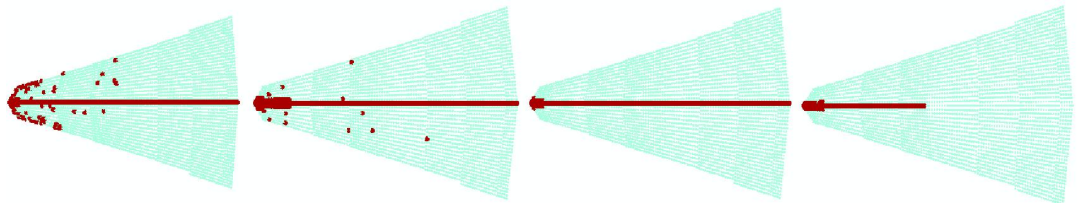


Figure 22: Feature detection with a global sensitivity value σ. The points are sampled on a piecewise bilinear surface with a sharp feature line, similar to the example in Fig.23-left. The angle between the surfaces varies along the feature line from acute (45°) to obtuse (140°). The detected feature points are highlighted by fat red points. $\sigma = 0.05, 0.1, 0.6, 1.0$ left to right.

The value fixed for the parameter $\sigma \in [0, \frac{\pi}{2}]$ makes the method more or less sensitive for sharp feature detection. The sensitivity is inverse reciprocal to the value of σ. Let us first investigate theoretically how the method is expected to behave for big and small sensitivity values, before studying a numerical example:
A big value of σ corresponds to clusters with a big distance between them obviously implies that clusters would consist of many points. This would work well for features which are distinguishable very sharp, i.e. features with a right angle or an acute angle. But at the same time it would ignore features with an obtuse angle. In this case it is more difficult to distinguish the noisy points from the correct surface normals. If σ is too big, one would end with only one cluster containing all points of the Gauss map and the sample point would not be recognizes as a feature, violating requirement 1.
A small value of σ stops clustering earlier. It would result in clusters which are more close together corresponding to feature with an obtuse or an acute angle. On one hand the method becomes thus more sensitive for detection of critical sharp features. On the other hand it may then be difficult to distinguish feature points from neighbor points, since their clustering behavior is very similar. It would violate requirement 2.

It seems that there might be some critical features which are difficult to detect. In practice, the choice of a user-controlled global sensitivity parameter works well for many examples. But for more complex examples, a global sensitivity parameter does not show

optimal results in presence of acute and flat angles in the same data set. We already suspected this behavior in the previous paragraph, so let us study now the limits of the present method with a particular numerical example. We constructed a piecewise bilinear surface with a straight feature line where the angle between the surface varies from acute (45°) to obtuse (140°). This data set corresponds to the left example in Figure 23. In Figure 22 is shown the result of feature detection for $\sigma = 0.05, 0.1, 0.6, 1.0$. It can be observed, that a very small value of $\sigma = 0.05$ or 0.1 produces many overrates not only near acute features, i.e. points which are falsely detected. A mean value of 0.6 detects all features but still produces overrates near acute features. A big value of $\sigma = 1.0$ however fails to detect flat feature points.

Figure 23: Some examples of sharp feature profiles with varying angles.

Obviously, if the point cloud contains features with acute angles and features with obtuse angles, one global parameter for sensitivity is not sufficient. A good global value for obtuse angles allows to detect all features, but will probably overrate features in regions of acute angles. In this case a whole region might get marked as sharp feature while the exact position of the feature cannot be determined. So a global value always needs a trade-off between finding all features (requirement 1) and finding the exact position of the features (requirement 2).

This observation motivated us to develop an extension of the algorithm which is presented in the next section.

3.6 LOCAL-ADAPTIVE METHOD

In the previous section it has been demonstrated, that the use of a global sensitivity parameter σ might not be sufficient in all cases. In the present section we will show how to make the parameter σ local and adaptive. The aim is to develop a method that changes the sensitivity value adaptively for different regions of the point cloud. It should compute automatically an optimal σ value for each feature candidate. Furthermore we also adapt the neighborhood size k automatically, so that the user is no longer obliged

to adjust the two parameters manually in order to get good results.
We achieve this by an iterative process. In a first initialization step the method described in Section 3.4 including flatness test on the whole point set and Gauss map clustering on the selected points does a feature search using global values for σ and k. In this first passage, σ will be set to a relatively low value while k will be relatively large. This delivers a set of many feature candidates. We generate a larger set in order to not miss any feature points with an obtuse angle, at the cost of also getting overrated acute angle features (see Figure 24). Since we now have a candidate list with many outliers, we reduce the number of candidates in the following iteration steps. In these steps the method checks the number of possible features in the neighborhoods of the feature candidates.

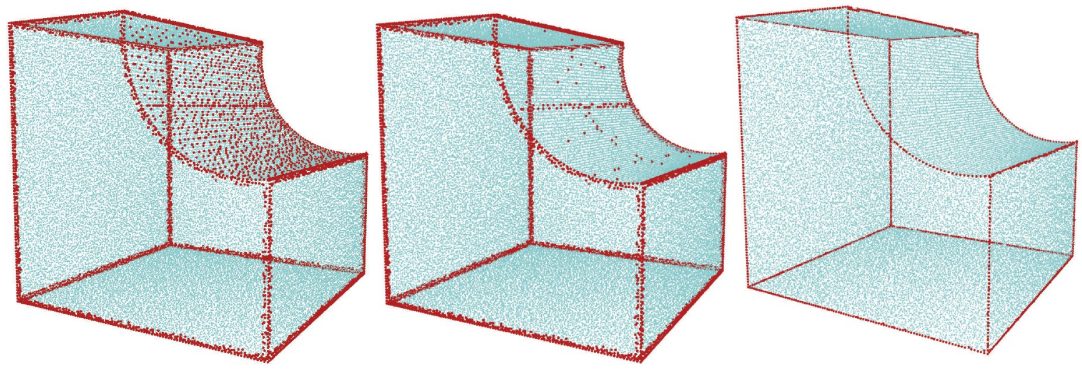


Figure 24: The two pictures on the left show a non-uniformly sampled point cloud with an overrated set of feature points obtained with two different parameter settings $\sigma = 0.1$, $k = 20$ (left), $\sigma = 0.5$, $k = 16$ (middle). The right figure is the result obtained with the iterative method applied to the overrated examples. σ and k are automatically adapted for each of the feature candidates.

In the case of a sharp feature e.g. an edge or a curved line, the feature points will lie on a line dividing the neighborhood in two parts. The percentage of feature candidates inside this neighborhood will be relatively low. A very high percentage of feature candidates inside a neighborhood indicates that the sensitivity was too high, i.e. value of σ too low, for this neighborhood. The existence of only one or no other feature candidate in the neighborhood indicates that the candidate is not a sharp feature but an outlier. Thus increasing carefully the sensitivity value σ would reduce the number of overrated points, see Figures 22, and 26. To also get rid of the overrated features at acute angles which don't disappear with increasing σ, we also locally reduce the size of the neighborhoods at every iteration step.

The algorithm for each iteration step is the following. We raise σ by 10% in the neighborhood of the current candidate and reduce the neighborhood size k by one.

Then only the candidate features inside this neighborhood are tested again for being a sharp feature using the Gauss map clustering described above. The raised sensitivity and the smaller neighborhood size lead to a reduction of the number of candidates in this neighborhood.

This step iterates until either the percentage of features in the neighborhood of the candidate reaches a reasonable value (during our experiments a value of 30% of neighborhood size worked well) or another break condition is reached. These additional breaking conditions can be chosen among maximum value for the sensitivity ($\sigma = 1.2$), a minimal size for the neighborhood ($k = 8$) and a maximum number of iteration steps.

The iteration process in pseudo code:

```
while ( iteration break criteria not reached ) {
    for ( Candidate in Feature Candidates ) {
     initParameters( n, sigma )
     Neighborhood = Candidate.Neighborhood

     while (Neighborhood.getNumberOfFeatureCandidates() ⩾ threshold ) {
      adjustParameters( n, sigma )

      //Check Candidate
      isFeature=checkforFeature( Candidate, n, sigma )

      if ( isFeature == false )
       removeFromCandidateList( Candidate )

       // Check Neighborhood of Candidate
       for ( Neighbor ∈ Neighborhood )
        if ( Neighbor.isFeature() ) {
         isFeature=checkforFeature( Neighbor, n, sigma )

         if ( isFeature == false )
          removeFromCandidateList( Neighbor )
        }
     }
    }
}
```

The right picture in Figure 24 is the result obtained with the iterative method applied to the overrated examples. σ and k are automatically adapted for each of the feature candidates. The two pictures on the left show a point cloud with an overrated set of feature points obtained with two different parameter settings $\sigma = 0.1$, $k = 20$ (left), and $\sigma = 0.5$, $k = 16$ (middle).

Adding iterative refinement steps to a process often increases significantly computational costs. Especially regarding large point clouds, checking the neighborhood of every data point iteratively is very costly, $O(Nk)$ in the worst case where each point has been selected as a feature candidate. This scenario is of course unrealistic.

In the present setting, the additional computational costs resulting from the iteration steps are quite low for two reasons. First, the number of data points checked during the first iteration is negligible with respect to the initial data set. They correspond to the feature candidates selected after the first pass using the Gauss map clustering with a set of not optimal global parameters. In the case of the 'cube with hole' examples, they present only 11% of the point cloud. In the case of the simple cube they present 9% and in the planes with the varying angles example 10%. Second, the number of remaining feature candidates decreases significantly after each iteration. This results from the fact, that each iteration of a single neighborhood will also eliminate a whole group of other candidates inside this neighborhood, since the neighborhoods of close points intersect each other. That means that the neighbors of the candidate, in case of a being not a sharp feature, are likely to be eliminated from the candidate list before their own neighborhood has been tested. This reduces the number of candidates very fast and fewer neighborhoods in the cloud will be tested again.
In a usual scenario after the first step only a small percentage around 10% or less of the point cloud will be marked as candidate for a sharp feature. This makes the additional computational costs for the refinement iterations significantly lower than the costs for the first global feature search.

3.7 RESULTS

We have implemented the sharp feature detection pipeline described in the previous sections. We use points sets sampled from some known geometries, such as the cubes

and the bilinear surfaces, as well as more complex models, such as the "fandisk" the "vase", and the "trim-star". To all models we applied both the basic algorithm described in Section 3.4 where we tried to find some optimal global parameters by testing many combinations, and the improved local-adaptive method with an automatic and adaptive choice of the optimal local parameters. The robustness of the method is tested and evaluated with respect to two aspects: the variation of the angle of a sharp feature and noise.

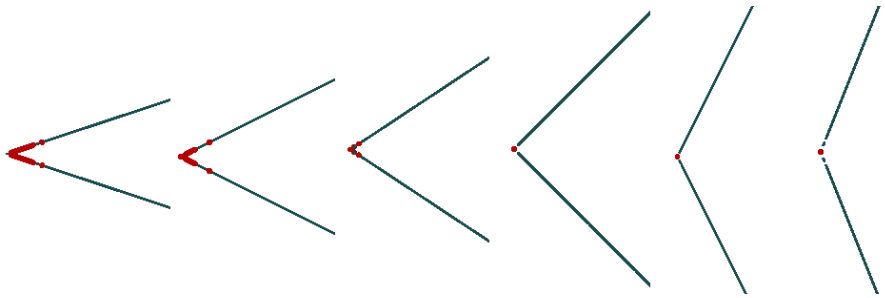


Figure 25: Feature detection on different angles using the local adaptive method.

3.7.1 *Robustness w/r to varying angles*

Let us first test the robustness of the method with regard to very acute and obtuse angles by testing two planes connected with a fixed angle between them. The profiles of these surfaces are shown in Figure 25. >From left to right the angles are 25°, 45°, 60°, 90°, 110°, 140°, 160° and 170°. The method works perfectly for all angles greater then 45°, even very obtuse angles don't cause any problem. The method also allows angles of 45° and less, but with very acute angles the results are no longer very precise. In this case, the overrated points don't disappear even when increasing σ and decreasing k. For the three acute angles the neighborhood was $k = 8$ for all others we chose $k = 10$. The problem of not detected features appears only with very obtuse angles near 180°, which is negligible.

3.7.2 *Comparison between global and local-adaptive method*

Let us now compare the performance of both methods (global and local-adaptive) using three test examples with a known number of sharp feature points. The first example is a

simple cube with 580 sharp feature points. The second example is a surface containing a sharp edge with a varying angle having 256 sharp feature points. At one end the angle is (45°) at the other end it is (140°). The third example is cube with a hole with 1350 sharp feature points. We checked the global method with different sets of parameters and compare it to the local-adaptive method.

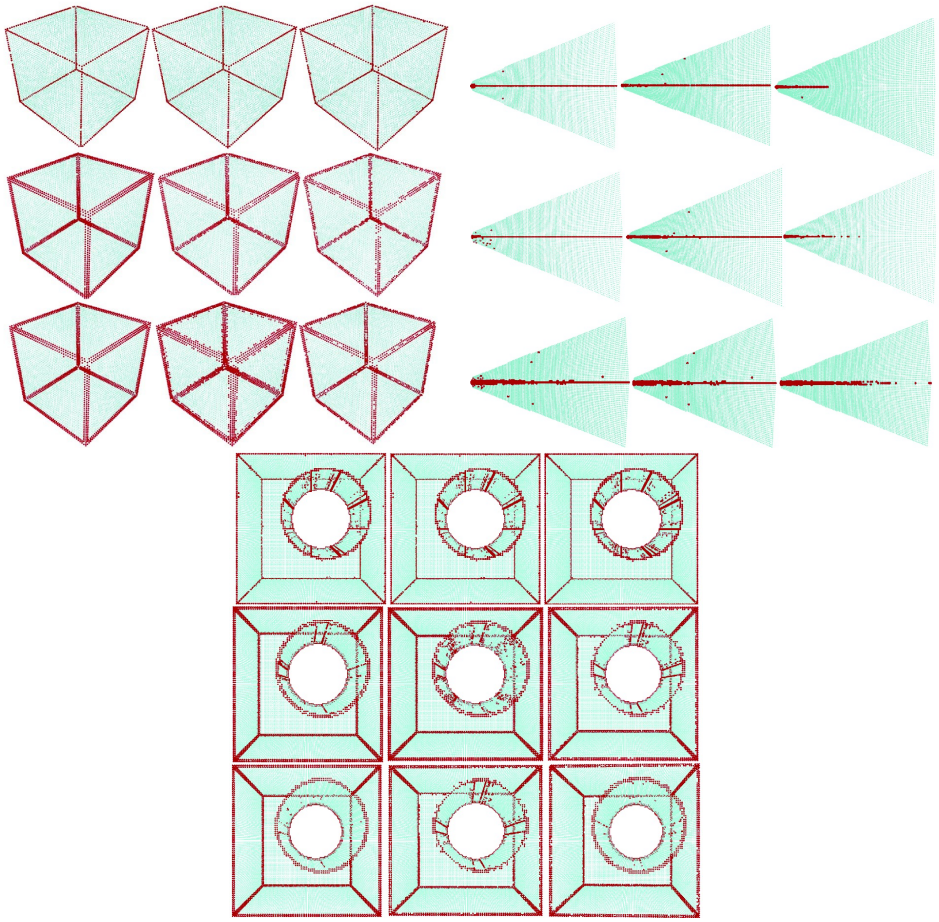


Figure 26: the examples used for the study with the global method. For each example 9 sets of parameters have been tested. The three lines correspond to $k = 10, 16, 20$, the three columns correspond to $\sigma = 0.1, 0.5, 0.8$.

3.7.3 *Global method*

Figure 26 shows the results for the global method. For each example 9 sets of parameters have been tested. The three lines correspond to $k = 10, 16, 20$, the three columns correspond to $\sigma = 0.1, 0.5, 0.8$.

The simple cube example shows, that for a dataset with only one kind of angle it is possible to find good global values for σ and k. $k = 10$ gives best results, see Figure 26 (first line). The optimal value of $\sigma = 0.6$ was found by trying several times the algorithm. It not fits into the figure here, but it results in a perfect feature detection of all 256 points of the cube equivalent to the local-adaptive method in Figure 27.
In the example of two surfaces joining with the varying angle, one can see the problems of the global method. A global value of σ is not sufficient. On one hand, the feature points on the right part of the edge, where the angle become obtuse, are not detected when $\sigma = 0.8$ is big (right column) in Figure 26. On the other hand, a small σ can detect them, but leads to outliers and overrates in acute angle regions. The size of the neighborhood k has a huge influence on the method too.
The importance of k is best seen in the cube-with-hole example. On one hand, a small neighborhood $k = 10$ performs best on the outside edges of the cube, but leads to bad results in the curved region inside the hole. On the other hand, a big neighborhood $k = 20$ performs better in the curved areas, but worse on the outside edges of the cube. All these examples confirm the conclusion we already did in Section 3.4: a global choice of the parameters can not detect all features satisfactory.

3.7.4 *Local-adaptive method*

The local-adaptive method can take a huge advantage on its capability to vary σ and k. For each feature candidate detected in a first pass, an optimal value of σ and k is determined and outliers are eliminated. However, the method is not able to add feature points during the iterations, since the final set of detected features is a subset of the feature candidates. Therefore, one has to initialize the iteration with a set of parameters, which won't neglect possible features. The experiences in the previous paragraph show that $\sigma = 0.1$ and $k = 20$ are good initial values. The result of feature point detection using the local-adaptive method is shown in Figure 27. Table 1 resumes the performance of the algorithm with respect to the three hand-made examples.

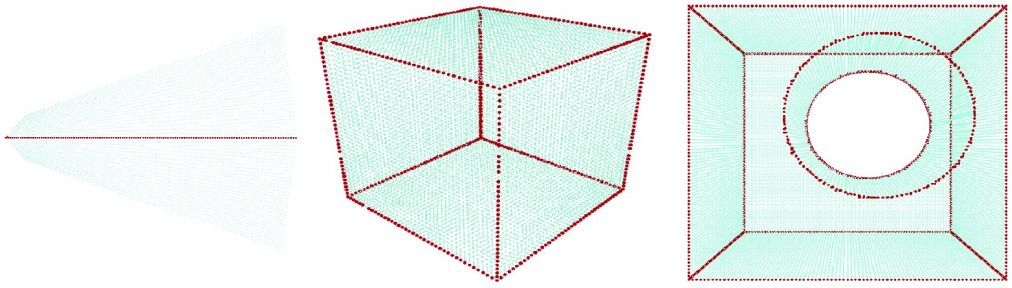


Figure 27: feature detection using the local-adaptive method.

	# points	feature points	detected	%	outliers
cube	10250	580	575	99%	
planes	6090	256	257	100%	1
hole	58720	1350	1398	100%	48

Table 1: Test results for the local-adaptive method.

3.7.5 *Robustness to noise*

In order to test and quantify the sensitivity of our local adaptive method to noise, we compare distances measured between real feature points on a test point cloud and on a perturbed point cloud. The test example is the cube-with-hole, since it has corner, convex and concave feature points. All points are lying in a ball of radius $R = 8.5$. The original point cloud has 17400 uniformly sampled points. Distance between neighboring points is 0.25. The perturbed point cloud is obtained by adding to each point a random vector chosen in a ball whose size is 0.4%, 0.8% and 1.2% of R.

The error induced by the noise is measured using the distances between the set of feature points in the original point cloud $Q = \{q_i\}$ and the set of estimated feature points in the perturbed point cloud $P = \{p_i\}$. The distances δ_∞ and δ_{avg} are defined similar to [35]. δ_∞ is the maximal distance between an estimated feature point p_i and its nearest feature point in Q. δ_{avg} is the average distance between an estimated feature point p_i and its nearest neighbor in Q.
Table 2 summarizes the results of the experiment for different noise radii. The number

noise	δ_∞	δ_{avg}	# features
0.0 %	0	0	839 (+0)
0.4 %	0.25	0.02	862 (+23)
0.8 %	0.26	0.03	875 (+36)
1.2 %	0.31	0.08	984 (+145)
1.6 %	0.75	0.19	1245 (+406)
2.0 %	0.93	0.21	1255 (+416)

Table 2: Distances between estimated features and real features of the cube-with-hole model. Noise as a percent of the model radius varies. The number of exact features 839 is compared to the number of estimated features.

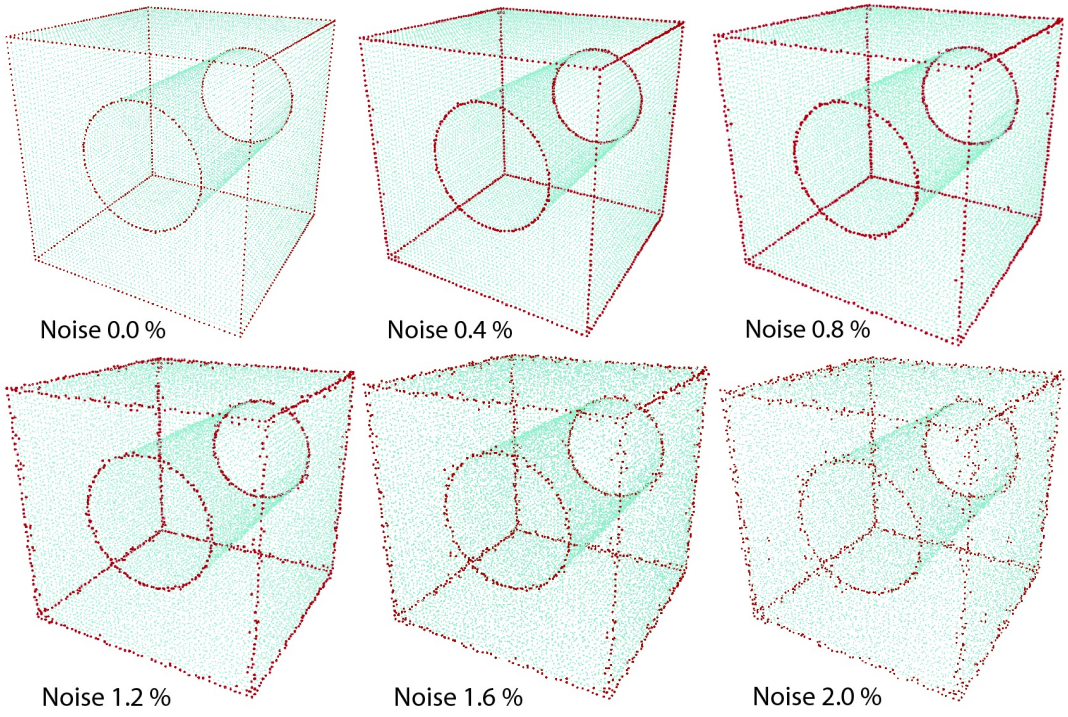


Figure 28: Estimated feature points on noisy point clouds. The original cube-with-hole model is perturbed with random noise.

of detected feature points is also a measure of quality, since our method aims to detect only feature points or a sparse set of points very close to a feature,

δ_∞ measures the presence of outliers. The experiment shows that in the case of mean noise (0.8%) the distance of isolated outliers $\delta_\infty = 0.26$ is equal to the point distance (0.25) in the original point cloud. Outliers are thus very close the neighbors of real features, see Figure 28. For bigger noise of 1.2% the isolated outliers are still very close to the feature points, but their number increases. Even though the number of so-called false features increases with increasing noise, the detection method is still stable, since isolated outliers stay close to real features, even for strong noise.

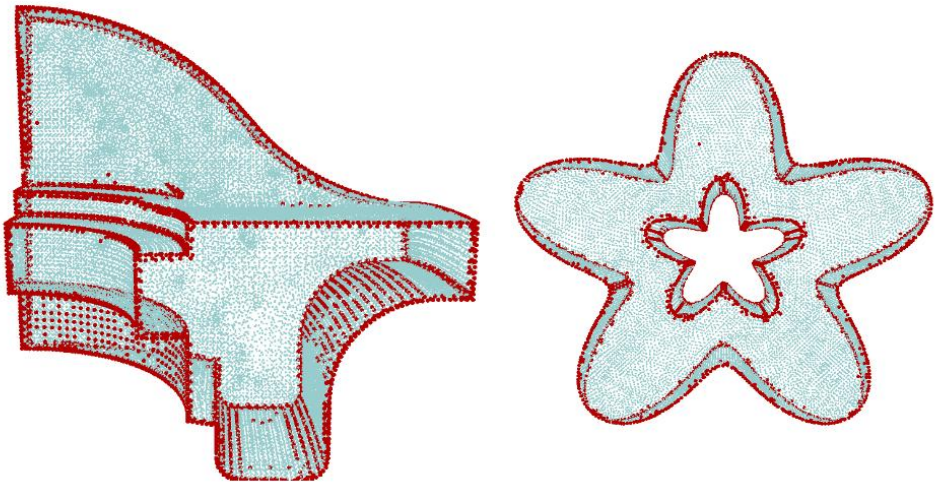


Figure 29: The global method on the fandisk and the trim-star.

3.7.6 *Complex point-sampled surfaces*

After these self constructed examples let us show some more real world examples. Here it is not possible anymore to measure the quality of the output of the algorithm, since the feature points are not known and generally don't lie directly on the feature as stated in [5]. Only a visual control is possible. First we tried the global method on the well-known fandisk ($\sigma = 0.7, k = 12$) and the trim-star ($\sigma = 0.6, k = 12$). It can be seen in Figure 29 that no optimal result can be achieved. We then applied the local-adaptive method to the following three examples. Figure 30 shows the detected sharp feature points on the fandisk model (41250 vertices). The local method here again delivers good results.

The next examples in Figures 31 and Figure 32 show the trim-star model (25100 vertices) and the vase (896000 vertices) which has some curvy, sharp features. Notice, that for the vase and the trim-star, the original model is a triangulation, but the points

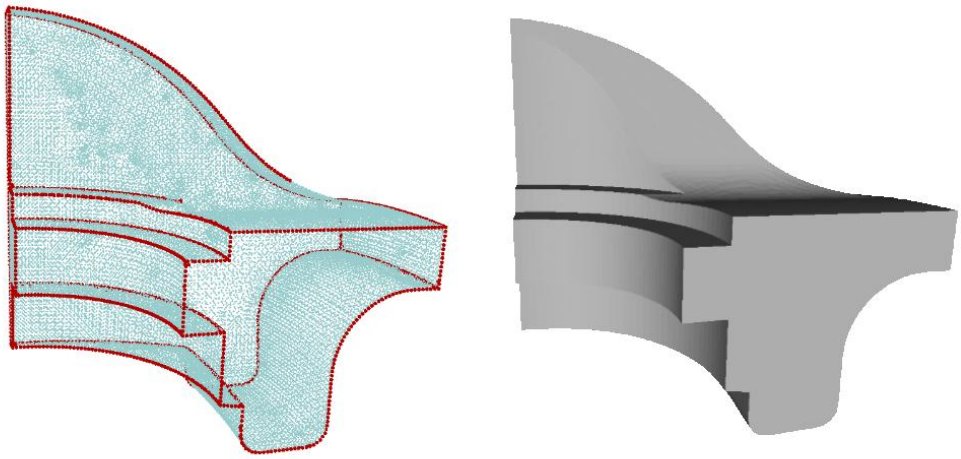


Figure 30: Sharp feature detection on the fandisk model.

are NOT lying on the feature. The triangle edges are zig-zagging along the feature lines.

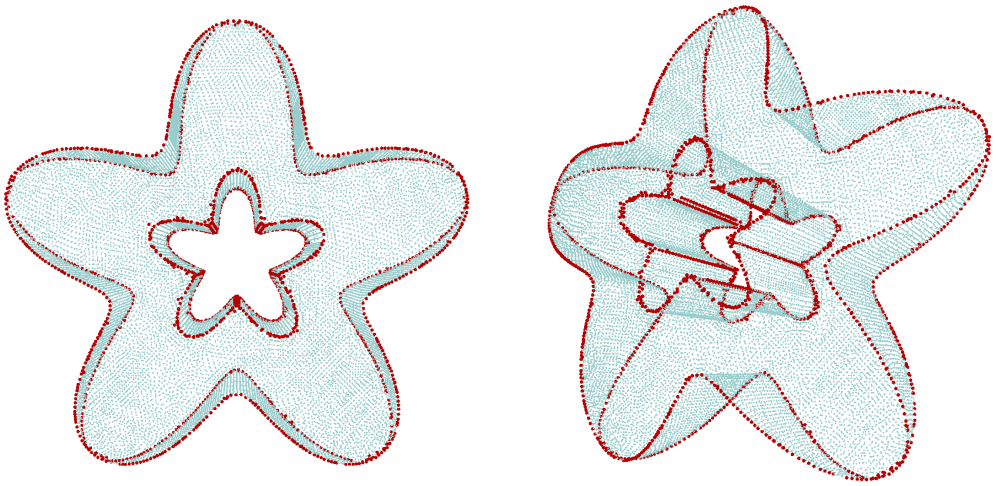


Figure 31: Sharp feature detection on the trim-star model.

Finally we used a scan data set by a Cyberware^{TM} scanner of a drill. Many data is missing near the sharp features and the precision is quite rough, see Figure 33, but the feature lines are correctly detected along the sharp lines of the drill.

Concerning computation time, our local-adaptive algorithm runs for the fandisk example with 41520 vertices in about 19 seconds, for the trim-star with 24444 vertices in

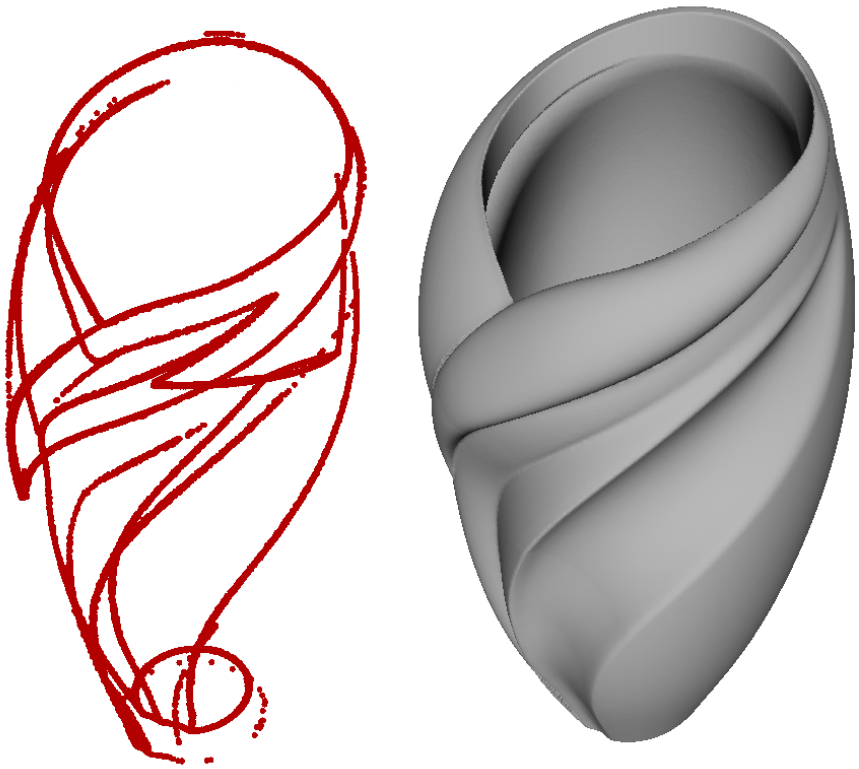


Figure 32: Sharp feature detection on the vase model.

30 s and for the drill with 23994 vertices 40 seconds. 0.6 seconds pre-processing time costs the k-nearest computation for the fandisk (PC: Xeon 3.0Ghz). The vase model takes 20 mn because of the time consuming and repeated clustering in the local method. This is however reasonably fast for most available models up to 100k points, and compares well to the multiscale PCA method, even though our implementation is not done optimally.

3.7.7 *Comparison to PCA methods*

The ability of our method to detect sharp feature points is the key characteristic and can be seen as a supplement to all previous methods. In fact, the notion of 'sharp' is

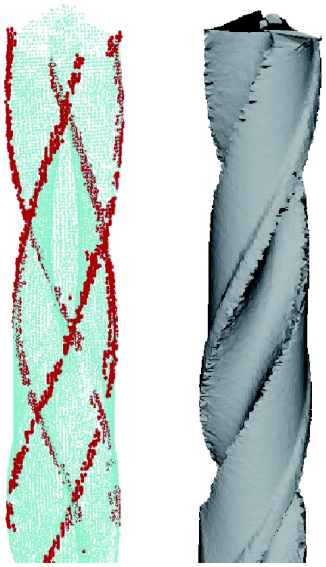


Figure 33: Sharp feature detection on a drill scan by a Cyberware^{TM} scanner. The data set is very rough and has missing data near the sharp feature.

important here. Previous methods, including [21, 39, 35], aim to detect more general features in unstructured point clouds such as high curvature or curvature variation regions. These methods result in large bands of feature points, even in the presence of sharp features. A post-processing step is then proposed in [21, 39] to make this set of points more sparse followed by an approximation with a smooth curve. All methods use variants of a principal component analysis (PCA) of the neighborhood of a point in order to estimate local curvature values.

In order to compare our local Gauss map clustering to PCA based feature detection for the special case of sharp features, we implemented the PCA part of the multiscale feature detection method [39]. It is a powerful method and easy to implement. Without measuring exactly the differences, Figure 34 makes a visual comparison. As it has been expected, by tuning several times the parameters of the multiscale method (surface variation 0.08, neighborhood size varies from 10 to 40, feature weight with lower and upper border 25 and 30, see [39]), we always end up with a wide band of estimated feature points which call for a post-processing to reduce the number of candidates to a small line in contrast to local-adaptive Gauss map clustering. It confirms that PCA-based methods are not best appropriate for sharp feature detection.

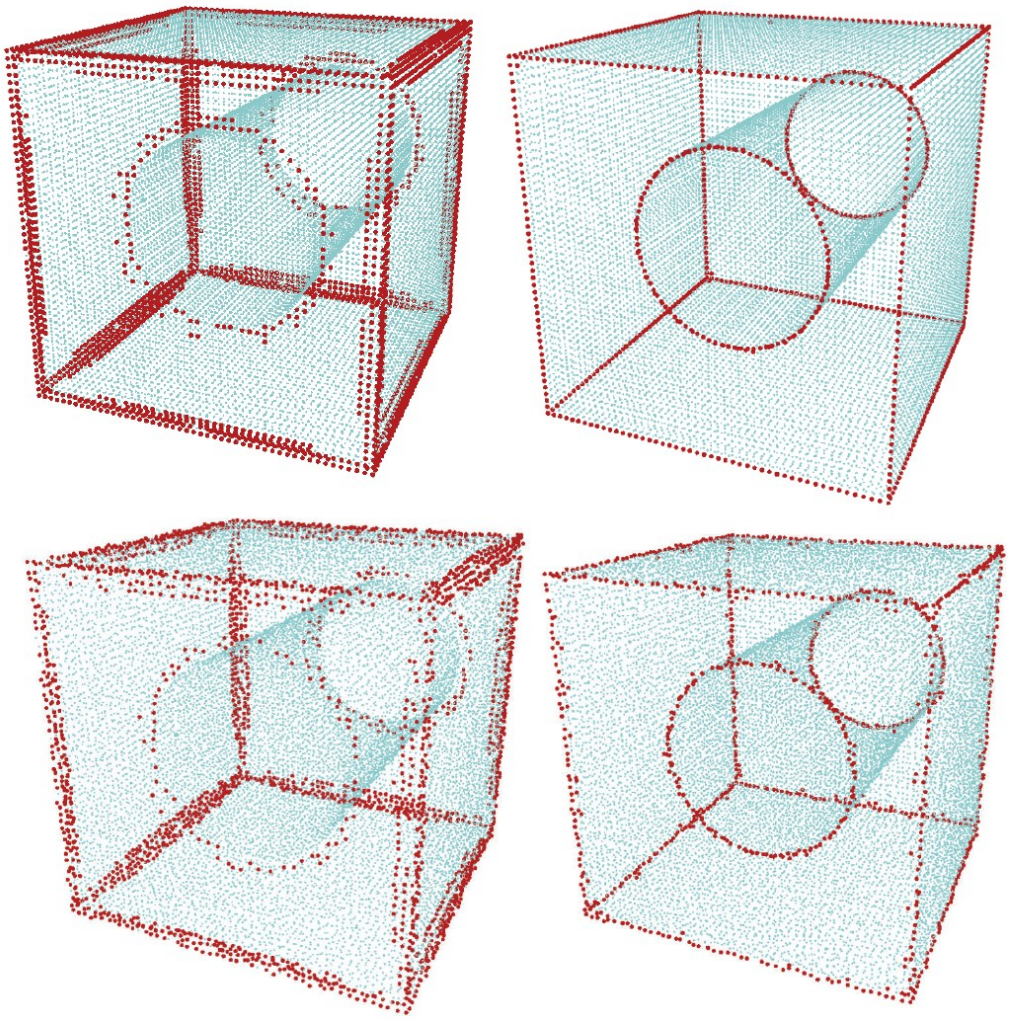


Figure 34: Comparison of PCA-based feature detection (left) with our local-adaptive Gauss map clustering method (right). Both methods are applied to the original uniformly sampled point cloud of the cube-with-hole example (upper row) and to the perturbed data set with 1.2% of noise (lower row).

3.8 CONCLUSIONS

In this chapter we presented a new method for sharp feature detection on point-sampled geometry. The proposed method uses Gauss map clustering for feature detection. It is fully automatic, without any user interaction. It does not rely on local surface reconstructions, and no normal information is required. A key contribution is the integration of an adaptive local sensitivity parameter for the feature identification, which reduces the

user dependency of the method. Many tests have been performed in order to demonstrate that the method works very successfully even on very complex geometry in contrast to algorithms based on global parameters. The resulting point cloud with the marked sharp features can be used for several applications now (surface reconstruction, non-photorealistic rendering, mesh generation, MLS-surface modeling). All line-type, corner-type sharp features are detected. Cone peaks are not treated here, the method needs to be adapted in order to recognize the particular clustering behavior in this case. The properties of the resulting feature point set (preciseness, sparseness, very few outliers) make the method an excellent pre-processing step for a surface reconstruction with sharp features. We will show this in the next chapter

Part IV

SHARP FEATURE RECONSTRUCTION USING MOVING LEAST SQUARES

4

SHARP FEATURE RECONSTRUCTION USING MOVING LEAST SQUARES

In the last chapter we showed a method to identify sharp feature positions in point cloud data. In this chapter we will use this knowledge for the reconstruction of the sharp features on the surface defined by the point cloud. As basis for our surface reconstruction we use the moving least squares approach (MLS) introduced by Levin [33] in 2003. MLS is a well known method for the surface reconstruction from point cloud data. But the usual approach produces C^2-continuous surfaces, that smoothes out all sharp edges from the data set. We enhanced the general approach by modifying the local neighborhoods in the presence of sharp features. The modification allows us to reconstruct sharp features in the dataset during the surface reconstruction. One advantage of this approach is, that we only have to manipulate regions with sharp features, while we can conserve the advantages of usual MLS, e.g. smoothing of noise, in the regions without sharp features.

4.1 ABSTRACT

Let's first give a short description of the method. We use the moving least squares approach for the surface reconstruction. This is a well known method for point cloud data but has the disadvantage of smoothing sharp edges in the point cloud. To get this problem solved, we first need to know the exact positions of the sharp edges in the data set. For this we use the feature extraction method of the last chapter as pre-processing step. After this step, we know for every point in the data set if it belongs to a sharp feature or not. We now use this knowledge for the MLS reconstruction. MLS is based on local approximations. So, during the projection step of a point onto the surface, we use a neighborhood of this point in the point cloud. If this neighborhood contains no sharp feature, we can do a usual MLS-projection. Only in regions containing sharp features, we need to change the approach. Our basis approach to reconstruct the sharp feature is to modify the neighborhoods and ignore features on the other side of the sharp edge. We first construct a local feature line inside the neighborhood that divides this neighborhood in distinct parts. We can than search for points in the neighborhood of the projected

point which are on the other side of the feature and just ignore these points during the MLS.

4.2 moving least squares and sharp features - state of the art

MLS was originally presented by Levin [33, 32]. He also introduced the projection procedure approach we are using to solve the MLS problem, see Section 2.3.2. Later his work was improved by Alexa et al. and the construction of their point set surfaces [2], and Kolluri [30] who introduced a provably good moving least squares. In this work Kolluri proved the correctness of the approximation using moving least squares. But all those approaches constructed smooth approximations of the surface defined by the data set. That means, they cannot deal with and reconstruct sharp features in the underlying dataset. In the last years a couple of approaches were made to integrate sharp features to MLS. In this section we will give an overview of some of the most popular approaches for moving least squares with sharp features.

As one of the first methods Fleischmann et al.[16] presented a MLS approach that could reproduce sharp features. The robust moving least squares fitting approach is based on robust statics. The robust statistics is used to search for outliers in the point set. They assume that the surface consists of several smooth patches connected by sharp features. The idea is now, that a sample point lying on another smooth patch will be identified as outlier. As we will see later, our approach is based on a quite similar principle. An example of the results of this method is shown in Figure 35. The left side of this figure shows the classical smooth reconstruction, the right side the reconstruction with sharp features. Instead of using the classical method for fitting a model to data via linear regression using least squares, they used a method that is more robust in respect to outliers based on the least median of squares. It estimates the parameters of the model β by minimizing the median of the absolute residuals, defined as the difference between measured data and estimated data.

$$\mathrm{argmin}_{\beta}\, \mathrm{median}_i \left| f_{\beta}(x_i) - y_i \right| \tag{4.1}$$

It can handle to fit a model to data that contains up to 50% outliers. Fleischmann et al. use a random sampling algorithm to solve problem 4.1. First, they select k points of the input data randomly and fit a model to these points. Then the median of $r_{i\beta}$ with

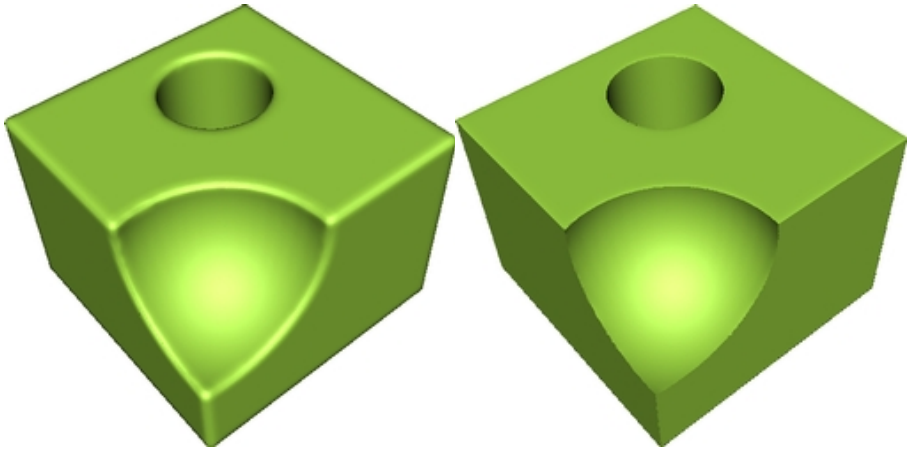


Figure 35: Example for Fleischmanns sharp feature reconstruction (from: [16])

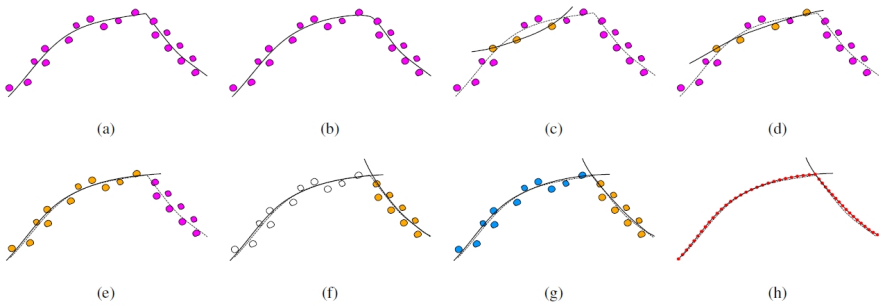


Figure 36: the principle of the iterative refitting (from: [16])

$r_i = f(x_i) - y_i$ of the remaining points is computed. They repeat this process T times to generate T models. The model with minimal median residual is then selected as final model. This model is now used as initial model for the iterative refitting shown in Figure 36. Figures (a) and (b) show the interpretation of the data points as piecewise smooth (a) and smooth (b) surface. To identify the sharp feature in the iterative process they first robustly fit a surface to a small subset of the points in (c). In the next step they add points with smallest residual and refit the surface to the updated subset (d). The final fit of the forward search is shown in (e). The remaining points are regarded as outliers to the first surface. These points are used in another refitting step to construct another surface part (f). The result is a surface that is defined as the intersection of the two surfaces in (g). Finally they reconstruct their piecewise smooth surface by re-sampling of the intersection of the two surfaces (h).

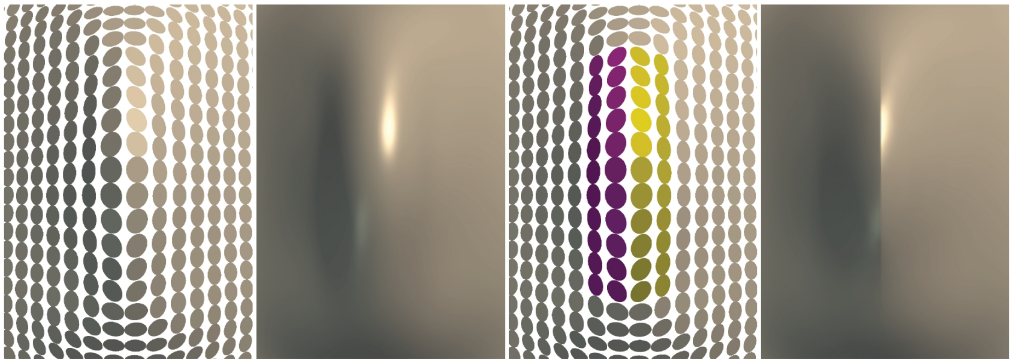


Figure 37: Tagging the point cloud with sharp features (from [20])

Gueenebaud [20] presented later APSS, algebraic point set surfaces. He used moving least squares fitting of spheres instead of planes. This leads to a good stability in undersampled datasets. The sharp feature extraction itself is done manually by tagging of the point cloud, or automatic and analogue to Fleischmann's method.
For n points they use $W(x)$ the $n \times n$ diagonal weight matrix and D the $n \times (d+2)$ design matrix.

$$W(x) = \begin{bmatrix} w_0(x) & & \\ & \ddots & \\ & & w_{n-1}(x) \end{bmatrix}, D = \begin{bmatrix} 1 & p_0^T & p_0^T p_0 \\ \vdots & \vdots & \vdots \\ 1 & p_{n-1}^T & p_{n-1}^T p_{n-1} \end{bmatrix}$$

The solution of the algebraic sphere fit at point $x \in \mathbb{R}^d$ is then:

$$u(x) = arg_u min_{u \neq 0} \left\| W^{\frac{1}{2}}(x) Du \right\|^2$$

Here u has to be constrained by a metric to avoid the trivial solution $u(x) = 0$. The authors use Pratt's constraint. It fixes the norm of the gradient at the surface of the sphere to unit length 1.
For the sharp feature reconstruction they use the concept of 'sharp points', that means points at a sharp feature provide two normals. The user tags the points in the points cloud witch are meant to be sharp features, see Figure 37.

In the end, they insert samples into groups, and assign lower weights to samples inserted into other groups than the own one. A parameter α controls the smoothness of the feature. $\alpha = 0$ reconstructs a sharp feature, $\alpha = 1$ completely smoothes the feature, see Figure 38.

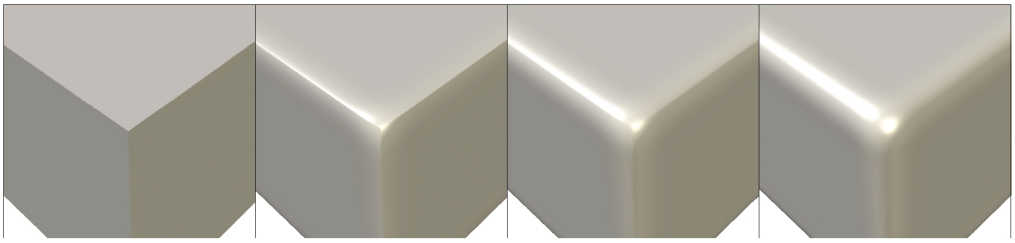


Figure 38: Sharpness control with $\alpha = 0, \alpha = 0.15, \alpha = 0.5, \alpha = 1$ (from: [20])

Öztireli et al. [37] use a kernel regression technique to reconstruct sharp features. They called it RIMLS, Robust Implicit Moving Least Squares. Analogues to Fleischmann they also use a robust statistic approach to find outliers belonging to different smooth patches on the surface. This technique has global parameters that can control the global sharpness of the reconstruction. Another important property if this approach is, that the resulting surface remains C^2-continuous. So the reconstruction does not have a real C^0-continuous sharp feature, but constructs a presentation which is still "C^2-continuous but looks sharp from the distance. Going close up to the surface, one can see the smoothness of the 'sharp' edges. Depending on the applications demands, this can be seen either as advantage or disadvantage.

Local kernel regression is a supervised regression method to approximate a function $f(x) : \mathbb{R}^d \rightarrow \mathbb{R}$ given by its values $y_i = f(x_i)$ at sampled points $x_i \in \mathbb{R}$. Core of this method is to approximate the function around the evaluation point x in terms of a Taylor expansion:

$$f(x_i) \approx f(x) + (x_i - x)^T \nabla f(x) + \frac{1}{2}(x_i - x)^T Hf(x)(x_i - x) + ...$$

where $Hf(x)$ is the Hessian matrix of $f(x)$. In their paper they show that one can transform this equation into:

$$\mathrm{argmin}_s \sum (y_i - (s_0 + a_i^T s_1 + b_i^T s_2 + ...))^2 \phi(x)$$

using a weighted least squares minimization for the unknown parameters $s = [s_0, s_1^T, s_2^T, ...]$ and $\phi(x)$ as symmetric decreasing weighting function. This is in principle equivalent to the MLS scheme.
A combination of the IMLS surface definition [42] with the local kernel regression approach yields to the robust IMLS surface they called RIMLS. It is defined by :

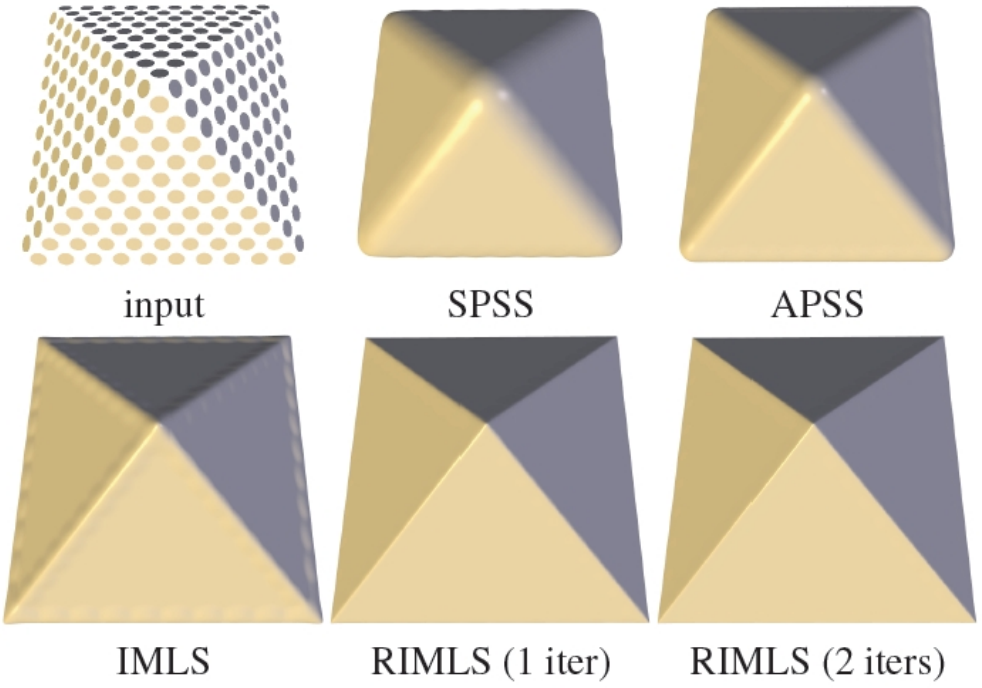


Figure 39: Reconstruction of a sharp feature using RIMLS (from [37])

$$f^k(x) = \mathrm{argmin}_{s_0} \sum (s_0 + (x_i - x)^T n_i)^2 \phi_i(x) w(r_i^{k-1})$$

with the residuals $r_i^{k-1} = f^{k-1}(x) - (x - x_i)^T n_i$. To increase the accuracy of the sharp feature reconstruction, they suggest the addition of a second re-weighting term penalizing normals far away from the predicted gradient of the surface. The new weight function they propose is

$$w_n(\Delta n_i^k) = e^{-\frac{(\Delta n_i^k)^2}{\sigma_n^2}}$$

By iteration they can now get close to the sharp feature and keep the surface C^2-continuous. An example for this can be seen in Figure 39.

They also showed a comparison with non sharp feature techniques, see Figure 40.

Another interesting approach worth to mention in this context is the ERKPA by Reuter et al. [41] ERKPA stands for Enriched Reproducing Kernel Particle Approximation. In

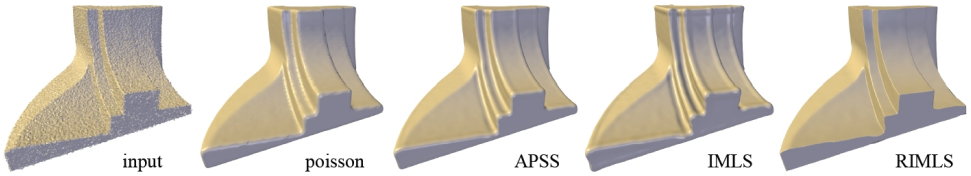


Figure 40: Various reconstructions of a noisy fandisk (from [37])

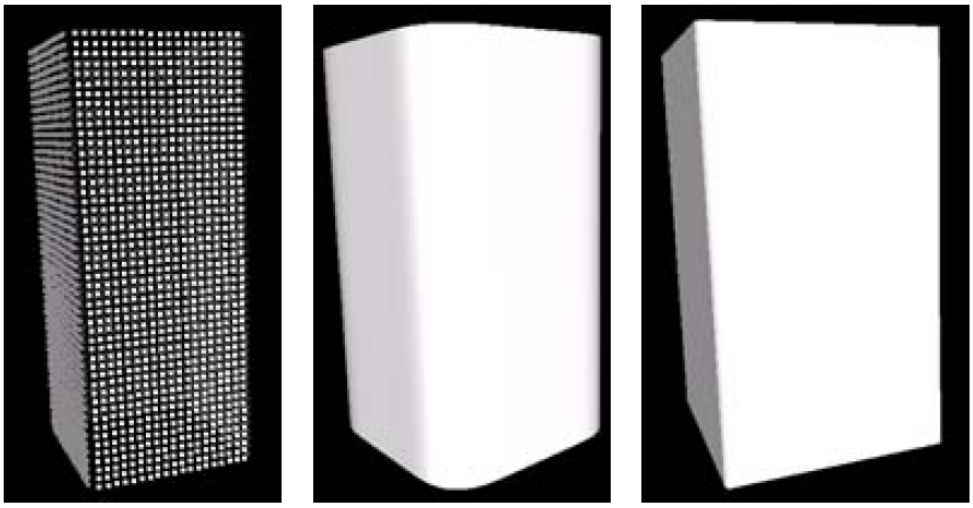


Figure 41: ERKPA Reconstruction of sharp features (from [41])

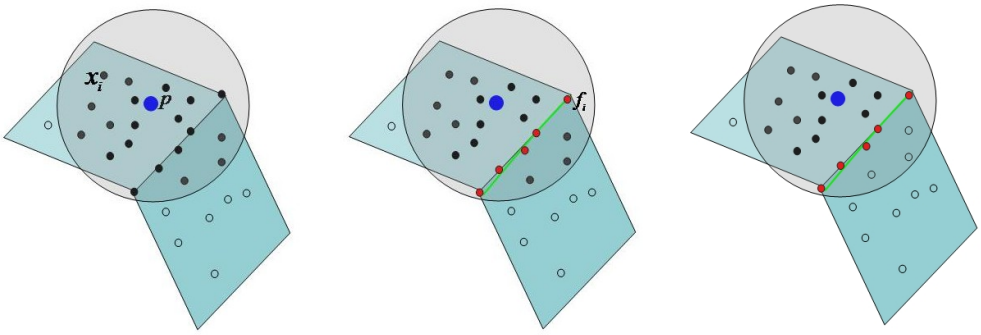
this approach the user has to tag the sharp features manually. Then Reuter et al. use a modification of the second step of the MLS projection (the computation of the local polynomial approximation). Instead of the normal projection, they use a projection operator based on ERKPA. They add an enrichment function $e(x)$ with discontinuous derivatives to the approximation function. The enrichment functions are compactly supported with a user specific support size to control the influence of the sharp feature. For n Features, n enrichment functions are needed. For the introduction of a sharp feature in the reconstructed surface, the user can specify a feature curve Λ_i. Then the enrichment function can be defined, so that the surface presents a tangent discontinuity along this curve. The feature splits the corresponding domain $\Omega \in \mathbb{R}^2$into two sub domains Ω_0 and Ω_1. Examples for the ERKPA reconstruction can be seen in Figure 41.

4.3 OUR METHOD - MLS WITH NEIGHBORHOOD MODIFICATION

In this section we will describe our method for sharp feature reconstruction and how we use and modify the usual moving least squares approach to reconstruct a surface containing sharp features. After presenting the basic idea of our method for sharp feature reconstruction in Section 4.1, we will go more into the details in the next sections. We will start with an overview of the algorithm in Section 4.3.1 followed by the details of the feature line extraction in 4.3.3 and an discussion of the quality of the feature lines in Section 4.3.4. Then we will describe how our method must be applied in the general setting 4.3.5 as well as the special case of corner-like features 4.3.6.

4.3.1 *An overview of the algorithm*

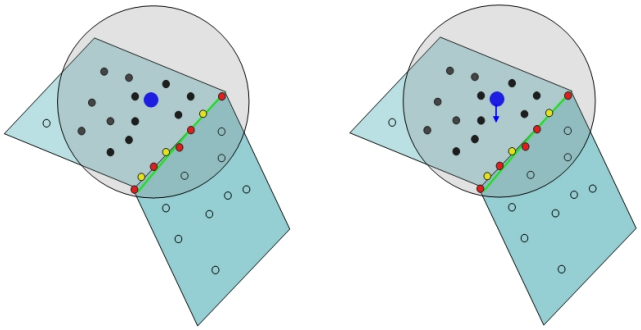
Let us now take a closer look to the details of our sharp feature reconstruction method. The principle of the whole process is shown in Figure 42. As mentioned before we use moving least squares for the surface reconstruction. This well known method for point cloud data in its classical form has the disadvantage of smoothing sharp edges in the point cloud. To solve this problem, we use the knowledge about the exact positions of the sharp edges in the data set, taken from the feature extraction method presented in Chapter 3 as preprocessing step. After the sharp feature detection we know for every point in the data set if it belongs to a sharp feature or not. This additional knowledge about the dataset is used to reconstruct the sharp features of the surface during the MLS-reconstruction. Let P be the point that is currently projected onto the surface. During the projection of the point P onto the surface, we have to compute the neighborhood of P in the point cloud. Reusing the kd-tree structure from Chapter 3, this is a simple task. If the resulting neighborhood contains no sharp feature, we apply the usual MLS-projection to construct the local approximation of the surface. Only in those regions containing sharp features, we need to change the classical MLS approach. Our basis approach for the reconstruction of the sharp feature is in fact not to modify the MLS method itself, but the neighborhoods used as basis for the reconstruction. If sharp feature points are contained in the neighborhood of P, we compute all the points in the neighborhood around P that are not on the same side of the feature than P and remove these points from the current neighborhood. To do so we approximate the sharp feature by the construction of a local feature line inside the neighborhood that divides this neighborhood in distinct parts. After this, we can check, which points inside the neighborhood are on the other side of



(a) Point P and its neighborhood N_P

(b) Feature points are identified and local feature line constructed

(c) Point that do not belong to the region containing P are removed from N_P



(d) Additional points are sampled along the feature line

(e) P is projected onto the surface using MLS projection

Figure 42: neighborhood modification, (a) neighborhood construction; (b) marking the features and construction of the feature line;(c) removing unwanted points from the neighborhood; (d) computation of points on the feature line to replace deleted points in the neighborhood; (e) projection of the point onto the surface

the feature than P and modify the neighborhood accordingly for the reconstruction. Next we will go through the details of the sharp reconstruction by distinguishing two cases of sharp features. The first case is the appearance of a line like feature in the neighborhood. This is the most common case. The second and special case arises if multiple feature lines meet inside the neighborhood. This is the case of corner like features.

4.3.2 *MLS Notations*

Before going into the details, let us introduce some notations and basics for the MLS reconstruction. The notations are also shown in Figure 43. Let P be a set of N unorganized points $P = \{P_1, P_2, ..., P_N\}, p_i \in \mathbb{R}^3$ sampled from a Surface S. Since we are working on subsets of the point set, N_p describes the neighborhood of a point $p \in \mathbb{R}^3$ in the dataset. p itself does not have to be a point of the dataset P. As in Chapter 3 we use the k-nearest neighborhoods, i.e. $N_p = \{x_1, x_2, ..., x_k\}$ that contains the $k \in \mathbb{N}$ points of the dataset with the minimal distance to p. The set of feature points is another important subset $F = \{f_i \,|f_i \in P \text{ is } feature\ point\}$ where $i \in \mathbb{N}$ is an arbitrary index since the features itself are not sorted.
We are going to use Levin's classical MLS-projection.
MLS-projection step 1:
In the first step of the classical MLS projection (also see Problem 2.16 in Section 2.3.2.2), we will use a local reference plane is computed by minimizing

$$\sum_{i=1}^{k} \left(\langle a, p_i \rangle - D\right)^2 w\left(\|p_i - q\|\right) \tag{4.2}$$

where q is the projection of p onto the hyperplane $H = \{x| \langle a, x\rangle - D = 0, x \in \mathbb{R}^3\}, a \in \mathbb{R}^3, \|a\| = 1$ and $p_i \in N_p$ are the neighbors of the point p. w is a smooth monotone decreasing weighting function

$$w(d) = exp^{-d^2/h^2} \tag{4.3}$$

where h can be used as a parameter to adjust the smoothing and interpolation behavior of the MLS, and d is the euclidean distance $\|p_i - q\|$.
MLS-projection step 2:
In the second MLS projection step (see Problem 2.18 in Section 2.3.2.2) the polyno-

mial approximation g of the surface inside neighborhood of p is than computed by minimizing:

$$\sum_{i=1}^{k} (g(x_i, y_i) - h_i)^2 \, w(\|p_i - q\|) \tag{4.4}$$

In this formula (x_i, y_i) are the coordinates of the projection of p_i on H in the local coordinate system of the reference plane. h_i is the height of p_i over H.

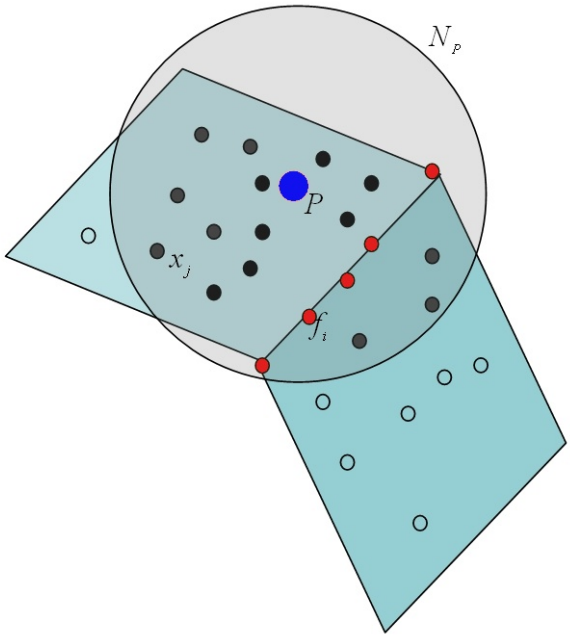


Figure 43: Overview of the situation of a single neighborhood.

4.3.3 *Local feature line construction*

In this section we explain how to modify the neighborhood of a point p for the MLS projection. In order to reconstruct sharp features we modify the local neighborhood N_p of p which is used in problem (4.2) and (4.4). Let again x_j denote the points belonging to N_p i.e. $N_p = \{x_1, x_2, ..., x_k\}$. We first check if some of the x_j belong to the set of sharp feature points F, we denoted as f_i. In a first step, the sharp feature inside N_p is approximated as a smooth curve. Second, all points $x_j \in N_p$ which don't belong to the

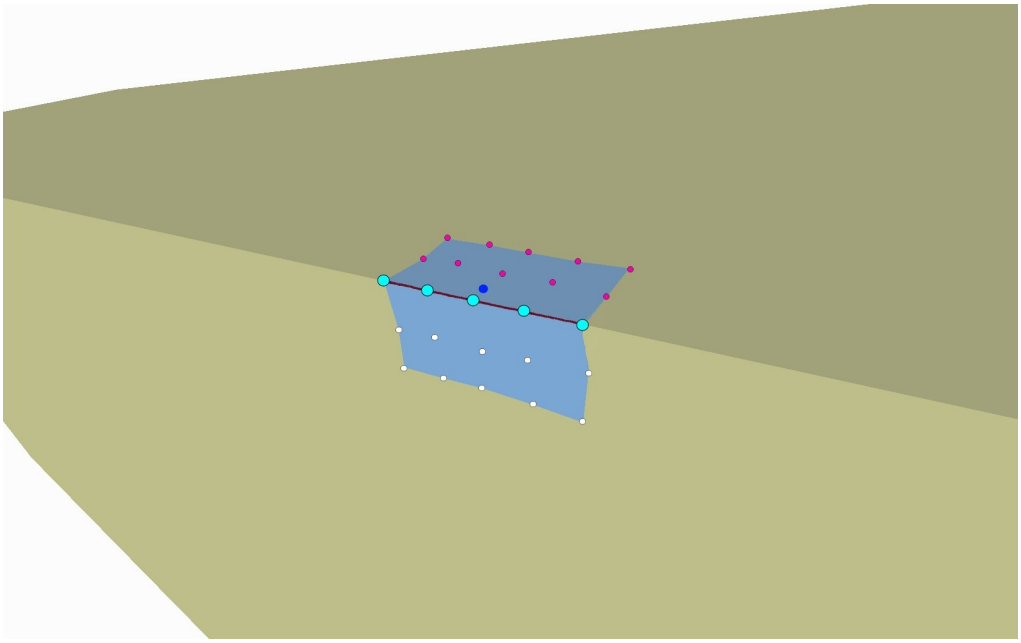


Figure 44: Feature line in neighborhood near a linear sharp feature along the edge of a cube

same surface part as p are removed from N_p.

Let us first focus on the local approximation of the sharp feature. The input for our feature line approximation are the feature points f_i inside the neighborhood N_p of the current point p. The neighborhood sizes we use are quite small. Usually we use a neighborhood size of $k = 20$. So the number of feature points, if there are, is usually small as one can see in Figure 43.

We use the feature points as input for a Bézier curve that approximates the sharp feature inside the neighborhood. The problem that arises here is the problem of the sorting of the points f_i. Since the data is unsorted we have to order the points for the curve generation. We decided to use a heuristic approach. We can assume, that the feature we are approximating is a line like feature, since the neighborhood N_p represents only a small area and thus the feature is locally an almost straight line. Using this we compute the distances between the feature points. The two points with the largest distance can be assumed to be the start and end point for the curve that means the control points b_0 and b_3 of the curve. One of these points is arbitrary used as the start point b_0. The other points are then ordered according to their distance to the start point. Now we have to choose the interior points b_1 and b_2 of the control polygon. If we have

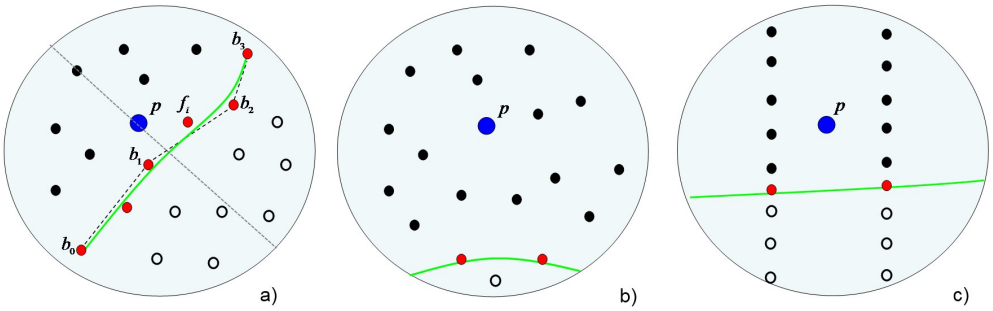


Figure 45: a) Feature line computation: red points are the feature points fi. A cubic B´ezier curve F (green) approximates the feature points. The control points b_0 and b_3 are set to the extremal feature points. A simple heuristic determines b_1, b_2. Together they form the control polygon on F (gray dotted); b) The corresponding situation with a small number of feature points f_i in N_p c) Situation with very non uniform sampling of the point set P

exactly four feature points we can just take these points. If we have more, we divide the area between the first and end point of the curve into two sectors by the bisector of the start and end point, and take one arbitrary point from each of those sectors to represent the interior control points. This process is shown in Figure 45a. With the neighborhood sizes we use, we usually get four to five feature points in the case of a line though the neighborhood. There are cases possible, where this way of approximation will fail. For example, if the dataset consists of points arranged in scanning lines like in Figure 45c and the sampling density along a single scanning line is much higher than the density between the scanning lines. In this arrangement of points, the k-neighborhood may consist only of points form one or two scanning lines, so that the neighborhood itself will not be a usable template for the surface or feature reconstruction. What remains is the case of having only a very low number of feature points f_i inside the neighborhood for a feature line approximation. In the case, that the number of feature points is lower than four, we ignore the feature points and do not approximate the feature since we use a cubic Bézier curves as approximation and though need at least four points. Also, such a small number of feature points is just not enough information for a useful interpretation and is common for situations where the sharp feature is positioned near the border of the neighborhood, or in a situation of wrong identified single sharp feature points. In both cases the influence of this neighborhood on the reconstruction of the sharp feature

is insignificant. Figure 45b represents this situation. If the neighborhood contains enough feature points, we approximate the feature by a Bézier approximation.

Figure 44 shows the result using data points sampled on a simple cube. The actual projected point is shown as blue dot. The neighborhood used for the local MLS and the construction of the feature line is shown in lighter blue, the feature points f_i of the neighborhood in cyan. The constructed feature line, shown in red clearly divides the neighborhood on a straight line. In the figure the points that are used for the final MLS projection in this neighborhood are marked in pink. The neglected points of the neighborhood are shown as white dots. In this simple case with all feature points lying on a straight line, the approximated feature line exactly matches the original edge.

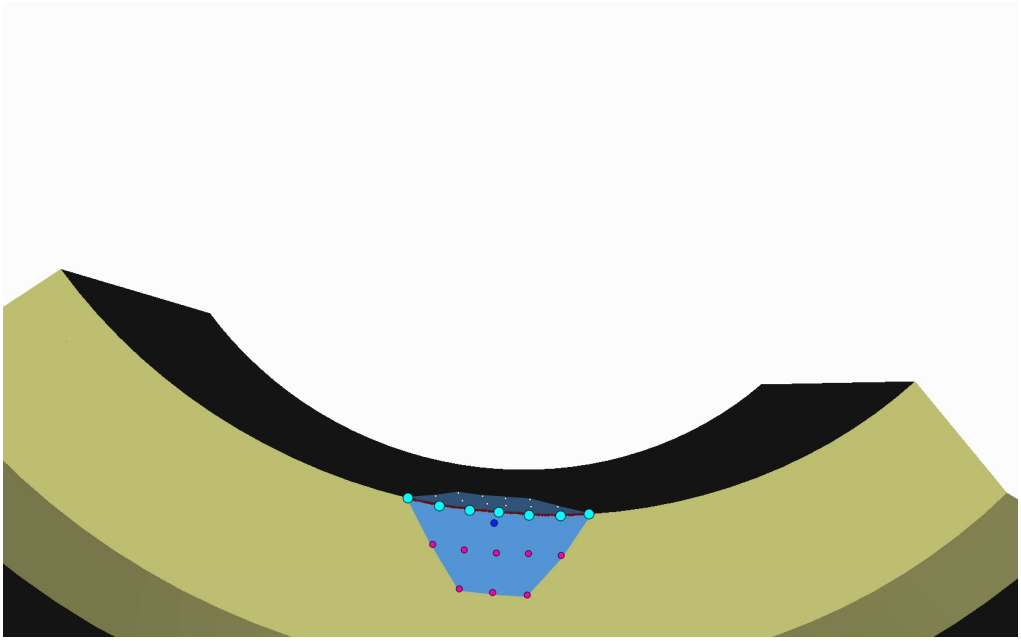


Figure 46: Feature line in neighborhood near a sharp feature edge in a curvy area of the fandisk

The result of the feature approximation in Figure 44 is made in an optimal scenario. In the next section we will show and evaluate the sharp feature approximation in the optimal and two more realistic situations containing curved areas. Examples concerning noise will be shown in Section 4.4.1.

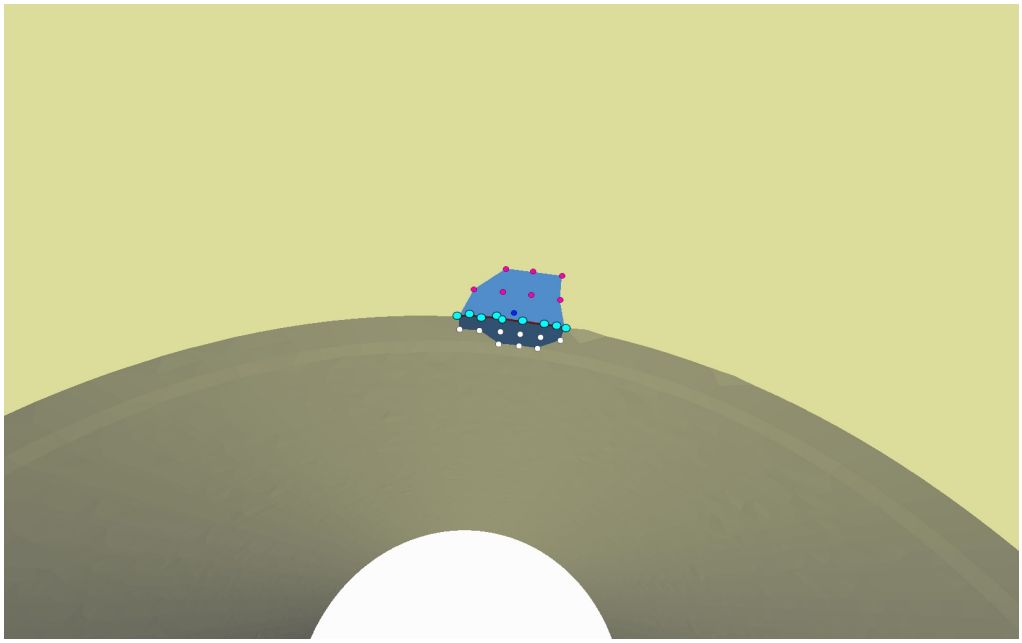


Figure 47: Feature line in neighborhood near a curved sharp feature

4.3.4 *Feature lines error analysis*

To evaluate the quality of our feature approximation we will now give an error analysis on the constructed feature lines. To measure the quality, we compute the distance of the reconstruction to the original and known feature lines from the original constructed data sets. As data set, we use the optimal scenario of the cube, a curved part of the fandisk model and another curved surface part inside the hole in the cube with hole model. In Section 4.4.1 we will use a section on the sharp feature of to intersecting curved planes in a noise free situation and at different levels of noise.
In the next examples, that although the perfect match of the feature line is only the case in such an optimal scenario, we still produce very good approximations in more common situations. So, the second example shows the behavior of the feature line generation in the curved area of the fandisk dataset. Here Figure 46 shows the situation. The feature line constructed by the method matches the wanted optimal feature line quite good. The third example in Figure 47 is a neighborhood positioned in the curved area of the hole in the data set of the cube with hole. Again, the approximated feature line is of a good quality. The three Figures 44, 46 and 47 show the results of the mentioned datasets in the noise free situation and one can see that the reconstruction is good

	Cube (edge)	Fandisk	Cube w hole
Max Error	0.0	0.0740	0.0447
Min error	0.0	0.0	0.0
Average error	0.0	0.0318	0.0206

Table 3: Error analysis of a single feature line. Measure for the error is the distance of the used feature line to the right feature line

these cases. In Table 3 we summarized the errors of the approximated feature line as distance to the real known feature line. The average distance of the points in the point cloud is about 0.125, the maximal error of the feature line in the fandisk example is about 0.07 for the fandisk and 0.04 for the cube with hole example. Those errors are on a very low level. The average error with 0.03 is even on a much lower level. This shows us, that the approximation works well in the noise free environment. For the results of the feature line generation in noisy data sets see Section 4.4.1 for a closer analysis.

4.3.5 *Modification of neighborhood*

After having the feature lines constructed and analyzed, we can now return to the reconstruction of the sharp feature and how we use these feature lines to modify the neighborhood during the MLS. If a usable feature line lies within a neighborhood, we have to think of two cases.

The most common case is the occurrence of a sharp feature in form of a curved line. This means that the local feature line in this neighborhood is a simple curve that separates the neighborhood exactly into two distinct areas. This matches the situation in Figure 42a. Let P be the point we want to project onto the surface. As mentioned above, we use the locally generated feature line to divide the neighborhood along the sharp edge and use only the points on the same side of the feature line than the current point P.

Having the feature line constructed like shown in Section , we now have to find and eliminate the unwanted points inside the neighborhood. These are the points that are positioned on the other side of the feature line than the currently projected point P. So we have to cross the feature line if we go from P to the unnecessary point on the surface.

But since the surface is not constructed yet, we use another criterion for this decision. We use an angle criterion to decide if another point from the neighborhood P_i is on the same side of the feature as point P. The principle of this is shown in the next Figure 48.

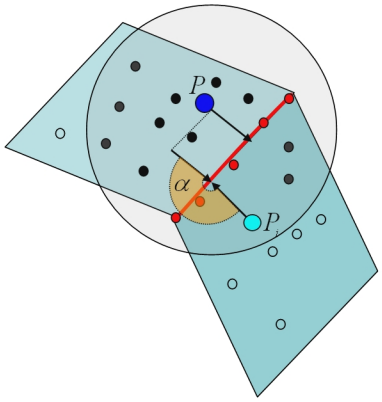


Figure 48: angle criterion; the angle α shows if P and P_i are on different sides of the feature

We compute the vector v_p from the current point P to the nearest point on the feature line as well as the according vector v_i from the other point P_i. After this, we compute the angle between these vectors.

$$\alpha = \arccos(v_p \times v_i), \tag{4.5}$$

If α is bigger than a threshold (we use 45°) we decide that it is not on the same side. This criterion turned out to work well in our situation but there may be situations or data sets where one has to adapt the threshold for better results. The result of this process can be seen in Figure 42d. Once the subset of N_P containing P has been selected, it can happen that the cardinal of the neighborhood is reduced significantly. To prevent problems that may arise through a too small number or neighboring points during the rest of the projection procedure, we have to replace the points we just removes from N_p. The best replacement for the lost points, especially regarding the reconstruction of the sharp feature are points that lie on the feature line. So, we check how many additional points are needed to fill up the neighborhood and than sample them along the local feature line accordingly. In the end, we receive a modified neighborhood of P, consisting of the original points on the same side of the sharp feature as P and additional points on the feature line, that replace the eliminated points (see Figure 42d). This modified

neighborhood can now be used in for a standard moving least squares projection of P. This way the smoothing effect on sharp features, which is a result of the overlapping of the neighborhoods and the blending of the resulting surface parts can be eliminated. Also the local approximation itself is closer to the sharp features, due to the elimination of the *'wrong'* neighboring points and their replacement on the feature line. The whole principle of the feature handling in the linear situation is shown in Figure 42

4.3.6 *Modification of neighborhood in the special case of corner features*

Up to now, we have assumed, that only one feature line traverses the local neighborhood N_P of P. But it can happen that several sharp feature lines intersect at a common vertex. The corner of a cube is a typical example where three sharp features meet. The general case, where several sharp features meet at a corner is called 'corner-like' feature. If this case happens inside the neighborhood we have to change the approach. In fact we can not represent a corner-like feature, where probably more than two lines meet with a sharp angle, by a single curve. Using the simple generation of a Bézier approximation as single feature line inside the neighborhood would lead to a bad and useless approximation of the feature line. Figure 49 shows this problem for two features in a single neighborhood. The attempt to generate a single approximation of the features delivers a wrong result.

One way to solve this problem would be to construct several feature lines inside the neighborhood. Unfortunately the number of detected feature points for the construction of more than one feature lines inside a single neighborhood is in most cases too small. And the use of larger neighborhoods however will lead to a worse computation time. So we decided to reconstruct only the dominating feature line inside the actual neighborhood. This means the nearest feature line to the point P which has to be projected. This way, the rejected feature lines will not be lost. They will be reconstructed in the neighborhoods in which they are the dominating feature line. To do so, we first need to sort the feature points inside the neighborhood according to the feature line they belong to. For this step, we use a modification of the gaussian clustering algorithm used for the feature detection in Chapter 3.

The idea is as follows:

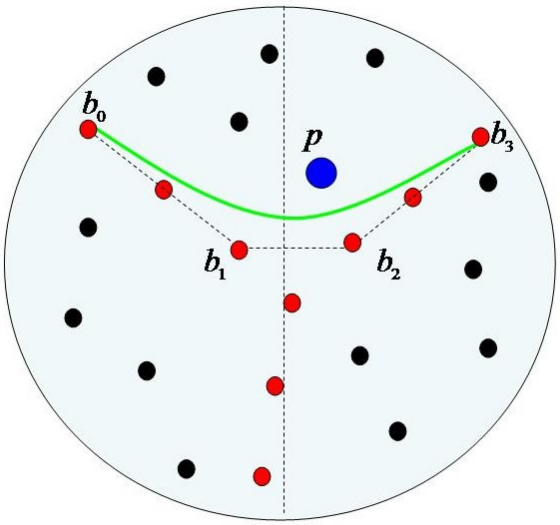


Figure 49: The figure shows the problem of multiple feature lines meeting at a corner. A single approximation of these feature points can lead to useless results

We compute for every feature point in the dataset the group of other features it is connected to inside its own neighborhood. We can compute this either on the fly during the MLS or directly after the feature detection for the complete dataset as preprocessing for the MLS. We decided to use the second variant, since it has to be computed for every sharp feature point anyway. We use a modification of the gaussian clustering algorithm from Chapter 3 to compute a list of the nearest connected features of every feature point in the data set. Since we do this as preprocessing step, we store the list for each feature point in a map. So let f_1 be the feature for which we want to find the connected feature points. The algorithm works in three steps. First, we take sets of three points inside the neighborhood of feature f_1: the actual feature point f_1, a second feature point f_2 and a non feature point p_i, see Figure 50. For each of these sets, we compute the normal of the resulting triangle. As in Chapter 3 we have to use unoriented normals as we do not know the orientation of the surface. For an better understanding, we removed the resulting opposing clusters in Figure 50 and show single clusters. In practice each single cluster represents two opposing clusters which have to be interpreted as one. We do so for every combination of the actual feature point f_1 with one non feature point and another feature

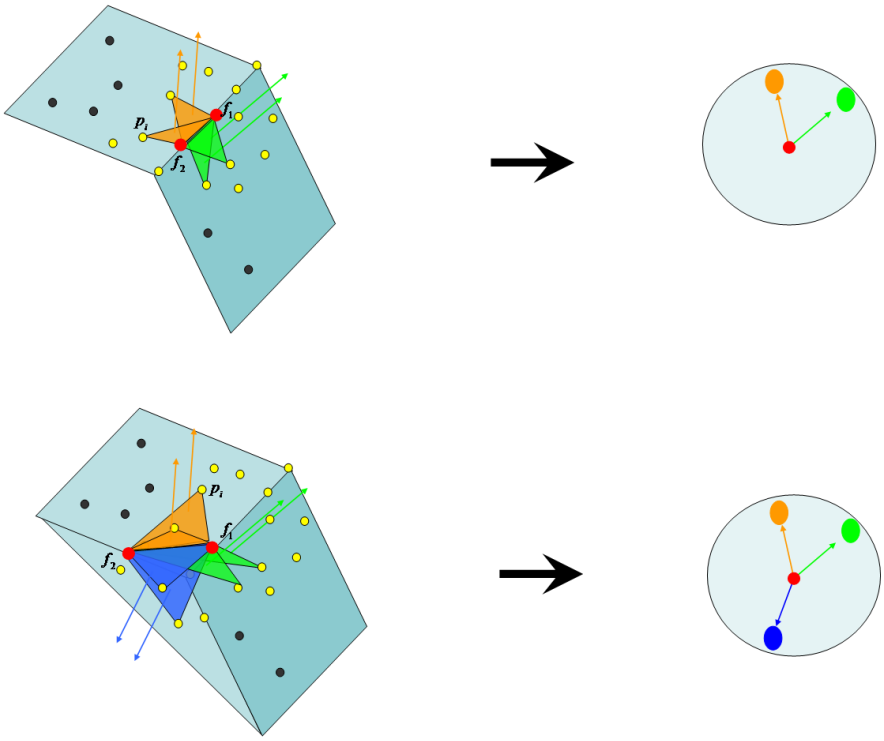
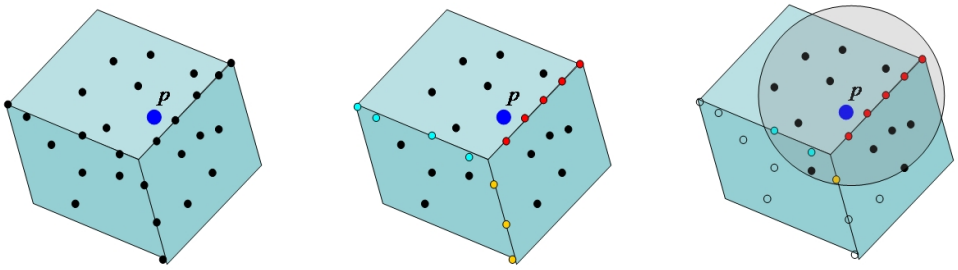


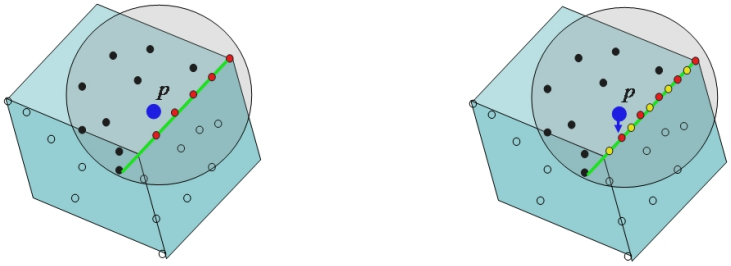
Figure 50: Feature clustering. The normals of the triangles of two feature points and one no-feature point are projected on the gaussian sphere; in the upper case the two feature points are positioned on the same edge, the clustering produces two clusters; in the lower case, the feature points are positioned on different edges. The clustering produces three clusters.

point in the neighborhood of f_1. In the second step, we project these normals onto the gaussian sphere around the actual feature point f_1. We apply a clustering analogue to the sharp feature detection and cluster the points onto the gaussian sphere. But this time we can use the result of the clustering to decide if the two features are positioned on the same sharp edge. Step three is the analysis of the clusters to decide if f_1 and f_2 are on the same sharp feature. There are different cases. If in the end two clusters remain on the gaussian sphere, the feature f_2 is on the same sharp edge as f_1, see Figure 50. If more clusters exist, like in the second example of Figure 50 they are positioned on different edges.

This way, we can collect the feature points that belong to the same edge for every feature point in its own neighborhood and store them in a list for each feature point.



(a) Starting situation at a corner (b) Clustering of the features (c) The neighborhood of P



(d) The nearest feature and its connected features are used to generate the feature line; points on the 'wrong' side are removed

(e) the removed points are replaced by points sampled on the feature line; P is projected onto the surface

Figure 51: neighborhood modification in corners, (a) situation at a corner; (b) clustering situation of the feature points; (c) construction of the k-neighborhood; (d) selection of the closest cluster, generation of the local feature line, deletion of unwanted points (analogue to linear situation); (e) replacing of the removed points in the neighborhood with points on feature line, projection of the P onto the surface

Figure 52 shows the result for a feature point near the corner of a cube. The red points in the figure are the features of the data set. The green dot is the actual feature of interest. Its neighborhood is presented by the circle. The yellow dots are the result of the gaussian analysis.

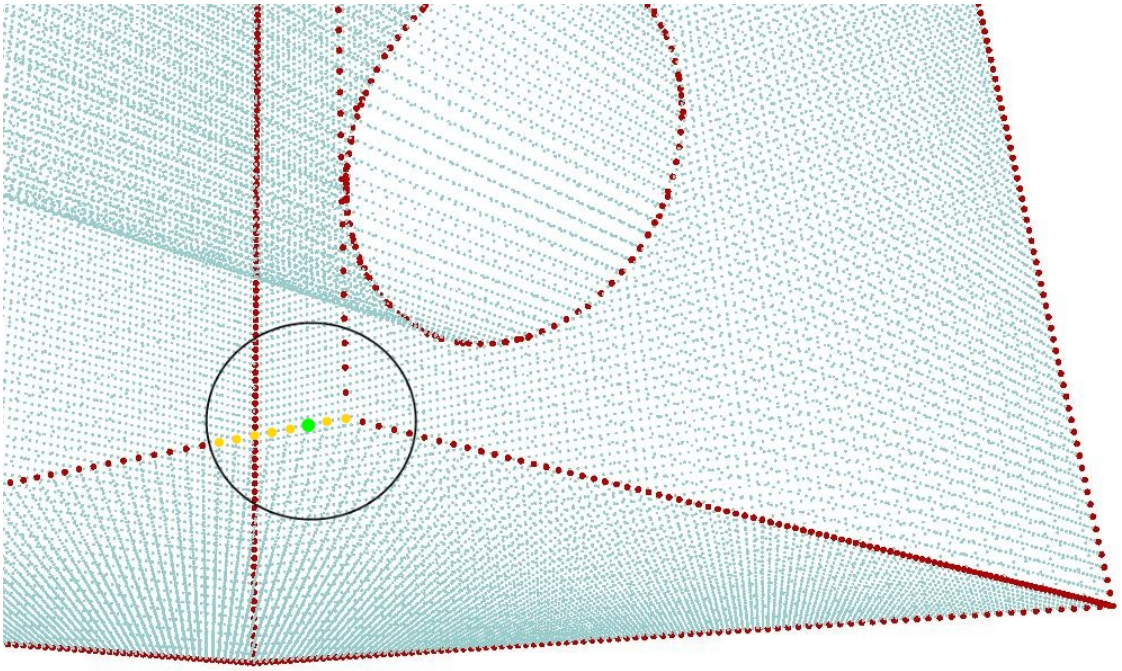


Figure 52: The result of the gaussian clustering for a feature near the corner of a cube. The green point is the feature of interest, the yellow points are its identified connected features.

The next step is to use this information and to eliminate feature points from the neighborhood for the feature line generation. Figure 51 shows the whole process. During the projection step for point P, we select the closest feature point P_f to P like in Figure 51d, and collect its neighboring features. We now use these feature points for the construction of the local feature line as in the linear case. The feature points that do not belong to the same edge are not deleted but treated as usual data points. The generated feature line is used to divide the neighborhood along the sharp edge and eliminate the points on the '*wrong*' side of the sharp feature, analogue to the linear case in Section 4.3.5. Again, we replace the removed data points by new data points sampled along the feature line. This new generated neighborhood is then used for the MLS projection like the one in the linear feature case. For the example of the feature line near the corner of a cube see Figure 53. Here, like in the examples above, the red line shows the feature line, the blue dot the projected point P and the pink dots, the used neighborhood points.

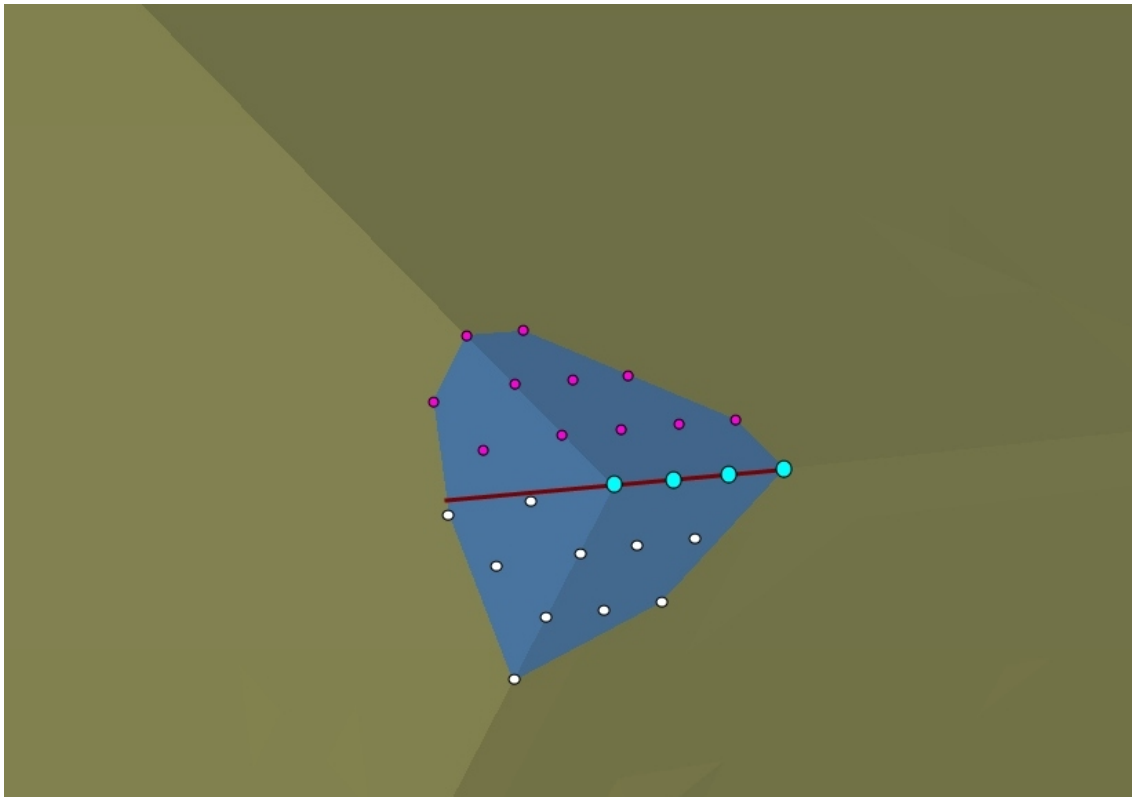


Figure 53: Feature line in a the example of the corner of a cube

4.4 RESULTS

We implemented the sharp feature surface reconstruction method presented in the previous sections and tested it with a wide range of datasets. From point clouds, sampled from known geometries like simple cubes to bilinear surfaces with varying sharp angles, as well as complex models like the 'fandisk', the 'trimstar' or the 'octaflower'. We also tested the robustness of the method with respect to noise and show some comparisons to other sharp feature preserving methods. The following Figures show some results of different data sets. Including a comparison to the smooth reconstruction and the original data. Figure 54 shows the fandisk. the left side shows a smooth MLS reconstruction, the middle the result of the feature detection and the right side the result of the sharp reconstruction.

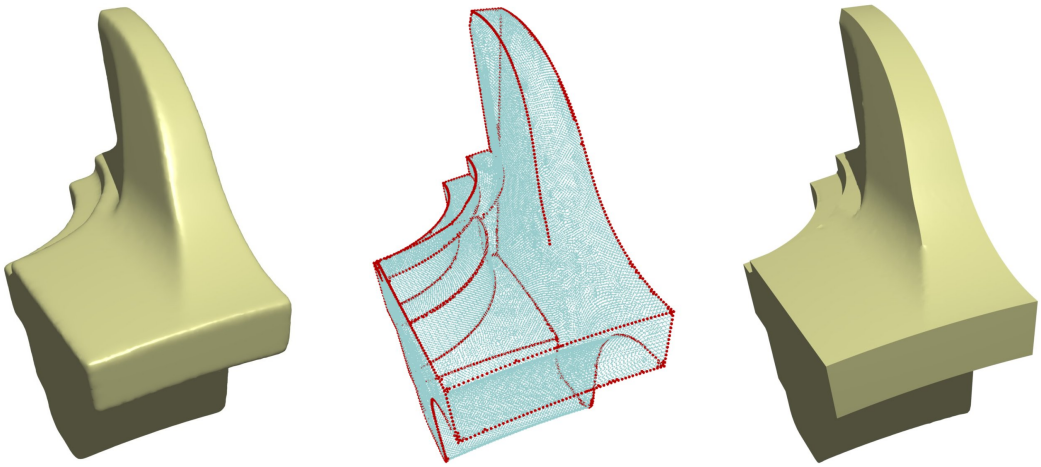


Figure 54: Reconstruction of the Fandisk: Left: smooth standard MLS reconstruction. Middle: feature points detected. Right: sharp reconstruction.

Another examplary reconstruction is shown for the trimstar in Figure 55. As before the left side shows the smooth reconstruction, the middle the sharp features and the right side the sharp reconstruction.

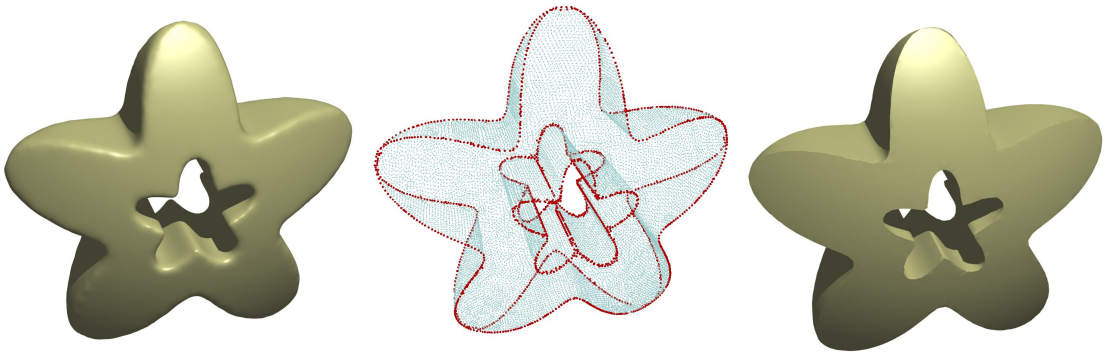


Figure 55: Reconstruction of the Trimstar: Left: smooth standard MLS reconstruction. Middle: feature points detected. Right: sharp reconstruction.

The next figure shows a reconstruction on a noisy real world data set of a scanned drill. From left to right, Figure **??** shows the original data, the smooth reconstruction, the sharp feature detection and on the right side the sharp reconstruction.

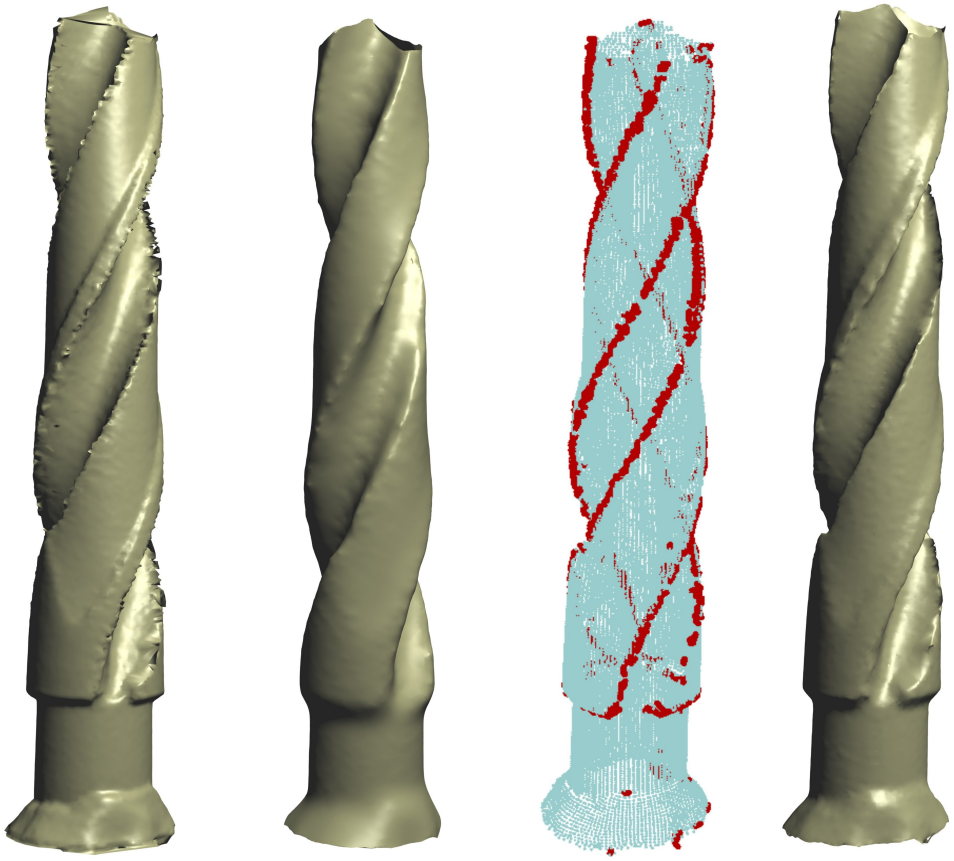


Figure 56: Reconstruction of a noisy drill dataset. From left to right, original data, smooth reconstruction, feature detection, sharp reconstruction

As last example before we go deeper into the analysis of the method let us show some reconstructions of the octaflower in Figure 57.

Before we take a closer look at the results of the surface reconstruction and the parameters used, we will check the feature lines and their effect and robustness to noise.

4.4.1 *Feature lines: robustness w/r to noise*

For this, we will go through some examples that we can use to analyze the behavior of the feature line in the presence of noise. Like in the previous chapter we add the noise by moving the points of the dataset arbitrary inside a sphere with a maximal radius of a

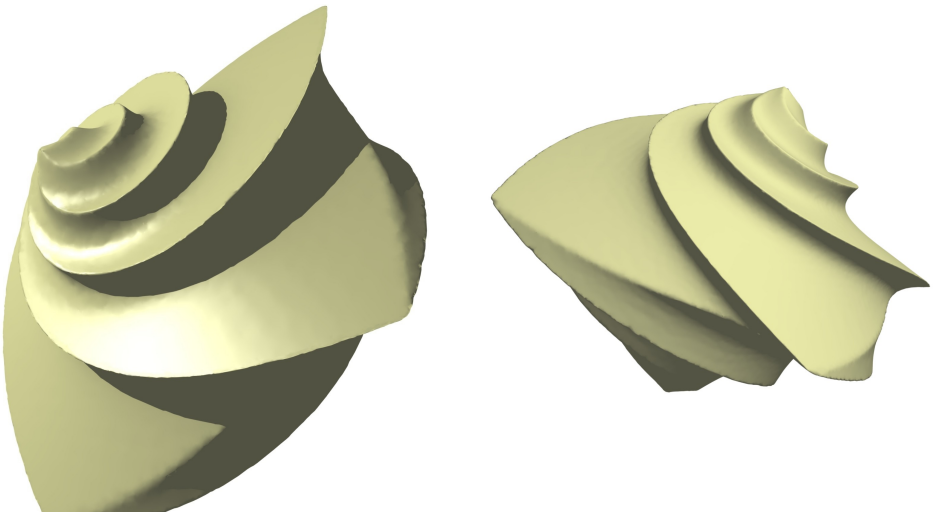


Figure 57: Reconstruction of the octaflower from different angles

given percentage of the size of the data sets bounding box around its original position. We use three levels of noise in the examples. The levels are low noise with maximal amplitude of 0.5% of the size of the dataset, medium noise with 1%, and high noise at 2%. As error we measure again the distance of the generated feature line to the correct feature line used for the construction of the data set like in Section 4.3.4.

Table 4 shows that error for two datasets. First the cube with hole example (see also Chapter 3, Figure 28) and a neighborhood in the curved area and as second example a neighborhood along the edge of a data set of two planes meeting in a linear sharp feature with an varying angle (see also Chapter 3, Figure 27). In Table 4 and Figure 58 one can see, that the error although rising at higher levels of noise stays at a reasonable level. Especially in the examples with a low level of noise, the errors stay comparable to the errors without noise. Figure 58 shows the noisy original data and the reconstructions planes example with respect to noise. One can clearly see the smoothing effect inherent to the moving least squares reconstruction especially in the non feature areas. For the planes example with high noise we added an alternative reconstruction with increased smoothing parameter of the used MLS weight function and neighborhood size($h = 0.5$,$k = 40$). The advantages of the smoothing abilities of the MLS are clearly visible in the figure and will be explained in more detail in Section 4.4.2. The example with the high noise of 2% of bounding box size gets close the limits, the methods can handle properly. One can see a beginning loss of the sharp feature reconstruction on the right side of the data set in the region with the obtuse sharp feature. The acute

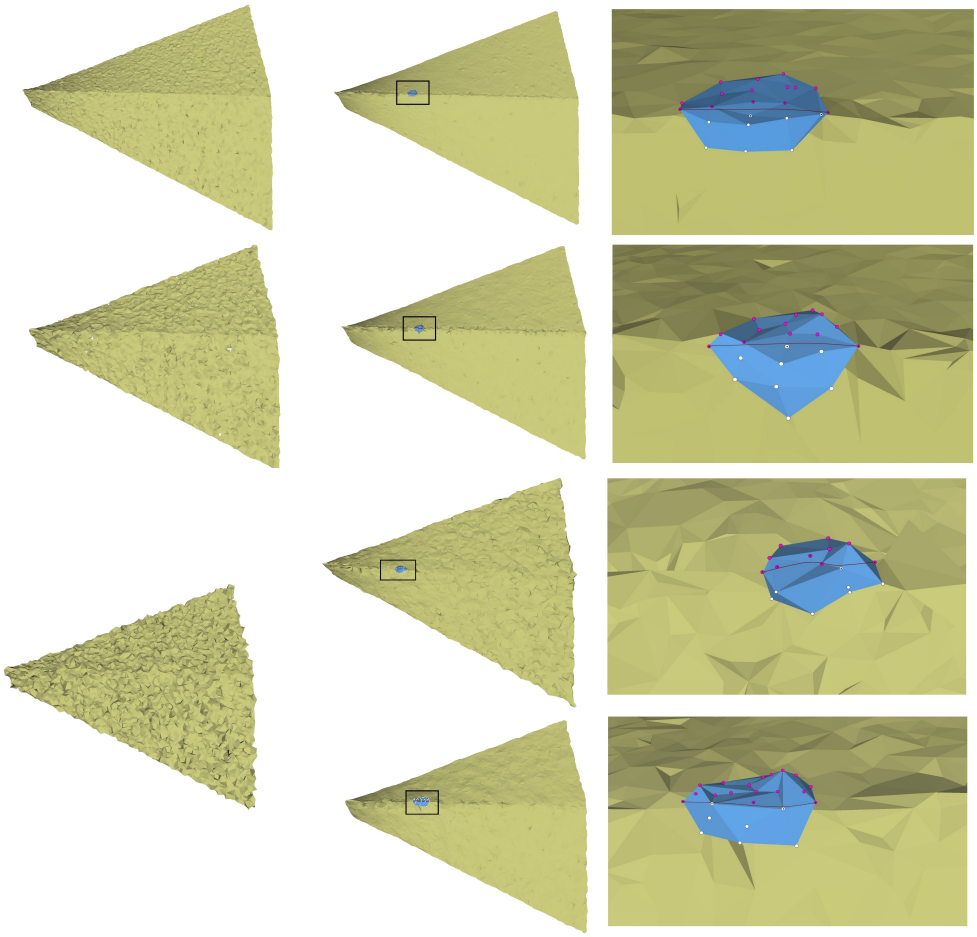


Figure 58: Noise analysis for an example using two planes meeting at varying angle. The left side shows a triangulation of the original data, the right side the reconstruction. From top to bottom the noise increases from 0.5% over 1% to 2% on the bottom row.
For the upper three examples, the neighborhood size was $k = 20$ and the smoothing parameter $h = 0.1$. For the high noise example two reconstructions are shown. One with the usual neighborhood size and MLS smoothness parameter and one with increased neighborhood size and smoothness parameter of $k = 40$ and $h = 0.5$.
As in the former images, the pink dots represent the neighborhood points used for the MLS. The feature points in the neighborhood are marked with interior dots

part of the feature is still reconstructed quite well. Although, one has to admit, that the obtuse sharp feature in the high noise data set is already hard to determine at all, in the noisy data, which can bee seen on the left bottom side of Figure 58. For the alternative reconstruction of the high noise example we increased the sizes of the neighborhoods

used for the feature detection and reconstruction from the usual used 20 to 40 neighbors. The adaption of this parameter in combination with an increased MLS smoothing of $h = 0.5$ generates a visible improvement in data sets with high noise, As the smoothness parameter generates a of course a smoothes surface, the increased neighborhood also improved the feature lines as Figure 58 shows. But this comes at the costs of higher computation times.

4.4.1.1 *Discussion on an alternative to local feature lines*

During the development process, we also thought about the use of a global feature line for the neighborhood modification. A global feature line would give additional information about the surrounding area of the neighborhood, but also has several shortcomings. First, the computation of a global feature line is much more complicated than a fast local approximation. Second, one could not use a single feature line, but a set of feature lines what makes the intersection process with the neighborhood more complex. And third, we are already using a method based on local approximations in the rest of the method. After all, the additional information, a global feature line would offer, does not lead to improvements worth the additional costs. The use of a global feature line would lead to equivalent results, but regarding computational expenses and the fact that MLS itself is based on local approximations it seems more appropriate to us to use local feature approximations instead. Since it does not have huge advantages over the local feature approximations, in the way we would use it, by larger computations costs and being more complicated, we decided to stay at local feature approximations for our method.

4.4.2 *Choice of Parameters for the MLS*

Our method offers us two parameters: The size of the neighborhood and the smoothness factor of the MLS. This section gives a short overview of the choices of the MLS parameters we have during the surface reconstruction and the influence of the parameters.

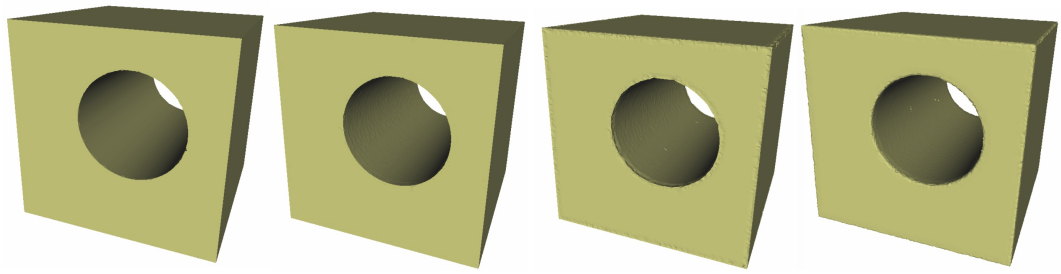
Size of neighborhood:
The first parameter we have is the *size of the neighborhood*. This parameter needs a trade off between the amount of information and computing complexity since it has a mayor influence on the computation time. The larger the size of the neighborhoods, the larger is the number of points we have for the approximation of the surface and the features.

	no noise	low noise 0.5%	med noise 1%	high noise 2%	high noise 2%
smoothness h	0.1	0.1	0.1	0.1	0.5
$\|N_p\|$	20	20	20	20	40
cube w. hole					
max Error	0.0447	0.0457	0.0880	0.0785	
min error	0.0	0.0052	0.0154	0.0352	
average error	0.0206	0.0185	0.0363	0.0591	
planes					
max Error	0.0620	0.0667	0.0712	0.1042	0.0821
min error	0.0	0.0049	0.0141	0.0442	0.0252
average error	0.0310	0.0474	0.0644	0.0752	0.0657

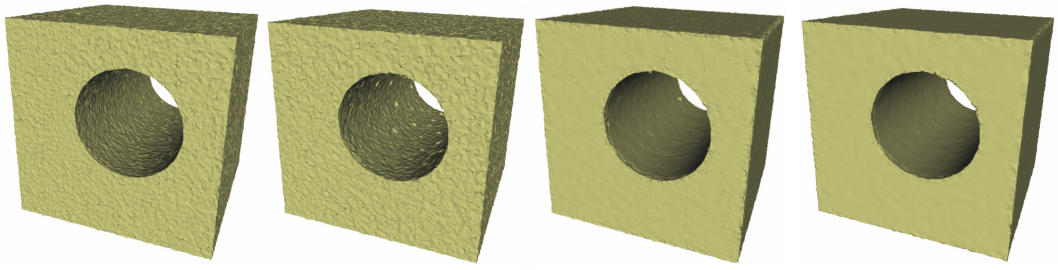
Table 4: Error analysis of a single feature line along the hole in the cube with hole dataset with respect to noise on the top and for the two planes dataset on the bottom. Measure for the error is the distance of the used feature line to the known feature line of the constructed original data. For the planes example two measurements with increased parameters are shown.

In our case especially regarding the sharp features we need at least four feature points inside a neighborhood for the feature line generation. So we have to choose the size accordingly. In the case of well and almost equally distributed samples in the data set, the k-neighborhood will be formed like a square of $\sqrt{k} \cdot \sqrt{k}$ points. If we assume the usual case of a line going straight through the neighborhood we need $k \geqslant 16$ to have at least four feature points. For most examples $k = 20$ was a good setting. In the unusual case of bad point distributions where for example the points of a k-neighborhood form itself a elliptical or almost line like structure this may lead to problems. But in those situation every reconstruction method based on k-neighborhood gets problems.

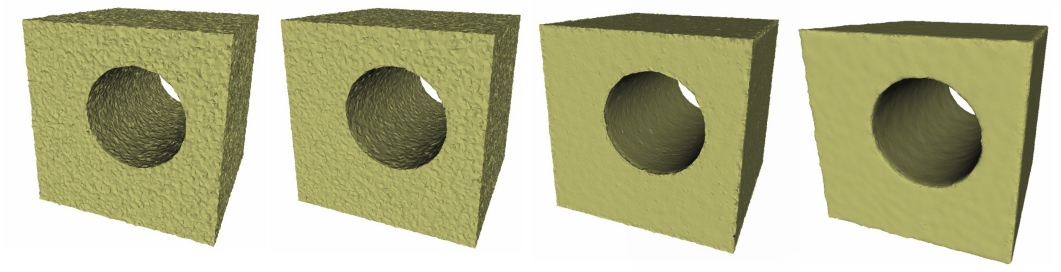
Smoothness parameter of MLS:
The other parameter we can influence is the *smoothness parameter* h of the MLS-weighting function (4.3). It influences the approximating behavior of the MLS surface. If h gets closer to 0, the MLS surface gets closer to an interpolating behavior. Thus it makes sense to increase h especially in the presence of noise. In noise free datasets h has only a minor



(a) Reconstruction with no noise



(b) Reconstruction with 0.5% noise



(c) Reconstruction with 1% noise

Figure 59: Reconstruction of the cube with hole data set with changing smoothness factor at different levels of noise ((a): no noise; (b): 0.5% noise; (c): 1% noise; smoothing factor h from left to right: Original data, $h = 0.1$, $h = 0.5$, $h = 0.9$

influence. In the noise free examples we use $h = 0.1$, in the dataset with noise will show, that increasing h can significantly improve the visual quality of the surface. Figure 59 illustrates, how h influences the reconstruction. In the middle image with $h = 0.1$ the MLS-reconstruction is close to an interpolation of the original data which is not the best choice in this noisy case. In the right image with $h = 0.5$ one can clearly see the increased smoothing effect. However, the sharp edges in the data set remain well reconstructed.

4.4.3 *Error analysis*

As measurement of the quality of the reconstruction, we used the distance between each point P_{mls}, projected onto the surface during the MLS and its nearest data point $P_{nearest} \in \{P_0, ..., P_n\}$ in the original point cloud. In Figure 60 one can see the results of two constructed datasets, we used for this issue. One dataset is once more the cube with a hole. This dataset is used since it contains curved, as well as flat areas and of course sharp edges. The other dataset is a set of two bilinear surfaces meeting at a straight feature line. The angle of this sharp feature varies from acute (45°) on the left side to obtuse (140°) on the right in combination with curved surfaces. This way, this dataset covers a wide range of possible real world situations. In the Figure 60 one can see that the reconstruction matches the original data very well. Table 5 shows the error of the reconstruction. The maximum error is only slightly above the average distance between the original data points. The reason for this is that MLS does not project points exactly onto the original data points. A large value of multiple times the distance between the data points would indicate that points were projected far away from the original surface. The very small average errors are also a strong indication that the reconstruction is quite exact.

	vertices	max. error	avg. error
cube w hole	60539	0.17	0.018
planes	22442	0.203	0.008
fandisk	39569	0.204	0.003

Table 5: Error analysis of the reconstruction. Measure for the error is the distance of the reconstruction to the original data

4.4.4 *Surface reconstruction: robustness w/r to noise*

We use the next tests to evaluate the behavior of the method with respect to noise, which is often a problem in real life. The test examples we used are:
1.) The cube-with-hole, since it has corners, convex and concave feature points;
2.) The surfaces with the varying angles having curved surfaces and acute and obtuse

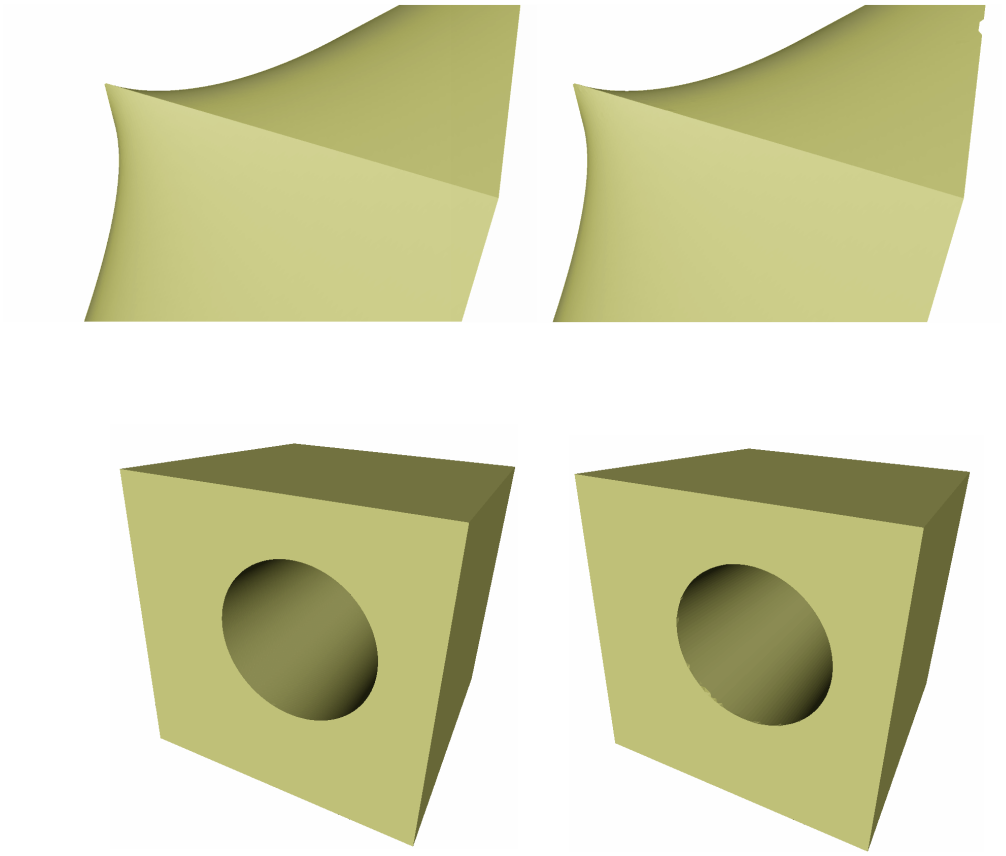


Figure 60: Original data vs. reconstructed surfaces. The left side shows the original data, The right side the reconstructions.

angles at the sharp edge.
The noise is obtained by adding a random vector to each point. This random vector is chosen in a sphere whose radius is a given percentage of the diameter of the dataset's bounding box. The results can be seen in Figure 61 and 62.
In Figure 61 we compared the reconstruction on the right side with the original noisy dataset (0.5% noise in the top image, 1% noise in the lower image). The left side shows the results of the feature detection which is the input of our method. The middle shows a triangulation of the original data and the right side shows the result of our reconstruction method. One can see a smoothing effect through the reconstruction that reduces the

noise significantly, while still preserving the sharp edges in the data set.

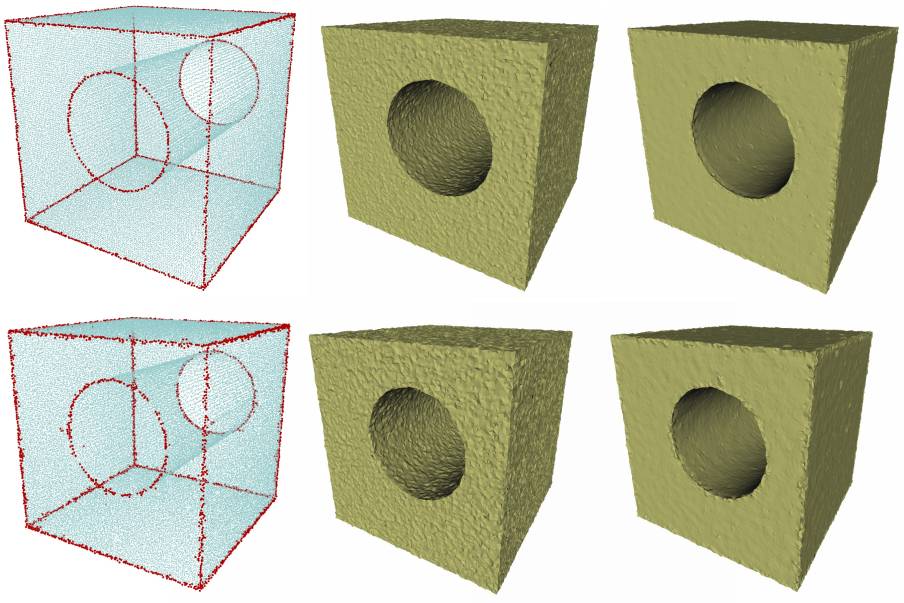


Figure 61: The left side shows result of the feature detection; the middle image the triangulation of the original noisy dataset; the right image the reconstruction of the noisy data of the cube with hole data set. The level of noise is 0.5% in the top and 1% in the bottom

Figure 62 shows the reaction to noise on the surfaces with a varying sharp feature. Also here, the left side shows the original noisy data and the right side the reconstructed surface. We used three stages of noise from 0.5% in the top, 1% in the middle and 2% on the bottom. One can see that the reconstruction is always smoother than the original surface while still preserving the sharp feature area. Even in the 2% noise example the edge is still clearly visible. But one can also see that the method reaches its limit in this case. In the region where the sharp edge meets in an obtuse angle the feature detection in no longer successful. Thus the reconstruction of the sharp edge in this region also fails. Taking a closer look at the original data one can see that the level of noise is too high to determine that there is a sharp feature at all even if one tries to search for sharp features manually. Table 6 shows the maximum and average errors. Again the distances of the reconstruction to the original data are used as error, similar to Table 5. Of course, in the noisy case, both the maximum and average error are worse than in the case without noise. But they are still on a quite low level. One reason for this is the smoothing effect of the MLS projection. It reduces the noise especially in the areas without sharp features

but it also increases the distance to the original noisy data points we used as error. So of course the errors of the noisy reconstruction are worse than in the noise free case, the reconstruction is still quite good.

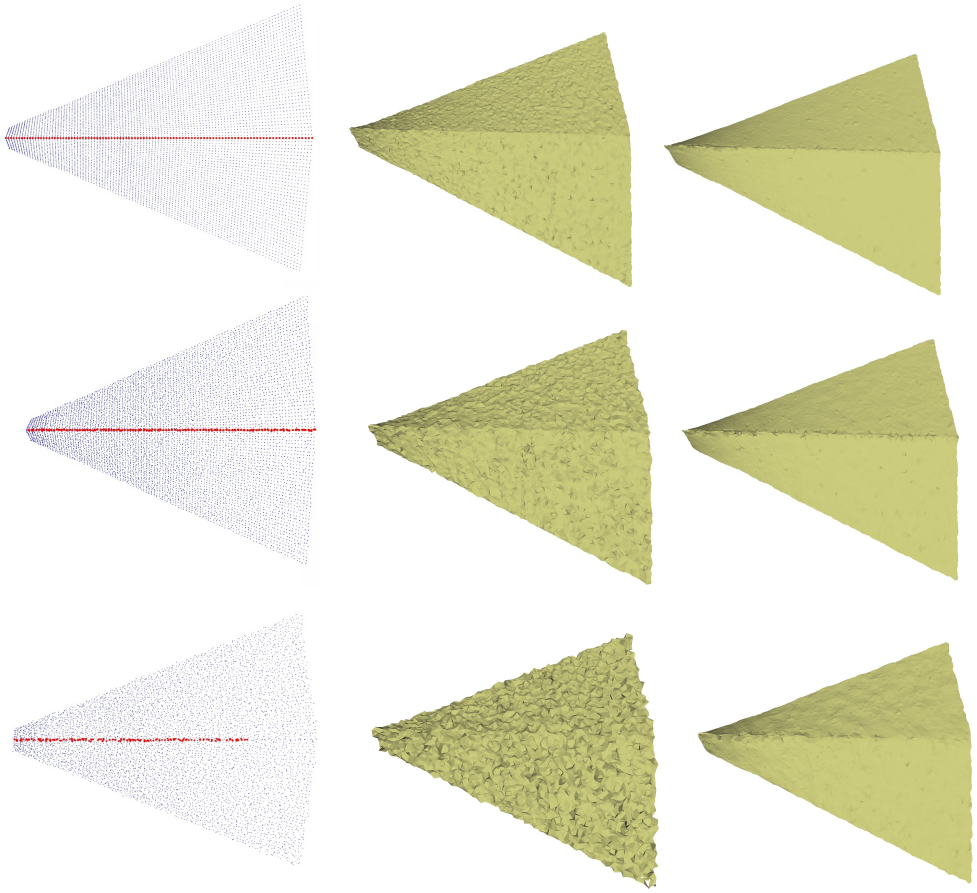


Figure 62: The left column shows the result of the feature detection, the middle column triangulation of the original noisy dataset, the right column the reconstruction of the noisy data

4.4.5 *Comparison to known methods*

We also compared our method to other reconstruction methods. For the example shown in Figure 63, 64 and Figure 65 we used the implementations of RIMLS [37] and APSS

	vertices	max. error	avg. error
cube w hole 1% noise	60539	0.43	0.0307
planes 0.5% noise	22442	0.205	0.0088
planes 1% noise	22442	0.22	0.0168
planes 2% noise	22442	0.23	0.0307

Table 6: Error analysis of the reconstruction with noise. Measure for the error is the distance of the reconstruction to the original data. The datasets used are the cube wit hole and the planes that where perturbed with noise.

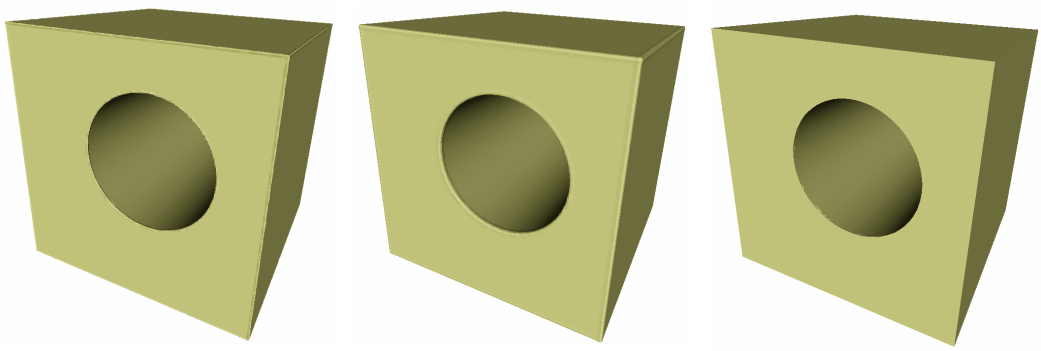


Figure 63: Comparison of different sharp feature reconstructions. the left side shows RIMLS; the middle shows APSS; the right side shows our method. (Implementation from MeshLab used for RIMLS and APSS)

[20] from the free software MeshLab [1]. One can see that our method works quite well in comparison to RIMLS. For RIMLS, we used the parameter settings that were recommended by the authors for a sharp feature reconstruction. Taking a closer look at the resulting images, one can see some differences in the sharp feature reconstructions. Our method reconstructed the sharp edges as real discontinuous sharp edge, while RIMLS shows a tendency to smooth these features a little. This is intended by the authors of RIMLS and can be seen especially in regions with obtuse sharp features as in Figure 65 on the right side of the dataset. In this case RIMLS smoothes the feature, while our method manages to keep them sharp. An even better example can be seen in the close up on Figure 64. Here the RIMLS constructs a smooth connection between the two surfaces. Our method constructs a sharp intersection in the same situation. The implementation

1 MeshLab; Visual Computing Lab - ISTI - CNR; http://meshlab.sourceforge.net

of APSS used in MeshLab unfortunately lacks the ability to reconstruct sharp features. But it can be used as an example to show the difference between a standard smooth MLS reconstruction method and reconstructions with sharp feature preservation. The figures clearly show the differences and advantages of preserving a sharp feature in these datasets. The implemented version of APSS smoothed the corners of the cube noticeable, while our method reconstructed clear and sharp edges.

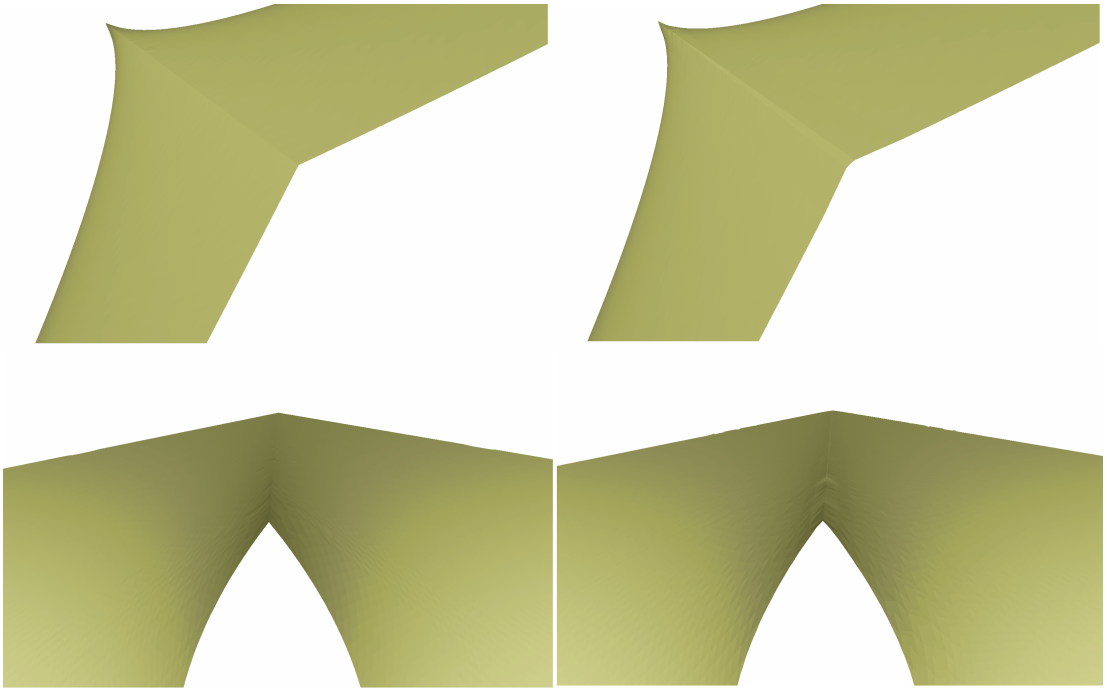


Figure 64: Comparison of the sharp feature reconstruction; the left side shows RIMLS, the right side our method.
Taken a closer look, one can see the still existing smoothing effect of RIMLS which is a side effect of its C^2-continuous reconstruction.

Another sharp feature reconstruction by Fleischmann et al. [16] was not tested with an implemented version. His outlier search and iterative surface growing is quite expensive, since one has to compute multiple local approximations of the surface. But it also leads to good results. Overall our method is a bit less expensive since most of the work, feature detection and feature clustering for the local feature lines can be done before the projection step, while the outlier search and the iterative growing of the surface parts should slow down the process.

Figure 65: Comparison of reconstructions to the original data. the left picture shows the original data, the middle shows RIMLS, the right side shows our method.

4.4.6 *Nonuniform sampling*

Another prominent problem for reconstruction methods are changes in the density of the point cloud data. Nonuniform sampling in general is also not a very big problem of the method as our test shows. Using a nonuniform sampling of the cube with hole example, the reconstruction had no further problems in a wide range of varying density. In the example in Figure 66 we changed the density of the sampling of the dataset from low on the one side to high on the other side. As one can clearly see in the Figure 66 the quality of the result is still very good. Problems occur, when the nonuniform sampling leads to regions in the dataset where the point density gets too low to identify a sharp feature. In this case the sharp feature won't be reconstructed and smoothed during the surface reconstruction.

The only form of non uniform sampling that will result in a fail of the reconstruction is the case of very high sampling in one axis in combination with very low sampling along another axis. In the extreme form of this scenario, the k-neighborhoods we use for the feature detection and reconstruction won't be sphere-like but very long stretched ellipsoids or even line-like. In these cases the surface reconstruction as well as the feature detection will fail. An example of this situation was shown in Figure 45c. Most reconstruction methods based on local surface approximations have problems in these cases.

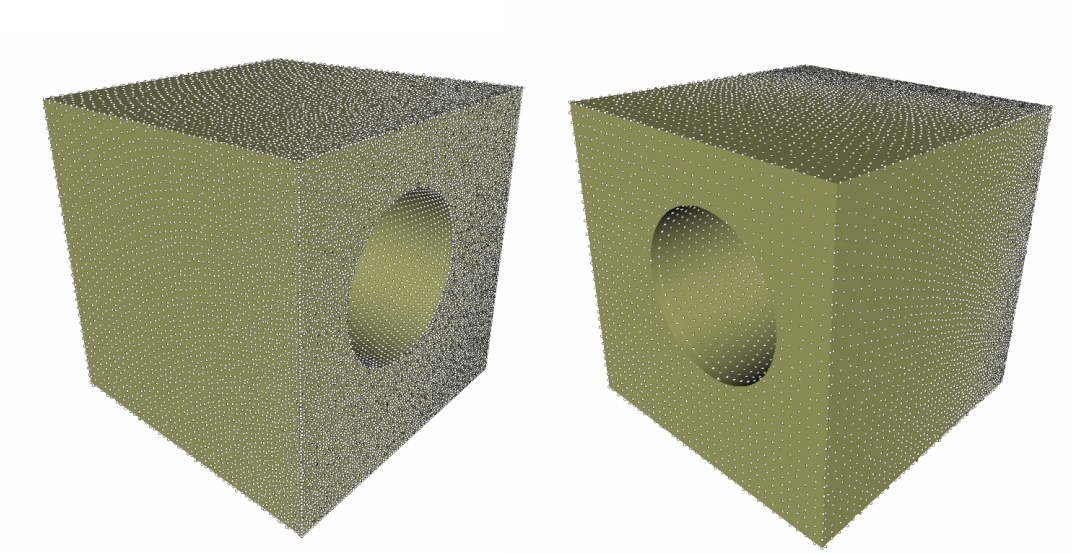


Figure 66: Reconstruction of nonuniform sampled data. the white dots are the original nonuniform distributed data points.

4.5 CONCLUSION

This chapter has shown a modified version of the moving least squares approach using a neighborhood modification to reconstruct a surface with sharp features. The method produces good results that are comparable or in many cases better than existing methods. The results are quite robust with respect to noise and nonuniform sampling. A possible way to solve those extreme cases of nonuniform sampling might be the use another type of neighborhood, since the properties of the k-neighborhood are the most important reason for a failure of the reconstruction. There is also remaining potential in the local feature line approximation method left, like the use of another approximation scheme.

Part V

BLENDING MLS-SURFACES WITH NURBS

5

BLENDING MLS-SURFACES WITH NURBS

5.1 ABSTRACT

In this chapter we are going to present an approach to combine the usage of NURBS-surfaces, e.g. from a CAD-application with a point based approach, e.g. from scanning data. We will blend these two surfaces in a smooth manner and in the end generate a single surface that combines the source surfaces. To achieve this we will sample the NURBS-surface in a first step to generate a second point cloud which we can combine to generate a single point cloud. The area where the two surfaces intersect and have to be blended will be constructed separately. The final surface is then computed as the MLS-reconstruction of the combined point clouds and the blending area. The main challenge lies in finding and generating the area where the two surfaces are blended. The idea of the blending process is shown in Figure 67.

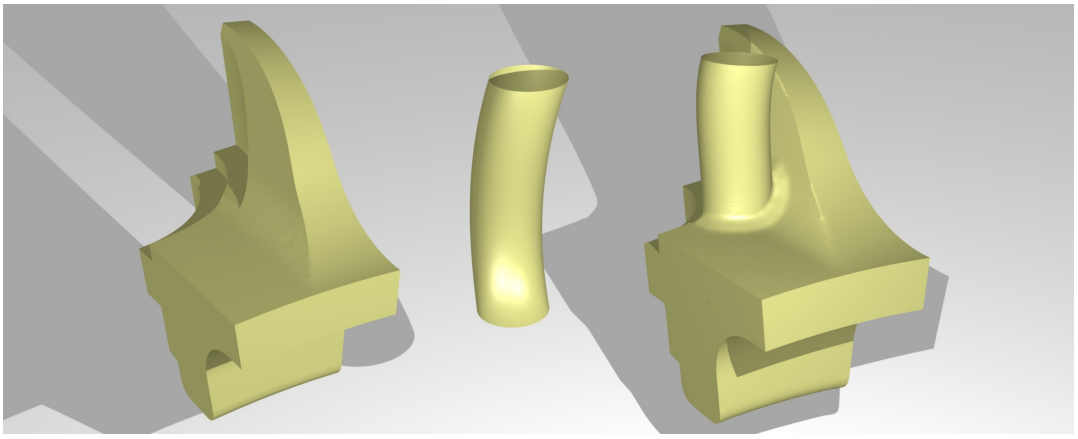


Figure 67: the blending process: the two surfaces on the left are combined, blended smoothly and form a new surface

5.2 PROBLEM DESCRIPTION

This chapter shows, how to combine a point based reconstruction of a surface via MLS with other surface types like NURBS surfaces. NURBS surfaces are usually used in CAD, while the point based reconstruction may come from a scanning device. A combination of those two data sources can be useful in a reverse engineering process. The combination of these two unrelated approaches, one being point based, the other being function based is not trivial. To combine two approaches, one has to find a global basis that allows us to transform and combine the data from both worlds. We decided to stay in the point based world, which means, we don't have to convert both datasets but only the NURBS-Dataset into a point based. This can be done by a sampling of the NURBS surface. Another problem is how to blend the two surfaces with each other. We solved this by the construction of a 'collar'-like blending area that will connect the two surfaces in a smooth manner. The width and height of the constructed 'collar' can be controlled using two parameters.

5.3 SOME RELATED WORKS

Often the blending between two surfaces in CAD is performed by filleting. In filleting one defines an intermediate surface, the so called fillet, to blend two surfaces into one smooth surface. So far our approach is quite similar. Usually filleting techniques are used to combine algebraic surfaces of the same type during the design and development of a product, e.g. blending of two B-Spline surfaces or two NURBS surfaces. Filleting between different types of surfaces like point based MLS surfaces and NURBS surfaces is not very common. One can classify the filleting methods in two groups, depending on their result. The first group results in three surfaces defined in three independent parameter domains, the fillet and the two original surfaces. The disadvantage of these methods is, that they often do not remove the parts of the original surfaces that are covered by the fillet. Also the definition as three independent surfaces with three independent parameter domains can make further designing work on the surfaces difficult. Often these techniques are referred to as *visual trimming* since the two original surfaces are not removed, but only *visually* covered by the inserted fillet. An examples for such methods can be found in [19].
The second group constructs a single surface as result that covers the two original surfaces and the fillet. So, the result is defined over a single parameter domain. This

makes those methods much more useful for designing purposes, but is also much more complicated. The two original surfaces have to be trimmed not only visually but also geometrically, making those methods referred to as *geometrical trimming*. After the trimming of the original surfaces, the fillet has to defined and inserted into the gap to connect the two surfaces. In a last step, the common parameter domain for the complete surface has to be defined.
Our approach works over all quite similar to the geometrical trimming. We also remove the overlapping parts of the surfaces, construct a gap and than close this gap with a constructed 'collar' blending the two surfaces. However, the main difference of our approach is that we can use it not only for the blending of to similar surface definitions, but also for the blending of two completely unrelated surface representations. The point based MLS-surfaces on the one hand side and the algebraic NURBS surfaces on the other hand.

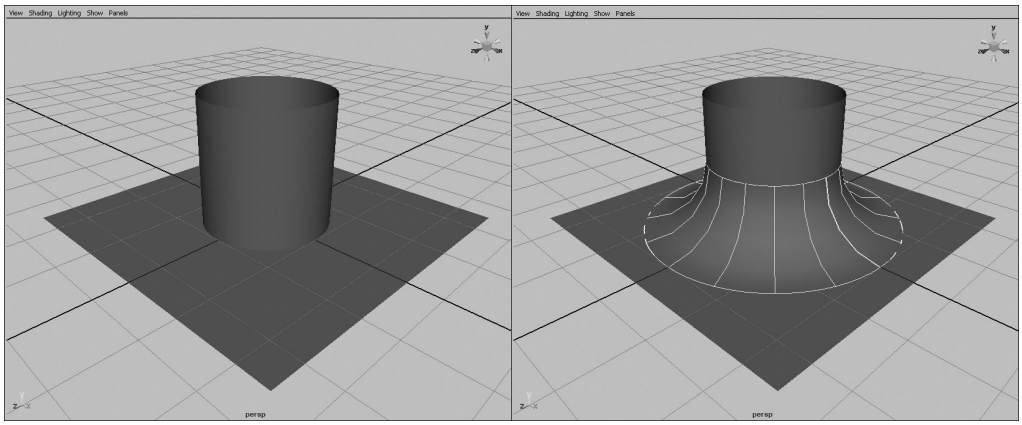


Figure 68: Blending of surfaces in the CAD software 'Maya'. Two NURBS surfaces are positioned to meet each other, then a collar like surface for the blending is added; pictures taken from www.maya-doc.com

Till now in actual CAD software the problem for NURBS/NURBS intersection and blending is solved using the before mentioned filleting methods. In a first step, the two surfaces are positioned in a way that they meet each other. In a second step a new NURBS surface is constructed, that form a collar around the wanted blending area with tangential continuity. For the CAD software 'Maya' [1] this process is shown in Figure 68.

In the recent time, not much has been done in the area of combining and blending of surfaces based on different surface definitions, like NURBS surfaces blended with a

1 Autodesk: Maya; www.autodesk.com

point based surface representations. Yang et al. [48] intersect NURBS surfaces and MLS surfaces. But in contrast to us, they work on a triangle based representation of the MLS surface. We wanted to stay at a point based level.

5.4 BLENDING NURBS WITH MLS-SURFACES

5.4.1 *Short description and notations*

During the blending process we will use several surfaces, point sets and subsets of those point sets. So let us first introduce some notations. As input we have two surfaces. One, is given by an *unsorted point set* $P = \{p_i\},\ p_i \in \mathbb{R}^3$ of N_{MLS} points representing scanned data of a surface. The second is a *NURBS-Surface* S_{NURBS}, defined in the parameter domain Ω.

$$x(u,v) : \Omega \subset \mathbb{R}^2 \mapsto \mathbb{R}^3, \Omega = \{(u,v) \in [a,b] \times [c,d]\} \tag{5.1}$$

Later we will sample the surface S_{NURBS} to receive a point set $S = \{x_i\},\ x_i \in \mathbb{R}^3,\ x_i \in S_{NURBS}$, of N_{NURBS} points.
During the blending process we construct a third point set that is responsible for the blending between S and P. We call this point set, the area for the blending or the *'collar'* $C = \{c_i\},\ c_i \in \mathbb{R}^3$.
We will also use some auxiliary point sets that represent the borders of the two point sets of P and S in the intersection area. We will call these border points α_i for the border points of the NURBS point set S and β_i for the point set P. The α_i are collected in $A = \{\alpha_i\} \subset S$, the β_i in $B = \{\beta_i\} \subset P$. In the end C will connect A and B and so close the gap between S and P.
Another important item is the definition of *'inside'* and *'outside'* of our surface. We will define the side of a surface as *'outside'*, where the blending is performed, also see Figure 70. Since we cannot make assumptions on the surface represented by point cloud P and how the user wants to blend the point set with the NURBS surface, the user has to define this. How this is done will be shown later in Section 5.4.2.1.
In the end we, construct a global point set

$$G = P \setminus \{X\} \cup S \setminus \{Y\} \cup C \tag{5.2}$$

that will be the basis for the reconstruction of a global surface via MLS. Where X and Y represent those points that have be removed from P respectively S due to overlapping

of the datasets and to generate the gap for the construction of the collar used for the blending.
Two parameters control the properties of the constructed '*collar*' for the blending. The *height* of the blending area is controlled by $t \in \mathbb{R}$, and the *width* by $w \in \mathbb{R}$. Height and width are analogue to the height and width of the collar shown in Figure 68 for the blending of two NURBS surfaces.

To blend the MLS surface with the NURBS surface we first have to decide, if the resulting surface will be a NURBS or a MLS surface. We decided to construct a MLS surface as global result. We had several reasons to do so. First, it is not a trivial task to reconstruct a NURBS surface from simple point set data. And second, and for us most important reason, the MLS reconstruction has some properties that are quite useful for the blending. Especially the property, that the resulting surface is smooth and continuous automatically.
So in the first step, we have to convert the NURBS surface into a representation that can be used as input for a MLS reconstruction. Since our MLS reconstruction from the previous chapter is based on point clouds as input we will sample the NURBS surface and construct a new point cloud that represents the blended surface of the point set P and the NURBS surface S_{NURBS}. This global point cloud can than be used as input for a MLS surface reconstruction of the complete input data. We want the blending area to be smooth and have to take special care about this. Although the properties of the MLS will result in a smooth surface even if the two point clouds are just combined without additional actions, an additional construction of the blending area will be more flexible to designing purposes and visual more appealing, since we can offer parameters that control the width and height of the 'collar' being constructed, and so the visual nature of the blended area. Another problem that occurs during a reconstruction with a simple combination of the surfaces is, that points which are enclosed by the other surface in both point sets, can disturb the reconstruction if they are not removed. In our approach we will remove those points during the blending process.

5.4.2 *Blending*

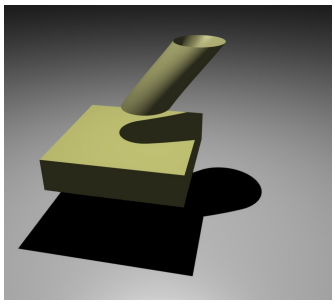
Starting with a point cloud P and a NURBS surface S_{NURBS} we have to find the area where the two surfaces intersect. There we generate an offset and create a gap between the two surfaces, that will be closed by the construction of a 'collar' C that blends the

two surfaces. The algorithm decomposes into five steps.
1. Sampling of the NURBS surface (Section 5.4.2.1)
2. Generation of an offset on the NURBS side (Section 5.4.2.2)
3. Generation of an offset in the point cloud P (Section 5.4.2.3)
4. Construction of the 'collar' C that blends the two other point sets (Section 5.4.2.4)
5. The MLS-reconstruction of the global point set $G = S \cup P \cup C$ (Section 5.4.2.5).

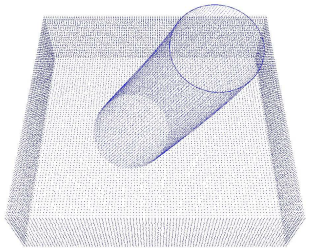
5.4.2.1 *Sampling the NURBS*

Before we can sample the NURBS surface, we have to decide on which side of the intersection the blending should be performed. This must be done by the user and can not be decided automatically. Think of a simple plane blended with a tube like in Figure 70. It is not clear if the plane should be blended with the part of the tube above or below the plane. Both way leads to possible and correct results but are perhaps not what the user intended. So, we let the user decide the starting point for the blending. For this, the user marks a border of the NURBS surface S_{NURBS}. This marks the part of the surface being blended with the MLS-surface and so we will sample S_{NURBS} from there to the intersection with the point cloud P. This side of the plane, respectively the point cloud P is now defined as 'outside'. Starting from this border, the surface will be sampled along the parameter curves, until it intersects with the point cloud P. After defining this side as 'outside' for the blending surface, the surface parts of the NURBS on the other side of the plane will be interpreted as 'inside' the surface and cut off
Without loss of generality, let us assume that the parameter domain Ω is the unit square and $(u, v) \in \Omega = [0, 1] \times [0, 1]$. Here the user selects a bounding curve for the NURBS surface, thus corresponding to the parameters being 0 or 1 at the start of the sampling, e.g. $u = 0$. The surface will now be sampled along the parameter curves starting at $(0, v)$ to $(1, v)$, see Figure 70. The sampling starts at the selected region (blue) along the parameter lines until it has offset t to the plane intersection. This results in the red curve as border for the NURBS surface. The dashed parts of the NURBS surface in Figure 70 are cut off.
The figure also shows the offsets that will be used later in Section 5.4.2.3 to generate the 'collar' for the blending.

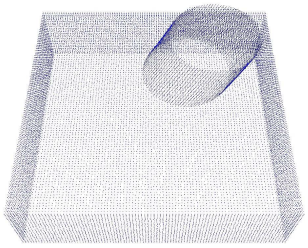
Another important characteristic during the sampling is the choice of the density of the resulting point set. A large difference in the point density of the two point sets that we want to combine will lead to problems during later reconstruction steps. We decided



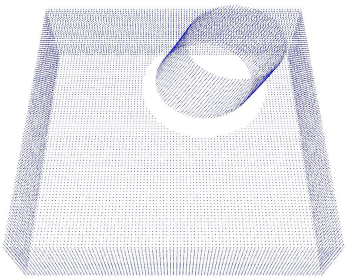
(a) The two surfaces before the blending process



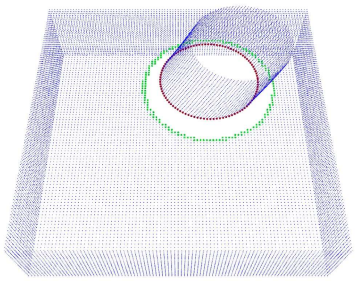
(b) The point sets P and S



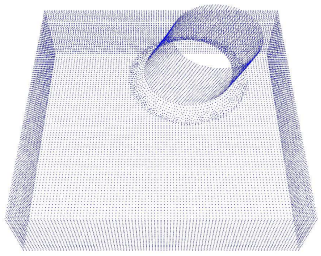
(c) Overlapping parts removed for S, offset for S generated



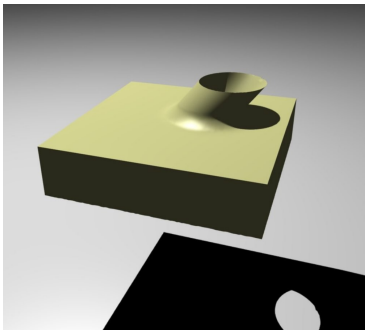
(d) Offset generated in P, data in P enclosed by S removed



(e) point sets P and S with offset and the borderlines on the NURBS side (red) and the point set side (green)



(f) The combined point set with the constructed blending area, used as input for the surface reconstruction



(g) Rendering of the blended final surface

Figure 69: Blending process

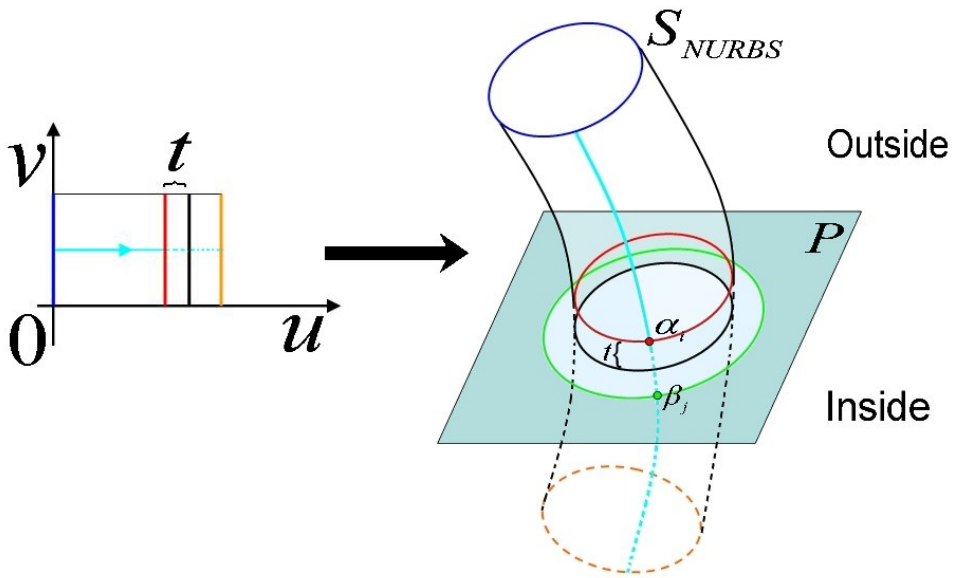


Figure 70: Example for the sampling of the NURBS surface S_{NURBS}. Starting at the selected region (blue) in the parameter domain the NURBS surface is sampled along the parameter lines until it has a distance of t to P, one exemplary parameter line is shown in cyan. The sampling along a parameter line stops if the distance to P is smaller than the parameter t. α_i is marked as border point on the NURBS side and stored. The dashed lines represent the parts of the NURBS surface that are 'inside' P after the blending and thus will be cut off. The figure also shows the offset generated in P in Section 5.4.2.3 and an associated border point β_i.

to use the average distance between points in point set P as density for the new sampled point set. Another value, for example the maximum distance, would be more sensitive to outliers in P.

Let us define the average distance between points in the point set as follows

$$d_{avg} = \frac{\sum_{i=1}^{N_{MLS}} dist(p_i, p_{nearest})}{N_{MLS}} \tag{5.3}$$

with N_{MLS} is the number of points in point cloud P and

$$dist(p_i, p_{nearest}) = \|p_i - p_{nearest}\| \tag{5.4}$$

is the euclidean distance of the point p_i and its nearest neighbor $p_{nearest}$ in P.

In the kd-tree we use as data structure for our point set, the nearest neighbor can be found in $O(\log N_{MLS})$.
We sample S_{NURBS} regularly with a point distance of d_{avg} along the parameter curves. We do this since the later MLS reconstruction delivers its best results on regular point clouds.

5.4.2.2 *Offset computation on the NURBS side*

During the sampling we also have to find an intersection curve between the point sets P and S. But we do not use the intersection curve directly for the construction of the blending area. We use a parameter t to construct an offset to the intersection curve. This way we can generate a gap between the point sets which we can use for the blending. We also have to remove the overlapping areas of the two point sets. For S this is done automatically during the sampling, since we stop sampling at the intersection with P. For the points in P that are enclosed by the intersection curve we have do apply some more work. But this will be shown in Section 5.4.2.3.
For the further explanation let us use the example of blending a cylindrical surface with the point cloud of a cube model, see Figure 69. We achieve the identification of the intersection area on the fly during the sampling of the NURBS surface to compute S. During this we will also remove the parts of the NURBS surface that get cut off by the original point cloud P. In Equation 5.2 those points belong to Y. To do so, we check the minimal distance between the new sampled point and P during the sampling process. If this distance is smaller than a given threshold, we mark this point as border point α_i and stop the sampling along the actual parameter curve. This threshold is used as parameter t that controls the height of the 'collar' and thus generates the offset mentioned earlier. All points α_i are of a distance $dist(\alpha_i, P) \geqslant t$ to the point cloud P. This generates a gap between the point sets S and P. The distance to the point cloud can be computed in $O(\log N_{MLS})$, since the point cloud is stored in a kd-tree. By stopping the sampling along the parameter curve, we also cut off those parts of the NURBS surface that would lie inside the surface of point set P. As shown in Figure 70 by inside the surface, we mean those parts of the NURBS surface that lie on the other side of point cloud P than the starting area selected by the user.

Special cases can arise. For example in case of a u-shaped tube blended with a surface. The actual algorithm cuts off all parts of the tube after the first intersection with the surface. So, if some parts of the NURBS intersect the point cloud a second time and thus

leave the surface, those parts would be cut off too. A solution to this problem is to allow the user to mark several regions as starting points of the sampling process. This will result in two independent sampling and blending processes for two separate parts of the NURBS surface.

By sampling the NURBS-surface this way, we construct a point cloud, that represents the original NURBS-surface while cutting off the overlapping parts with the original point set P. This part is shown as dashed line in Figure 70. Figures 69b and 69c show the situation before (Figure 69b) and after (Figure 69c) the removal and the offset generation. The result in Figure 69c, is a point set S that has a minimal distance of t to the point set P. The gap generated by this offset will be used in Section 5.4.2.4 for the generation of a 'collar' like surface for the blending.

During the sampling, we also receive a set of border points $A = \{\alpha_i\}$ close to the original point cloud. Those will become important in the next steps. For smoother results in the later steps, a set of border points with a higher density will be useful. So, we now sample some more border points between the existing ones if necessary until we have reached a wanted density of border points. Since the border points are defined on the NURBS surface we can compute these points via interpolation in the parameter domain. The density of the sampling has a large influence on the reconstructed result. For example Emmanuel Candès presented some work about the necessary sampling rate for an exact signal reconstruction in [7], and [8]. Although we do not need a perfect signal reconstruction, since we want to generate a smooth blending in this area, the same density as the rest of the point set will not be good enough for a good result in most cases. So, we doubled the density of the border points in A in respect to the rest of the point sets P and S to receive better results. Another reason to increase the point density in the border of the NURBS surface is that this part of the later constructed point set C, representing a 'collar' that blend the other two point sets, can be exactly computed due to its NURBS origin. The other border on the side of the given point set P, that we construct in the next section, has worse properties. This is also the reason, why we will later (Section 5.4.2.4) start the construction of the 'collar' on the NURBS side of the data and not in point set P.

5.4.2.3 *Offset computation in the MLS point cloud*

Until now, we prepared only the NURBS surface for the blending. Next we have to modify the other surface represented by point set P. We have a set of points A that

represents the border of S, see the red points in Figure 72. In the next step, we have to identify the green points in this figure. To construct a point set that blends P and S,we also need the border for point set P, the mentioned green points in Figure 72. So, we have to find a set of border points $B = \{\beta_i\}$ in P as counterpart to the set of NURBS border points A. One way to do this is to compute the nearest neighbors for the $\alpha_i \in A$ in P. During the computation of A we used the parameter t as offset to generate a gap between the point sets. We do the same here but for more flexibility we use second parameter for this side of the gap. This way we can control the height and width of the 'collar' C used for the blending independently. So, to generate this offset, we can not just use those points in P with the nearest distance to the NURBS border points α_i. We have to identify and remove those points in P, which are enclosed by S or closer to S than the width parameter. In equation 5.2 X represents those points.
Figure 71 shows how we do this. We define spheres around each NURBS border point α_i and eliminate those points in P that are enclosed within these spheres (Figure 71 left side). The size of the spheres is the parameter that controls the offset for the collar construction on the side of P and can be interpreted as the 'width' of the 'collar', blending the two surfaces. Thus, we called this parameter the 'width' w. Next, we have to eliminate points from the P that are enclosed by the collar curve in P. To eliminate those points, in our example the points of P inside the tube, we move spheres from one border point to each other and delete the points that get enclosed by these spheres. On the left side in Figure 71, spheres around the NURBS borderline (red points) are constructed. The points inside these spheres are marked (cyan points) and deleted. Now the nearest neighbors of the α_i in point set P are identified and used as borderline (green points). The right side of Figure 71 shows how we delete the interior points. This time, the sphere around one of the NURBS borderline points (red) is moved through the data of P towards all the other NURBS borderline points. To move the sphere through the data, we start with a sphere with the origin at one α_i. We compute the line $\overline{\alpha_i\alpha_j},\ i \neq j$ and compute multiple spheres along this line until we reach α_j. The points in the MLS point cloud that get enclosed by those spheres are marked and removed. We have to do this $\|B\| - 1$ times with one fix but arbitrary α_i and the $\|B\| - 1$ $\alpha_j,\ j \neq i$.

Now, after the construction of a hole in the point set P we can collect the border points in P. As initial set of border points in P, we take the nearest neighbor of each α_i in P.

$$B = \{\beta_i | \beta_i \in P,\ \beta_i \text{ is nearest neighbor of } \alpha_i\},\ \alpha_i \in A \subset S \tag{5.5}$$

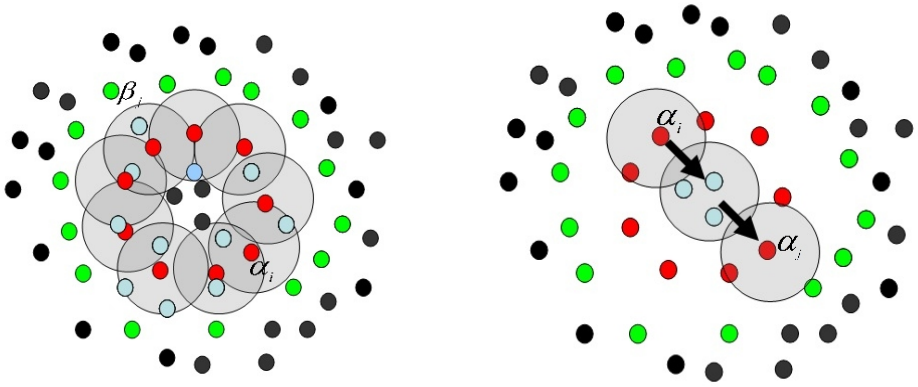


Figure 71: The principle of the borderline construction in the MLS point cloud. The left side shows the construction of the borderline, the right side shows the deletion of interior points; On the left, spheres around the border points (red, α_i) on the sampled NURBS point set S are constructed and the enclosed points (cyan) in P are removed. After this, the points enclosed by the α_i are removed by moving spheres to the other $\alpha_j \in A$ and removing of enclosed points. The border points (green, β_i) in P are than found via nearest neighbor search.

Depending on the properties and quality of P, the initial set for B has to be expanded, since it may contain gaps and holes. We also did this on the NURBS side by sampling the NURBS collar with double density, but this time we cannot sample the additional border points for B from an existing surface. To increase the density in the Border line of P, we insert points between the initial points in B via linear interpolation. Here we have to rely on the quality or the collar points α_i in A. We can do this because they are based directly on the NURBS surface and are thus precise and can be sampled as dense as necessary.
After these steps, we have two unconnected point sets $S \setminus \{Y\}$ and $P \setminus \{X\}$, each with a set of border points A and B. Between the point set, the offset has generated a gap (see Figure 72 and 69e). The next step is to connect the two point clouds by the construction of a 'collar' surface or blending surface that fills up the gap between the border points in A and B.

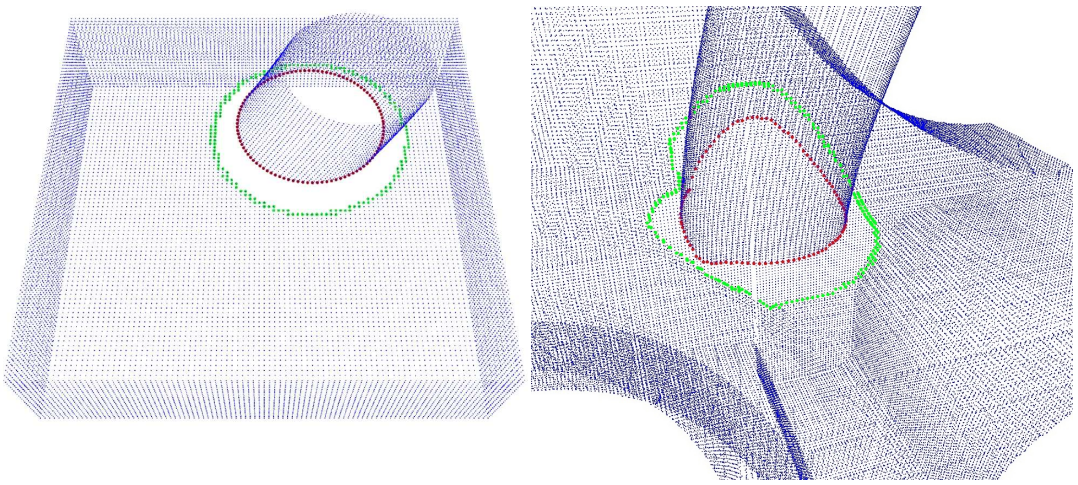


Figure 72: Pictures showing the border lines of the MLS (green) and the NURBS-surface (red) and the resulting gap between the two surfaces that will be used for the blending. the left side shows a simple example, the right side a more complex where the blending area covers a an edged and curved surface

5.4.2.4 *Construction of the 'collar'-like blending area*

For the construction of the 'collar' we have to connect the two sets of points representing the borders of the two point clouds, A and B. While the border set A of the NURBS part is a point set lying on a smooth and well defined curve, the border set B of point set P have worse properties and may contain some holes or gaps making the appearance of this side of the collar a little bumpy. In Figure 72 one can see the quality difference between the well defined border points in A (red) and the border points in B (green) in the examples. How we address this problem will be shown later.

To close the gap between A and B we must compute points inside this gap. We start on the side of A, since those points are exact samples from a smooth surface. For every point α_i in A, we compute the k-nearest neighbors in the border set B of point cloud P, see Figure 73. Here, k can be chosen very small, a large k will lead to a large oversampling in this area. We chose $k = 3$. The nearest neighbors can be computed in $O(N_B \log N_B)$. Since $N_B << N_{MLS}$ is very small in comparison to the whole dataset P this computation is quite fast. Now we interpolate linearly between α_i and its k nearest neighbors in B to insert points in between that will form the 'collar'. Figure 73 shows this step. The number of points computed between A and B is related to the size of the gap, and

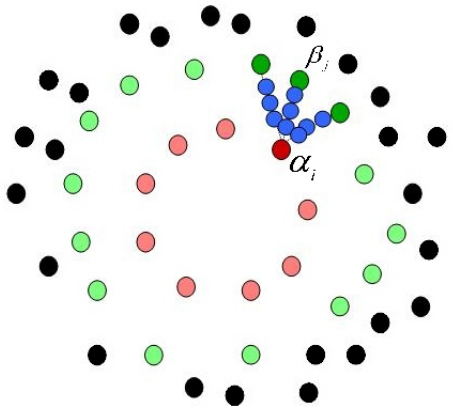


Figure 73: Principle of the collar construction; for each $\alpha : i$, $k = 3$ neighbors in B are determined and additional points are interpolated in between.

should be significant higher than the density in the two point clouds. If we do so for every α_i of the NURBS border, we receive a relative large number of points in the gap area. Those points are collected in a new point set C that will later form the 'collar' for the blending.
The distribution of the points in C is not very well at the moment. Due to the linear interpolation the points on the 'collar' form line like structures connecting the sets of border points A and B like in Figure 75 on the left side or in Figure 73. A surface reconstruction of these points will result in a surface which forms stripes and ripples. Even a smoothing MLS-reconstruction will not eliminate these artifacts completely but will only reduce the intensity of these ripples. The final surface will look like in Figure 74.

To eliminate these artifacts, we first oversample the region and then start a thinning process to reduce the density of the point set to a reasonable level. The goal is to construct a point set with smoothly distributed points that can be used as a good basis for a smooth reconstruction. In a first version we only increased the number of points used to generate the 'collar' to reduce the problem. However this improved the situation, but could not eliminate the artifact complete. Although, the large number of points inside the 'collar' and the resulting higher point density in this area also produce some new problems during the MLS reconstruction. One of these problems is, that MLS uses k-neighborhoods during the reconstruction. So, the neighborhood sizes in number of points is fixed during the MLS, but the real physical size of the neighborhoods and

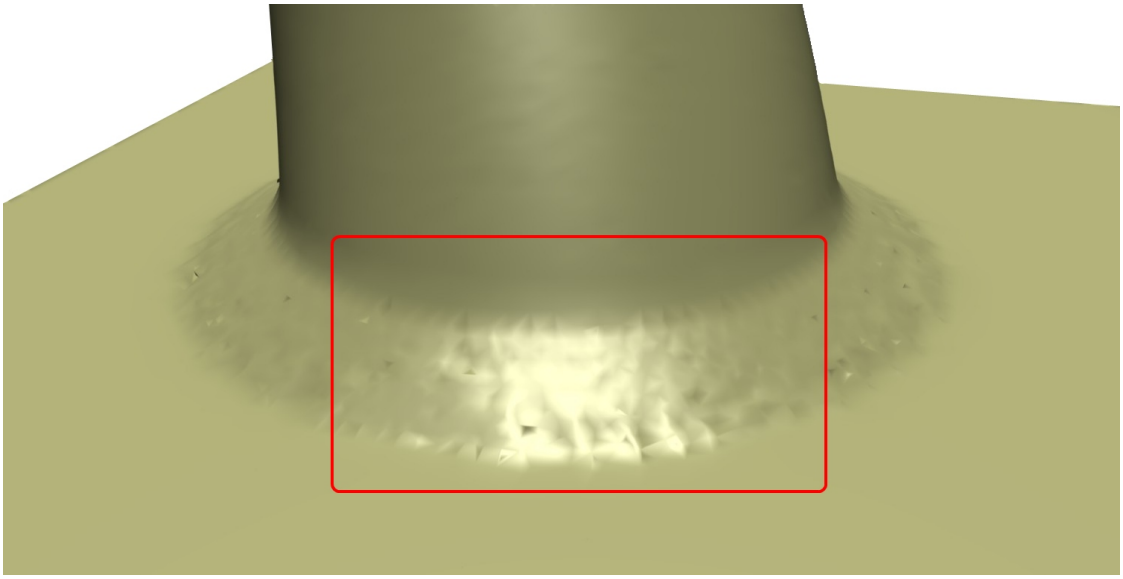


Figure 74: The ripples and artifacts after reconstruction

thus the influence of a single neighborhood on the local MLS approximation would be smaller in the area of the 'collar' than in the rest of the point cloud. This leads to a worse behavior especially in the area where the high density 'collar' and the normal density point clouds connect each other. Achieving a smooth and good looking surface in these regions gets more complicated.
Another problem is that the large number of points also increase the computational costs without increasing the surface quality significantly. Increasing k to use larger neighborhoods during the reconstruction in the collar area would reduce and eliminate most of the quality issues but only at higher additional computational costs. So we decided to think of another way to solve the 'collar' problem.
For this we considered what kind of point distribution and preconditions are optimal for a point set to generate a smooth MLS surface. The result is a point set, where the points are almost equally distributed over the whole point set. In those cases even a relative small number of samples can generate a good surface. So, for a high quality, easy to implement and fast constructed blending surface we need the collar to be almost uniformly sampled with the density of points being almost similar to the other two point sets P and S. Now we have two possibilities to achieve this.
The first possibility will be the construction of an equally sampled collar between the border curves of the two surfaces. The second is a thinning of an oversampled point

set until the wanted density is reached. Starting only with the point coordinates in A and B, we decided to use the second possibility and thin out the dense sampled point set constructed like the 'collar' mentioned above that is forming the ripples. We even increased the number of samples inside the collar generating a large oversampling of the 'collar'. The next goals is to remove the unnecessary points from C. First, this reduced the number of points and thus the computation times, and second, we can achieve a more suitable distribution of the points in C. In the end we want a point density in C that is similar to the two other point clouds P and S and almost uniform.
Subsequently we have to start a thinning process. Thinning is a common technique in computer graphics and often applied to generate coarser meshes for example for level of detail representations. Some techniques are shown for example by Dyn in [13]. A mesh free thinning technique is presented by Dyn in [14]. But most of those common techniques try to keep the error to the original data as low as possible. To achieve this, those methods remove always those points from the dataset that have a minimal effect on the original surface. This is not what we want to achieve and has some disadvantages to what we will do. First the thinning needs an initial reconstruction that is used as basis to determine the point that has to be removed in the next step. This point is the point that produces the minimal error in the surface. This means in our case, that the ripples as 'properties of the original data' will not be eliminated as we wanted since the error becomes bigger by the elimination respectively smoothing of the ripples. So, we use a technique that does not take care of errors in respect to the original data, which would be the distance of the new surface to the original oversampled surface. We do not want the final surface to be as close to the original surface as possible and keep the features of the surface. In fact, we want to increase the error to the original surface and smooth the features. We only care about the distribution of the resulting points in the collar.

Starting with one random collar point $c_i \in C$ we remove the other collar points in a sphere around it. To achieve a distribution of points similar to the other point clouds, the radius of the sphere must be adapted accordingly. We already know the radius of this sphere from the average point distance in point set P. It is the same distance we already used during the sampling of the NURBS surface in Section 5.4.2.1.

This way, we iterate through the list of the 'collar' points and remove points nearer than this given distance from C. By removing the points directly from the set of 'collar' points, we can not only make sure that those points no longer disturb the surface reconstruction but also speed up this step, since the number of elements in C gets significantly smaller

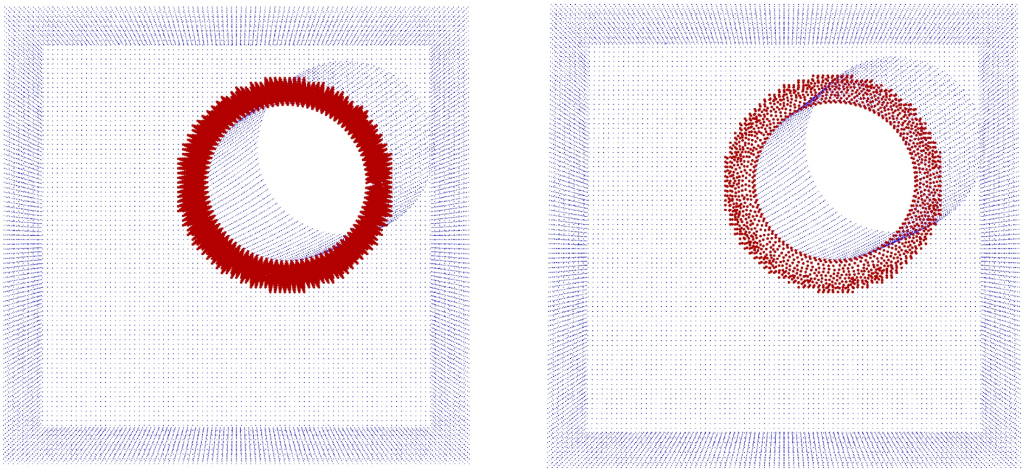


Figure 75: The oversampled 'collar' and the final 'collar' points after the thinning process

during each step. After this, we have a nice and relative uniform distribution of points in the collar leading to a better, smoother result of the MLS reconstruction.
This step also smoothed some of the errors generated by the properties of the border line of P. The Figure 75 shows the points of the 'collar' before (left side) and after (right side) the thinning process. The stripe like point distribution that forms the ripples during the reconstruction could be eliminated completely using this thinning process.

The result of the MLS-reconstruction after the thinning process can be seen in Figure 76. In comparison to the first results from Figure 74 one can clearly see the improvements and the elimination of the ripples in the 'collar'.

5.4.2.5 *Generation of the surface via MLS*

The last step of the blending process is the reconstruction of the global surface G. To reconstruct this final surface, the point set P, the NURBS point set S and the blending point set C are combined and a standard MLS reconstruction can be applied on the combined point set. Although we used only linear interpolation to close the gap between P and S, the blending area of the global surface will be smooth due to the properties of the MLS reconstruction. Of course, for a reconstruction without smoothing of sharp features in P and S we can use our method from Chapter 4. More details on sharp features will be discussed in Section 5.4.3.

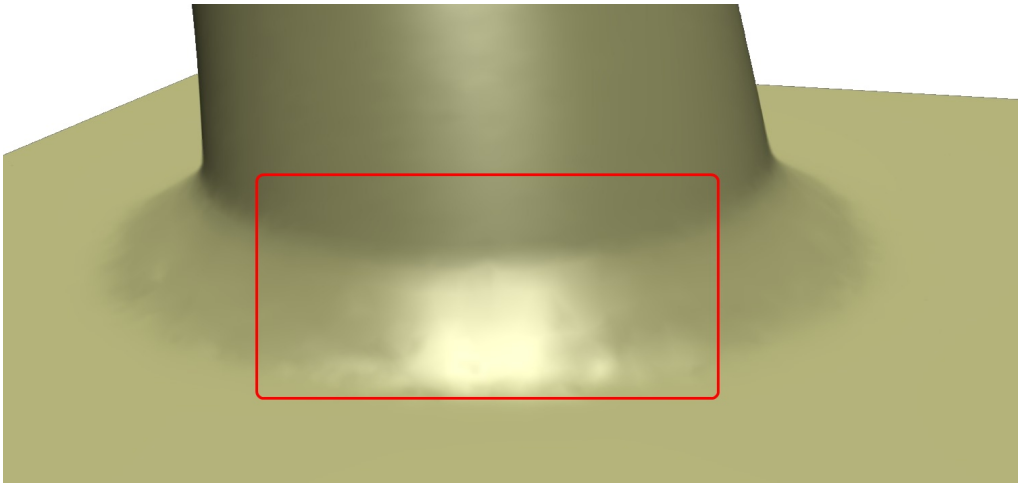


Figure 76: The result of the method: a smooth blending area without ripples

5.4.2.6 *Optimization in 'close curves'*

We also added some optimization if some special case appear in the data sets, that may impair the result. Problems can arise, if the surface of P and the NURBS surface S intersect in an acute angle. In this case the region used for the blending has to form a close curve. So the gap might be too small and the result might not look very smooth after applying a triangulation. So in this case, we use a closer sampling of points for the reconstruction resulting in smaller triangles in the triangulation. In addition we increase the parameters t and w that determine the height and width of the blending region. This leaves us more space to construct a smoother blending. These actions combined result in a smooth surface even in those special cases. Figure 77 shows the positive effects of this optimization step. Another example for the blending in a more complicated close corner can be seen in Figure 79 in a fandisk example the smooth blending in the quite sharp corner is marked in red.

5.4.3 *Combination with sharp features*

As mentioned above, the blending procedure can be easily combined with the sharp feature reconstruction from Chapter 4. In case of a sharp feature reconstruction we decided, that the blending region C is always reconstructed in a smooth manner i.e. not as sharp feature. There is no difference in the result, if the order of blending and feature

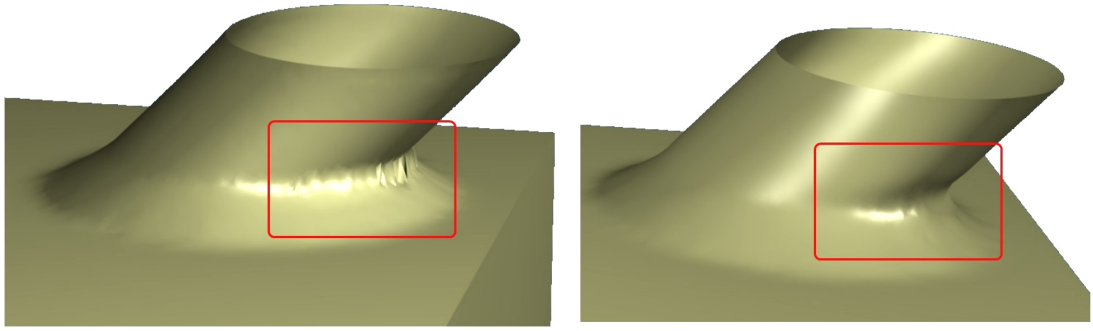


Figure 77: Optimization in close curves: The left picture shows the normal approach in a close corner; the right picture shows the same data after the optimization

detection is changed. Even if the feature detection is applied before the blending and if in a possible case of a bad dataset some points in the blending region are marked as features we gave the blending a higher importance than the feature reconstruction. So a sharp feature point that is also marked as 'blending point' in C is ignored during sharp feature reconstruction. This was an efficient way to avoid unwanted sharp reconstructions in the blending region. The way we finally took was:
First we do the blending step, i.e. the combination with the NURBS-surface and the construction of the 'collar' as mentioned above in this chapter. After this we apply the sharp feature detection from Chapter 3 on the combined blended point cloud. And at last we perform the sharp feature MLS reconstruction shown in Chapter 4.

5.5 RESULTS

Let us now take a look at the results of the blending procedure. As Examples we use a tube defined as NURBS surface blended with some of the known datasets from the previous chapters.

5.5.1 *Examples*

One example is the simple point cloud of the cube. Figure 78 shows the reconstruction with a smooth blending zone keeping the sharp edges and corners of the original data

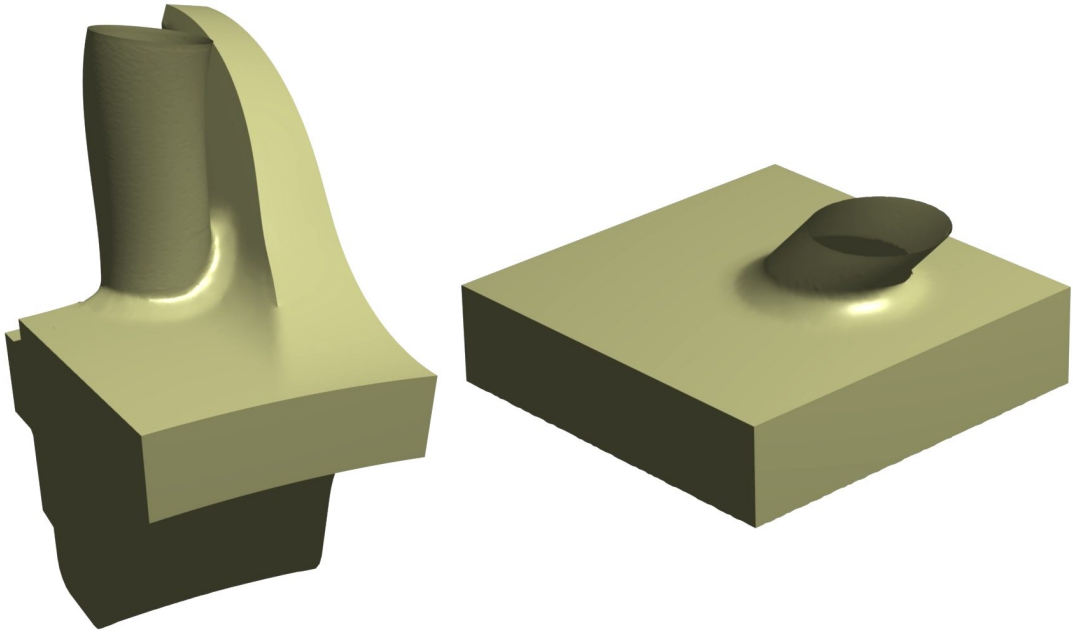


Figure 78: The examples fandisk and box blended with a tube

set. Another example is the fandisk as a more complicated real world example of a constructed and than scanned surface. The intersection area we choose in this example is much more complicated and the blending area covers not only a flat area like in the cube example, but goes around sharp corners and into a smooth area at a relative close curve. Figure 78 shows the example of the fandisk blended with a tube on the left side and a simple cube blended with a tube on the right side.
The next figures 79 and 80 show close-ups of the fandisk example from different angles to show the details of the blending area in the complex intersection area. One can see quite well the properties of the reconstruction. The red marked areas mark the intersecting parts of the blending zone of the tube and the fandisk. Especially in the smaller red area in Figure 79 one can see the smooth blending performed even in close corners following the optimization in Section 5.4.2.6. The area marked with cyan shows the conservation of the sharp features and the area of the changeover into a smooth feature. Figure 80 shows the more complex side of the blending area from a different angle. The tube intersects the fandisk not only from the top, but also from the front making the smooth blending more challenging than in the easy case of the cube and the tube. Here the blending has to covers a sharp edge as well as the smooth and curved part of the fandisk.

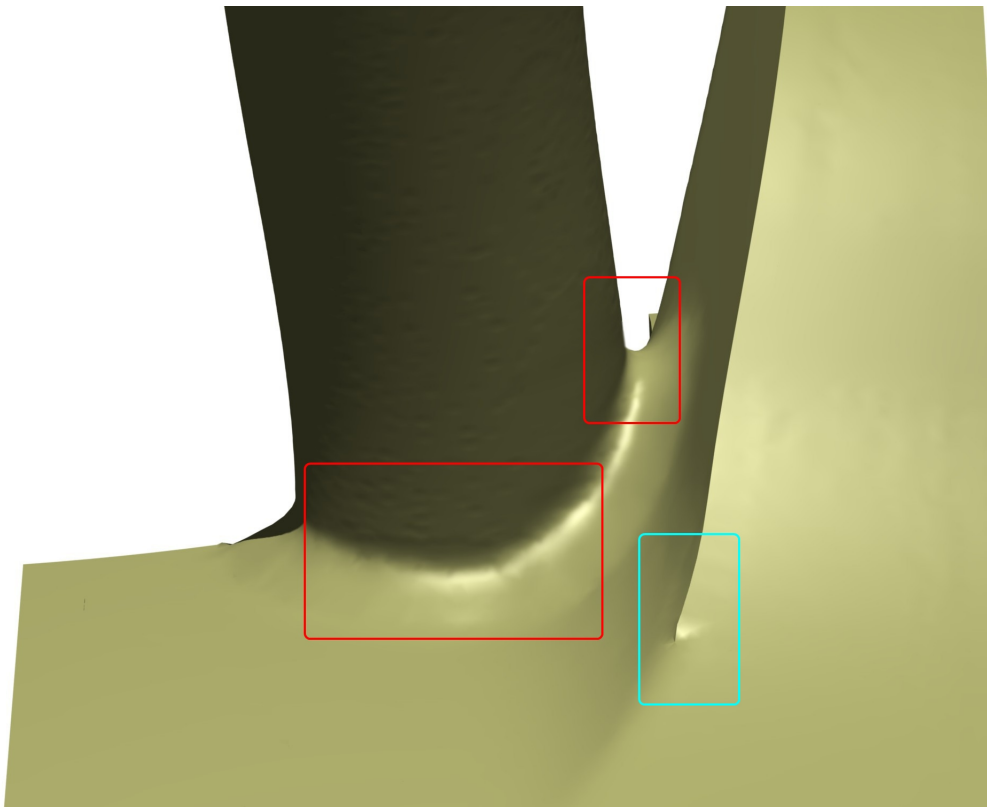


Figure 79: Close-up of the fandisk example. The red areas show the properties of the blending area between fandisk and tube. The area marked in cyan shows the transition of sharp and smooth feature during feature detection and reconstruction

5.5.2 *Varying the 'collar'*

There are several ways to modify the size and the look of the generated collar. Two parameters t and w control the collar generation. The first is the distance during the search for the cut of the NURBS-surface and the point cloud. This defines the 'height' t of the collar. The other parameter is the offset for the deletion of points in the point cloud which is equivalent to the diameter of the spheres used in this step. This parameter controls the offset for the blending area in the point set P on the MLS side. One may call it the 'width' w of the collar. In the following figures 81, to 87 different settings for these values are used to show some of the possible results. One can vary from small and low 'collar' (Figure 81) to a wide and high 'collar' (Figure 87). One can see, that

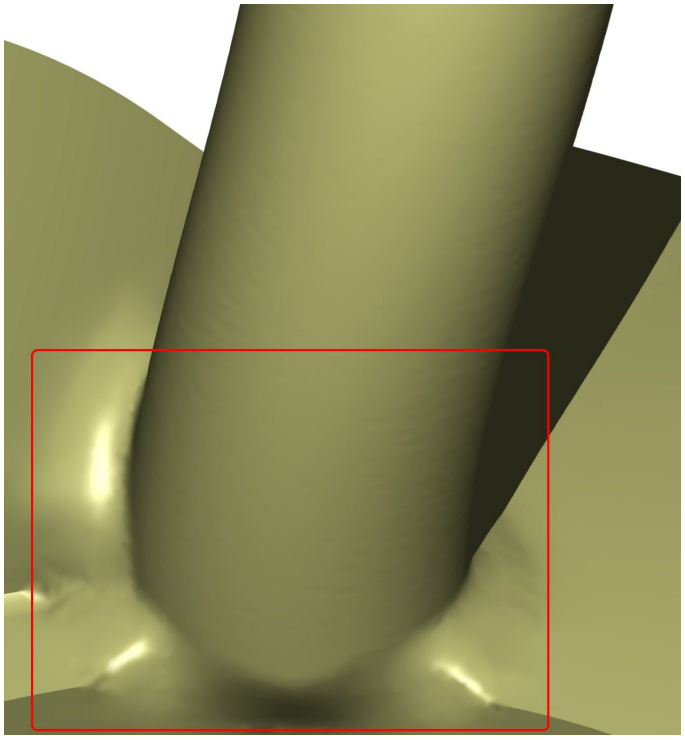


Figure 80: Another close-up of the fandisk example showing the complicated blending area over an sharp edge into a smooth curved wall.

the possibilities have a wide range for a large freedom in the design the user wants to achieve.

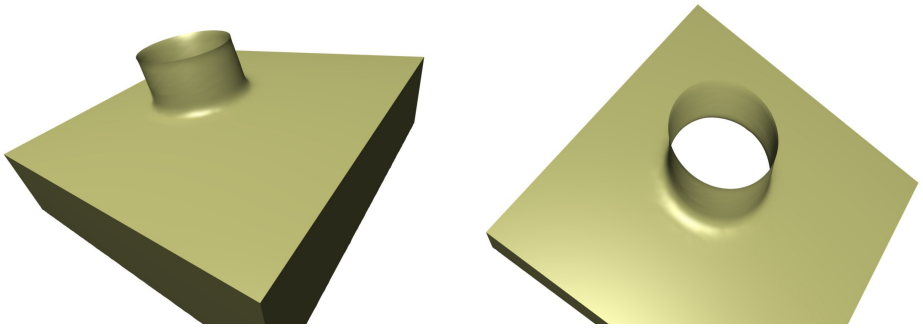


Figure 81: showing different settings for the construction of the 'collar'. Varying the size of the spheres for the border generation

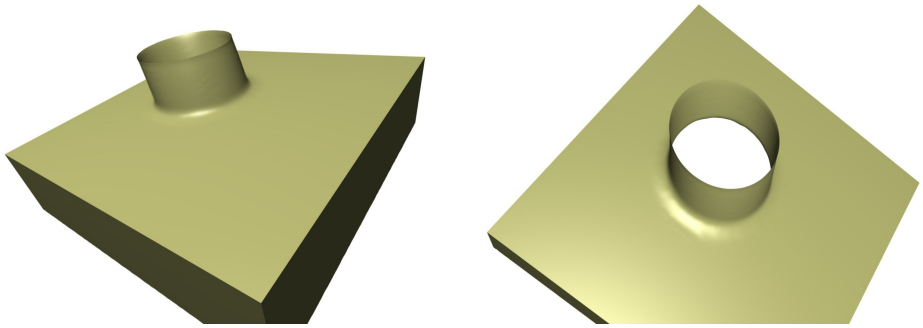


Figure 82: showing different settings for the construction of the 'collar'. Varying the size of the spheres for the border generation

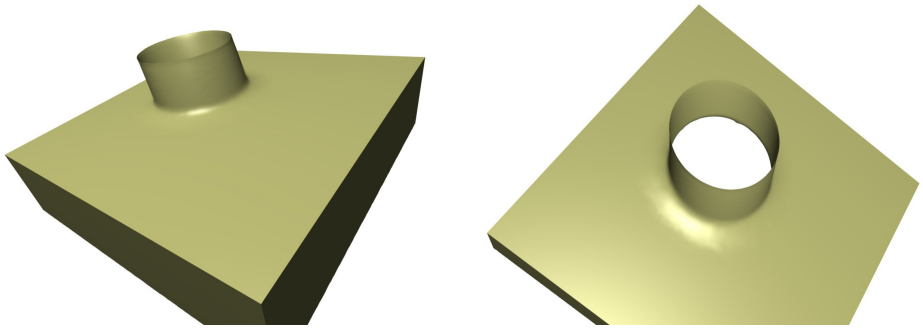


Figure 83: Different settings for the construction of the 'collar'. Varying the size of the spheres for the border generation

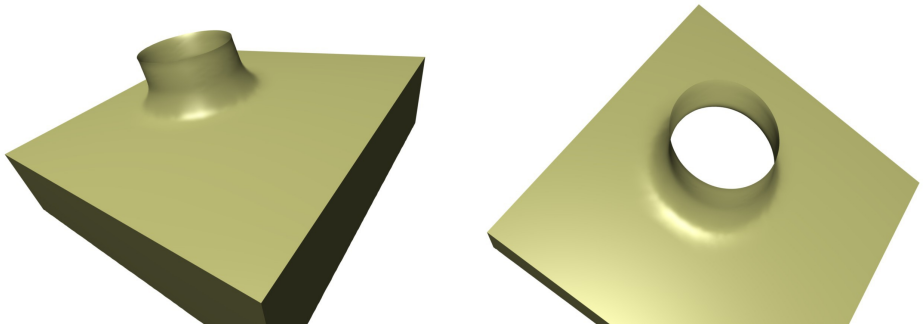


Figure 84: Different settings for the construction of the 'collar'. Varying the size of the spheres for the border generation

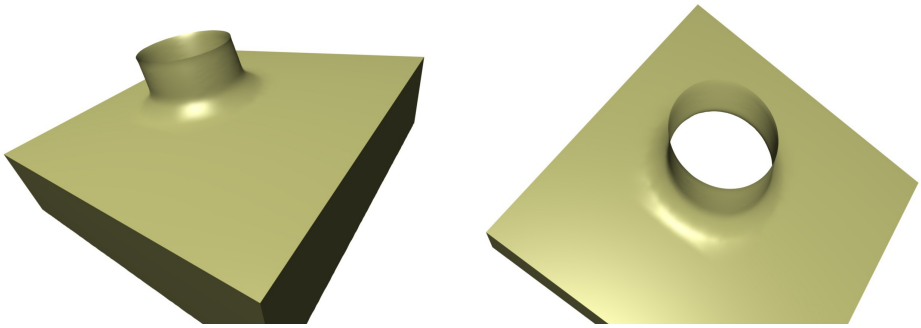


Figure 85: Different settings for the construction of the 'collar'. Varying the size of the spheres for the border generation

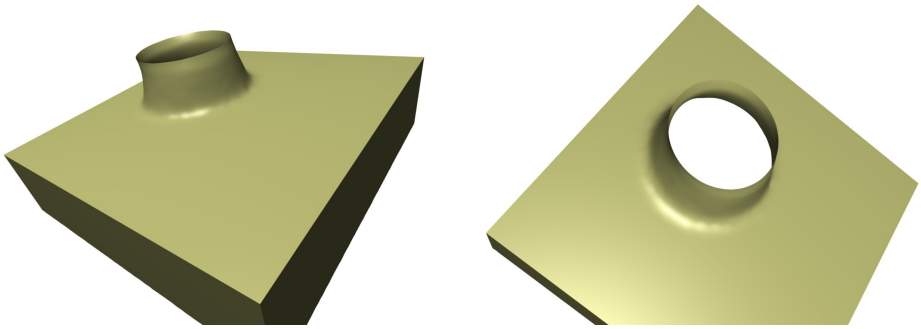


Figure 86: Different settings for the construction of the 'collar'. Varying the size of the spheres for the border generation

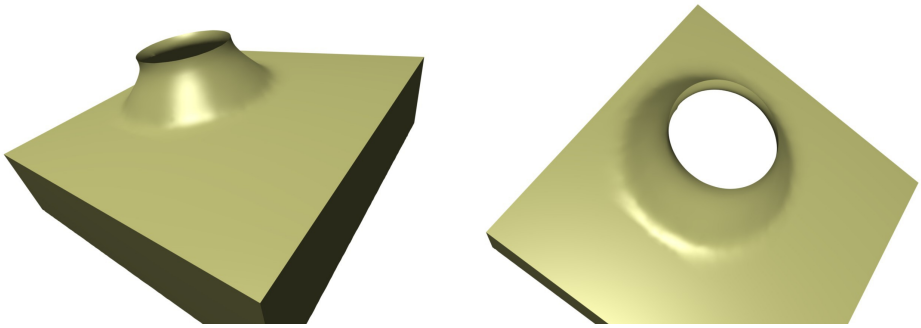


Figure 87: Different settings for the construction of the 'collar'. Varying the size of the spheres for the border generation

5.5.3 *Noise*

As in the previous chapters we tested the algorithm with respect to noise. Since NURBS-Surfaces as a constructed surface does not contain noisy data, we applied the noise only to the point cloud that represents the scanned surface. As noise we once more move the data points randomly inside a sphere of a given size e.g. 1% of the size of the bounding box. The noisy point cloud is then used as input for the algorithm and blended with the NURBS-surface.

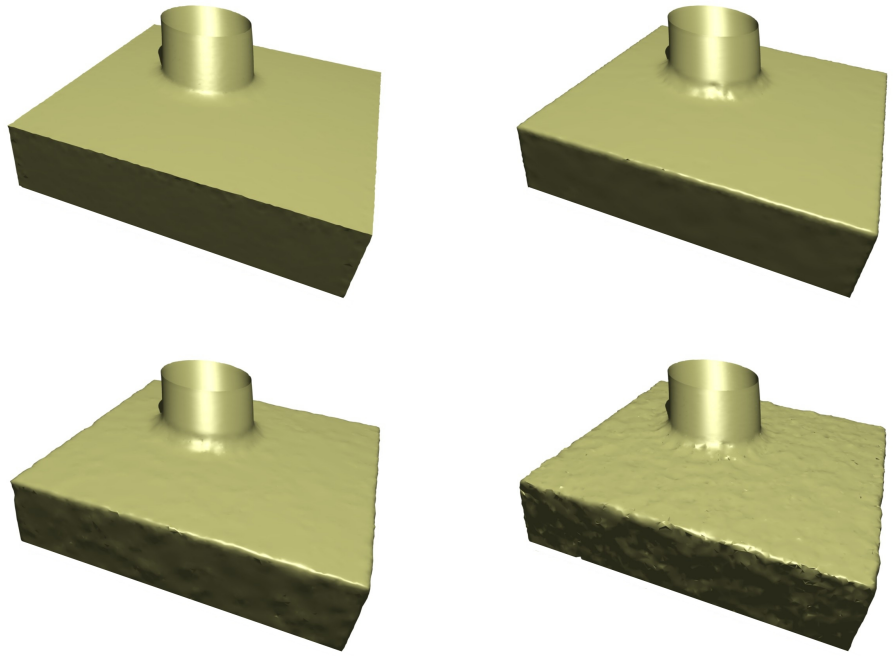


Figure 88: Results with noisy data, the upper row shows the result of two levels of low noise and medium noise; the lower row shows high and very high noise

As expected the behavior of the algorithm regarding noise is similar to Chapter 4. The results of the examples can be seen in the following pictures. On the upper row in Figure 88 one can see the result for a low level of noise at 0.1% and 0.5% of the radius of the bounding sphere of the dataset. At this level of noise, the result is still of a good quality and even comparable to the result without noise. The lower row shows the result a medium level of noise, 1.0% and high level of 2.0%. The quality starts to decrease, but the smoothing effect of MLS can still eliminate some of the noise effects at

the medium level. Especially the blending area is nearly of the same quality than before. The reconstruction of the sharp edges is still recognizable but not as good as before. At least a high level of noise of 2.0% shows that the limits of the method are, as expected, the same as for the sharp feature reconstruction. This level of noise can not longer be compensated by the reconstruction and the MLS very well. But the blending itself stays stable and produces a reasonable blending area. Although, at high level of noise the quality of the 'collar' also gets reduced significantly. The main problem for this loss of quality is the rising noise in the border points on the MLS side, making this side of the constructed 'collar' extremely bumpy. Additional smoothing of the border points would compensate this effect to a certain degree but not solve this problem completely.

5.6 CONCLUSION

In this chapter we have shown a new method to combine the point based MLS with function-based NURBS-surfaces. We can blend these two unrelated surface types in a smooth manner providing good control over the behavior and look of the blending area. In the future one might improve and extend the method more for example by adding other various surface types to this blending method. There are also some more improvements conceivable for example to increase the quality with respect to noise even more by adding some smoothing techniques to the blending area at high levels of noise.

BIBLIOGRAPHY

[1] Alexa, M., Behr, J., Cohen-or, D., Fleishman, S., Levin, D., and Silva, C. T. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics 9* (2003), 3–15. (Cited on page 24.)

[2] Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., and Silva, T. Point set surfaces. *IEEE Visualization 2001* (2001). (Cited on page 72.)

[3] Alexa, M., Rusinkiewicz, S., Alexa, M., and Adamson, A. On normals and projection operators for surfaces defined by point sets. *In Eurographics Symp. on Point-Based Graphics* (2004), 149–155. (Cited on page 12.)

[4] Amenta, N., Choi, S., and Kolluri, R. K. The power crust. In *SMA '01: Proceedings of the sixth ACM symposium on Solid modeling and applications* (New York, NY, USA, 2001), ACM, pp. 249–266. (Cited on page 40.)

[5] Attene, M., Falcidieno, B., Rossignac, J., and Spagnuolo, M. Sharpen&bend: Recovering curved sharp edges in triangle meshes produced by feature-insensitive sampling. *IEEE Transactions on Visualization and Computer Graphics 11*, 2 (2005), 181–192. (Cited on page 63.)

[6] Böhm, W., Farin, G., and Kahmann, J. A survey of curve and surface methods in cagd. *Comput. Aided Geom. Des. 1* (July 1984), 1–60. (Cited on page 26.)

[7] Candes, E. J., Romberg, J., and Tao, T. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory 52*, 2 (Feb. 2006), 489–509. (Cited on page 120.)

[8] Candes, E. J., and Wakin, M. B. An introduction to compressive sampling. *IEEE Signal Processing Magazine 25*, 2 (2008), 21–30. (Cited on page 120.)

[9] Carr, J. C., Fright, W. R., and Beatson, R. K. Surface interpolation with radial basis functions for medical imaging. *IEEE Transactions on Medical Imaging 16* (1997), 96–107. (Cited on page 19.)

[10] Demarsin, K., Vanderstraeten, D., Volodine, T., and Roose, D. Detection of closed sharp edges in point clouds using normal estimation and graph theory. *Comput. Aided Des. 39*, 4 (2007), 276–283. (Cited on page 39.)

[11] Dey, T. K., and Sun, J. Normal and feature approximations from noisy point clouds. *Proceedings FST and TCS 2006* (2006), 21–32. (Cited on page 12.)

[12] do Carmo, M. P. *Differentialgeometrie von Kurven und Flächen*. Vieweg und Sohn, 1983. (Cited on page 26.)

[13] Dyn, N., Floater, M. S., and Iske, A. Adaptive thinning for bivariate scattered data. *J. Comput. Appl. Math 145* (2002), 505–517. (Cited on page 126.)

[14] Dyn, N., Iske, A., and Wendland, H. Meshfree Thinning of 3DÂ Point Clouds. *Foundations of Computational Mathematics 8*, 4 (Aug. 2008), 409–425. (Cited on page 126.)

[15] Farin, G. *NURB Curves and Surfaces*. A.K. Peters, 1995. (Cited on page 27.)

[16] Fleischmann, S., Cohen-Or, D., and Silva, C. Robust moving least-squares fitting with sharp features. *ACM Trans. Graph.* (2005), 37–49. (Cited on pages xii, 40, 72, 73, and 106.)

[17] Franke, R. Scattered data interpolation: test of some methods. *Math Computation 38* (1982), 181–200. (Cited on page 17.)

[18] Franke, R., and Nielson, G. Smooth interpolation of large sets of scattered data. *International Journal Numerical Methods Engineering 15* (1980). (Cited on page 18.)

[19] Gloudemans, J. R. Filleting of Aircraft Components Using Non-Uniform B-spline Surfaces. *Master.s thesis, VPI & SU* (1989). (Cited on page 112.)

[20] Guennebaud, G., and Gross, M. Algebraic point set surfaces. In *In Proceedings SIGGRAPH 2007* (2007). (Cited on pages xii, 74, 75, and 105.)

[21] Gumhold, S., Wang, X., and McLeod, R. Feature extraction from point clouds. *Proceedings of 10th International Meshing Roundtable* (2001). (Cited on pages 38, 51, 53, and 66.)

[22] Guy, G., and Medioni, G. Inference of surfaces, 3d curves, and junctions from sparse, noisy,3d data. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*, 11 (1997), 1265–1277. (Cited on page 40.)

[23] Hardy, R. L. Multiquadratic equations of topography and other irregular surfaces. *Journal of Geophysical Research 76* (1971), 1905–1915. (Cited on page 19.)

[24] Hastie, T., Tibshirani, R., and Friedman, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.*, 2th ed. Springer, Boston, MA, USA, 2009. (Cited on page 50.)

[25] Hildebrand, K., Polthier, K., and Wardetzky, M. Smooth feature lines on surface meshes. *Proceedings of Symposium on Geometric Processing* (2005). (Cited on pages 36 and 37.)

[26] Hubeli, A., and Gross, M. Multiresolution feature extraction for unstructured meshes. *Proccedings of IEEE Visualization* (2001), 287–294. (Cited on pages 36 and 37.)

[27] Kirby, M. *Geometric Data Analysis: An Empirical Approach to Dimensionality Reduction and the Study of Patterns*. John Wiley & Sons, Inc., New York, NY, USA, 2000. (Cited on page 19.)

[28] Kobbelt, L., and Botsch, M. A survey of point-based techniques in computer graphics. *Computers and Graphics 28* (2004), 801–814. (Cited on pages x, 12, 13, 14, and 15.)

[29] Kobbelt, L., Botsch, M., Schwanecke, U., and Seidel, H.-P. Feature sensitive surface extraction from volume data. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), ACM, pp. 57–66. (Cited on page 36.)

[30] Kolluri, R. Provably good moving least squares. *ACM SIAM Symposium on Discrete Algorithms* (2005). (Cited on pages 20 and 72.)

[31] Lee, D., and Wong, C. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica 9* (1977). (Cited on page 43.)

[32] Levin, D. The approximation power of moving least-squares. *Mathematics of Computation 67* (1998), 1517–1531. (Cited on pages 20, 24, and 72.)

[33] Levin, D. Mesh-independent surface interpolation. *Geometric Modeling for Scientific Visualization* (2003), 37–49. (Cited on pages 20, 71, and 72.)

[34] Liu, Y., and Xiong, Y. Automatic segmentation of unorganized noisy point clouds based on the gaussian map. *Comput. Aided Des. 40*, 5 (2008), 576–594. (Cited on page 45.)

[35] Mérigot, Q., Ovsjanikov, M., and Guibas, L. J. Robust voronoi-based curvature and feature estimation. In *Symposium on Solid and Physical Modeling* (2009), W. F. Bronsvoort, D. Gonsor, W. C. Regli, T. A. Grandine, J. H. Vandenbrande, J. Gravesen, and J. Keyser, Eds., ACM, pp. 1–12. (Cited on pages 39, 61, and 66.)

[36] Overmars, M. H. *The Design of Dynamic Data Structures*. Springer, Berlin, 1983. (Cited on page 42.)

[37] Oztireli, C., Guennebaud, G., and Gross, M. Feature preserving point set surfaces based on non-linear kernel regression. *Computer Graphics Forum 28*, 2 (2009). (Cited on pages xii, 40, 75, 76, 77, and 104.)

[38] Pauly, M., Gross, M., and Kobbelt, L. P. Efficient simplification of point-sampled surfaces. *VIS '02: Proceedings of the conference on Visualization '02* (2002), 163–170. (Cited on page 14.)

[39] Pauly, M., Keiser, R., and Gross, M. Multi-scale feature extraction on point-sampled surfaces. *Computer Graphics Forum* (2003). (Cited on pages 38, 51, 53, and 66.)

[40] Piegl, L. On nurbs: a survey. *IEEE Comput. Graph. Appl. 11* (January 1991), 55–71. (Cited on page 29.)

[41] Reuter, P., Joyot, P., Trunzler, J., Boubekeur, T., and Schlick, C. Surface reconstruction with enriched reproducing kernel particle approximation. *Eurographics Syposium on Point-Based Graphics* (2005). (Cited on pages xiii, 76, and 77.)

[42] Shen, C., O'Brien, J. F., and Shewchuk, J. R. Interpolating and approximating implicit surfaces from polygon soup. *ACM Trans. Graph. 23* (August 2004), 896–904. (Cited on pages 20 and 75.)

[43] Shepard, D. A two-dimensional interpolation function for irregulary-spaced data. *Proc. ACM National Conference* (1968). (Cited on page 17.)

[44] Watanabe, K., and Belyaev, A. G. Detection of salient curvature features on polygonal surfaces. *Computer Graphics Forum* (2001), 385–392. (Cited on pages 36 and 37.)

[45] Weinkauf, T., and Günther, D. Separatrix persistence: Extraction of salient edges on surfaces using topological methods. *Computer Graphics Forum (Proc. SGP '09) 28*, 5 (July 2009), 1519–1528. (Cited on page 36.)

[46] Wu, J., and Kobbelt, L. Optimized sub-sampling of point sets for surface splatting. *Proceedings of Eurographics 2004* (2004). (Cited on page 14.)

[47] Wu, J., and Wang, Q. Feature point detection from point cloud based on repeatability rate and local entropy. In *Proceedings of the SPIE Vol.6786* (2007), p. 67865. (Cited on page 51.)

[48] Yang, P., and Qian, X. Direct boolean intersection between acquired and designed geometry. *Comput. Aided Des. 41* (February 2009), 81–94. (Cited on page 114.)

[49] Zwicker, M., Pfister, H., van Baar, J., and Gross, M. Surface fitting. *In Proceedings of ACM SIGGRAPH 2001* (2001). (Cited on page 13.)

CURRICULUM VITAE

Education

1986 - 1990: Elementary School: Stresemannschule Kaiserslautern

1990 - 1996: Hohenstaufen Gymnasium Kaiserslautern

1996 - 1999: Albert Schweitzer Gymnasium Kaiserslautern

Civil Service

08/1999-09/2000: Community service at the community office of environment in Kaiserslautern

Studies

10/2000-10/2006: Diplom-Informatik at the Technical University Kaiserslautern
Degree: Diplom-Informatiker (Dipl-inf)

01/2006-04/2006: Project thesis: Geometrische Modellierung des kardialen Gefäßsystems (Geometric modeling of the cardiac blood vessel system) at the DFKI[2] Kaiserslautern

05/2006-10/2006 Diploma thesis:
Entwicklung einer Ontologie zur Modellierung von Herzvitien (Development of an Ontologie for the modeling of heart diseases) at the DFKI Kaiserslautern

01/2007-12/2007: Industrial experience in a co operational Project between the TU Kaiserslautern and the ProCAEss GmbH Landau

01/2008-04/2011: Member of IRTG 1131

2 German Research Center for Artificial Intelligence