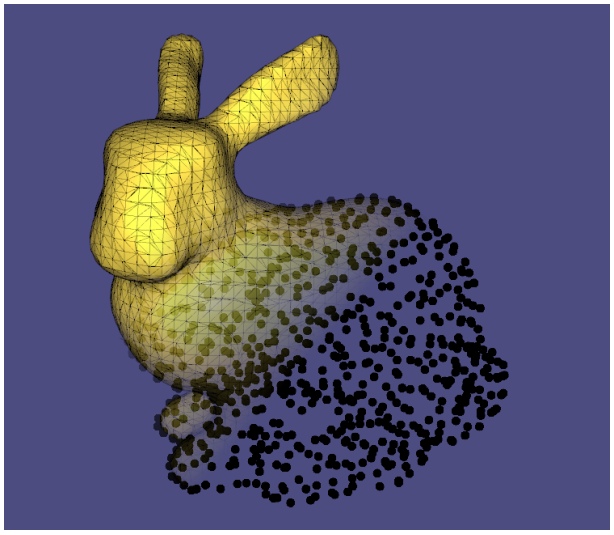
STRUCTURE-AWARE SURFACE RECONSTRUCTION WITH SPARSE MOVING LEAST SQUARES



Alexander Lelidis

Bachelor Thesis
August 2015

Supervisors:
Prof. Dr. Markus Gross (ETH)
Prof. Dr. Marc Alexa (TU)
Dr. Cengiz Öztireli (ETH)

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Technische Universität Berlin

cgl computer graphics laboratory

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Abstract

Surface reconstruction from scanned data is still an important task in computer graphics. It is the only way to get a discrete representation of real world models. In this thesis, we propose a reconstruction framework, which combines successful methods from different fields of computer science and can easily be extended. Our approach is based on the assumption, that most real world surfaces can be characterized by a few features. This fact leads us to compressive sensing and dictionary learning. Our method extends the classical moving least squares technique with a learned dictionary and a sparse solving process. This reformulation gives us in return the fast reconstruction due to the local nature of MLS, but also does not lose the global view of the data thanks to the learned dictionary. Our technique is capable of reconstructing undersampled and noisy models in a state-of-the-art run time. Extensive experiments show our progress and document our way from the one to the three dimensional case. Additionally, we implement a few established reconstruction methods in our framework, which allows us a live comparison of the different results and shows that our method is able to keep up with them.

Summary

This bachelor thesis engages the problem of surface reconstruction from oriented 3D points. We propose a new technique that combines methods from the field of compressive sensing and dictionary learning. The local nature of the moving least squares, enables fast surface reconstruction, but still keeps the global geometry in context with the precomputed dictionary. In this thesis we review the roots of surface reconstruction, the state-of-the-art methods and discuss their advantages and disadvantages. In terms of compressive sensing we discuss the different definitions of sparsity and their mathematical models. We also shortly look into dictionary learning and its usage in surface reconstruction. After a related work review we explain the theoretical idea of our algorithm and take a deeper look into moving least squares, K-SVD and solving a least squares system with a sparse constraint. In our experiments we try to prove that the theory works and create tests for the 1D and 2D case. We analyze the run time and the error of different L_1 norm base solvers. Finally, we summarize the results of our contribution.

Zusammenfassung

Diese Bachelorarbeit beschäftigt sich mit dem Problem der Oberflächenrekonstrution von orientierten 3D Punkten. Es wird eine neue Technik vorgestellt, welche Methoden aus dem Bereich "compressive sensing" (komprimiertes Abtasten) und "dictionary learning" (maschinelles Lernen) kombiniert. Aufgrund der lokalen Definition von "moving least squares" (bewegte kleinste Fehlerquadrate) wird eine schnelle Oberflächenrekonstruktion ermöglicht, wobei der globale Blick auf die Geometrie aufgrund des vorberechneten Wörterbuchs nicht vernachlässigt wird. In dieser Abschlussarbeit werden die Grundlagen der Oberflächenrekonstruktion aufgearbeitet und die Vor- und Nachteile von aktuell angewandten Methoden diskutiert. Im Bereich "compressive sensing" werden verschiedene Definitionen von "sparsity" (Seltenheit) und deren mathematische Modelle betrachtet. Es werden Methoden zum Erlernen eines Wörterbuchs erarbeitet und ihre Anwendung im Feld der Oberflächenrekonstrution diskutiert. Nach der Revision der zusammenhängenden Methoden und Algorithmen, wird das theoretische Fundament der neuen Methode geschaffen. Dabei ist eine detaillierte Auseinandersetzung mit dem moving least squares und dem K-SVD Algorithmus notwendig. Zusätzlich erfolgt eine Betrachtung der Lösung von linearen Gleichungssystemen mit einer Sparsity Einschränkung. In anschließenden Experimenten wird versucht, die Theorie zu überprüfen. Es werden verschiede Tests für zunächst den eindimensionalen und darauffolgend den zweidimensionalen Fall erstellt. Die Laufzeit und die Rekonstrutionsfehler von verschiedenen L_1-norm basierenden Lösern werden analysiert. Abschließend werden die Ergebnisse zusammengefasst.

Contents

List of Figures

List of Tables

1

Introduction

In the last years the amount and quality of 3D scanning devices have increased drastically. The devices are not anymore highly complicated scientific laboratory material, they are now affordable and easy to handle. A big step towards the user was the Kinect sensor, whose RGBD output could easily be used to create 3D point clouds. This example shows, how easy it is to get 3D scanned points. Of course there are many other devices, which all have their field of application, depending on the scale, noise, scanning precision and scanning time. However, nearly all devices are facing the same problems. At first, there is the obvious trade-off between accuracy and storage, which means that a detailed point cloud requires a large amount of points. A good example is the famous scanning project of Michelangelo's David [Lev04], that required 250 GB of storage. Also the trade-off between noise and scanning time is important. Most of the time we are not interested in the raw point set, but in the underlying surface. Since the early 90s many robust surface reconstruction algorithms have been developed, which can handle scanning problems like noise, outliers and holes. It is useful to compute a 2D surface defined by a point could, that could be triangulated and used for rendering or animation. Reconstructing a surface from a raw point set is still an open challenge. The reasons behind this are various. A big problem is the distinguishing between noise and sharp features of the surface. There is no clear mathematical way to separate noise from really sharp features, e.g. spikes or edges. The usual way of handling noise is to average the neighboring points to smooth out the noise, which works really well, if the surface is very smooth. But in case of sharp features, they can easily get blurred out. Current methods rely on manually defining the areas with sharp features [GG07] or using precomputed dictionaries [XZZ+14a]. Since local approximation with moving least squares [Lev03] solves the problem efficiently, but gets easily confused by noise or outliers and loses the global view of the surface. Global methods using radial basis functions (RBF) are more robust, but don't give a general solution. Reconstruction with RBF with a large input point set requires solving a huge linear system. Regardless of the computational cost and run time, the interpolation matrix could become ill conditioned and the solving could be unstable.

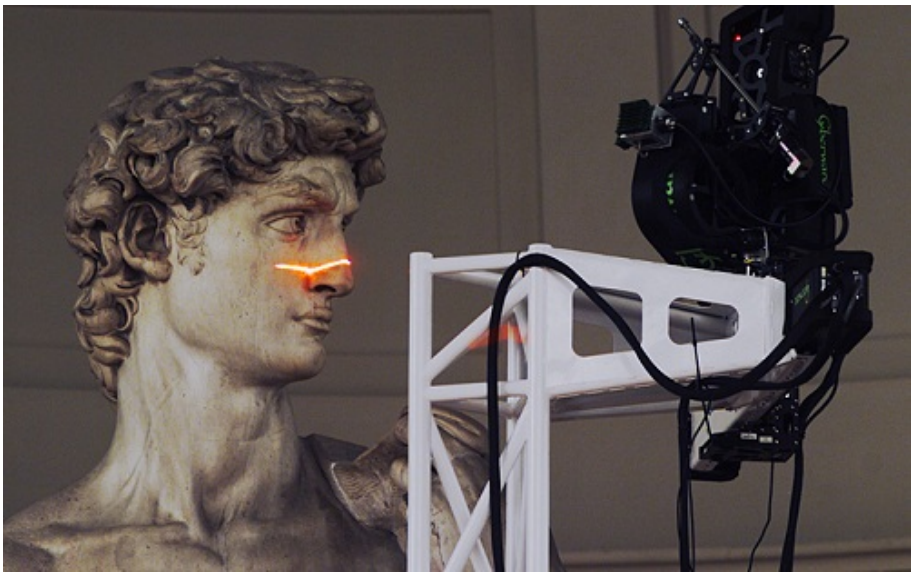


Figure 1.1.: *The laser scanner used to create a digital reconstruction of David.*

This thesis faces the problem of surface reconstruction that takes raw 3D points as an input. The samples could belong to an arbitrary piecewise smooth surface with sharp features like creases or corners. They normally contain scanning artifacts like noise, outliers or holes. Also the sampling of the surfaces may not be uniform. The output of our method is a mesh defined by its vertices and their connectivity. This problem description is very common in the field of surface reconstruction. As mentioned before the coexistence of noise and sharp features causes a very challenging problem. In this thesis, we will combine global and local methods in an efficient manner by using learned local geometry basis and sparse moving least squares fits.

The thesis is organized in the following way. In section 2 we review related work in the field of surface reconstruction, compress sensing and dictionary learning. Afterwards, in Section 3 we present the theoretical background of our algorithm. Section 4 discusses the performed experiments and their results. Finally, the thesis ends with the conclusion section.

2

Related Work

In the following section we give an overview of the relevant work, which we separate in 3 parts. First, we take a look into surface reconstruction and discuss currently used algorithms, in terms of noise, sharp features and irregular sampling. In the second part we discuss sparse signal reconstructions and review different mathematical solution approaches and their benefits. We also give an overview of the surface reconstruction algorithms, which use sparse norm for surface reconstruction. Then, we shortly review methods from the field of dictionary learning and their application in the surface reconstruction.

2.1. Surface reconstruction

In the last 20 years a lot of research has be done in the domain of surface reconstruction from scanned data. A lot of reconstruction algorithms have been proposed. However, nearly all existing methods can be classified into direct and indirect methods, which both have different benefits and downsides. The direct methods on the one side may require denoising, because they directly operate on 3D scanned points, whose outliers can be very confusing. Also the selection of a good subset of the scanned points, that gives a good representation of the surface, is a challenging task. Finally the resulting output needs to be triangulated to fit the given problem description.

On the other side indirect methods require the construction of an implicit function, which zero set defines the surface. If we don't get the surface normals from the scanning device, the algorithm needs to estimate those. Surface normals are very important for rendering algorithms and for computing discrete differential mesh properties. Finally, we also need to run the marching cubes algorithm [LC87] to obtain a triangular discrete mesh.

2.1.1. Direct methods

Direct methods usually interpret a subset of scanned data points as vertices and try to link them directly to a graph, which represents the surface. It is very common to run the algorithm only on a subset of the input points, because otherwise resulting mesh would be very dense and not smooth on continuous surface areas due to noise. A traditional algorithm for this purpose is the Spectral Surface Reconstruction [KSO04], where they perform Delaunay tetrahedralization, which is basically the 3D extension of Delaunay triangulation [CX04]. Afterwards they use a version of spectral graph partitioning to decide whether each tetrahedron is inside or outside of the original surface. Because of the local inside-out decisions, which are based on the global view of the model, this algorithm can handle a several amount of outliers and regions with no samples (holes). Still the spectral algorithm is not infallible, the biggest problem is that it produces unwanted handles and has slow run time.

The more recent scale-space meshing method [DMSL11] proposes first to filter the raw input data based on the intrinsic heat equation, also called mean curvature motion (MCM) and then to interpolate the subset of the filtered points to reconstruct the surface. It is still likely that this method produces reconstruction artifacts like jaggy edges.

Another well established method [DG01] tackles the problem of detecting an undersampled surface region. This scanning problem often appears on the boundary of a surface or a region with high curvature. The method uses this information to detect boundaries and sharp feature of the scanned object and uses them to reconstruct non smooth surfaces. The biggest drawback is that the method makes some assumption about the distribution of the scanned points, which are not necessarily fulfilled for every scanner.

A good overview of methods for surface reconstruction via Delaunay triangulation can be found in [CG06].

In summary, we can say that direct methods are still very sensitive to noise. Additionally, the computation of the Delaunay tetrahedralization or the 3 dimensional Voronoi diagram is highly expensive in terms of memory and CPU time.

2.1.2. Indirect methods

Indirect methods, also called implicit surface reconstruction methods, construct an implicit function, typically a signed distance function (SDF). This function is positive on the outside of the scanned object and negative on the inside. Indirect methods perform isosurfacing of the zero level set of that function to gain the surface of the scanned geometry. We can distinguish the indirect methods in two main categories: local and global methods.

Global methods

Global methods for surface reconstruction consider all points at once and solve only one big dense symmetric linear system. In 2001 J.C. Carr proposed a method [CBC$^+$01] that reconstructs 3D Objects with polyharmonic radial basis functions (RBF). Through a greedy algo-

rithm in the fitting procedure they are able to reduce the number of RBF centers to represent the surface and gain a significant compression and the possibility to process millions of surfaces points. The polyharmonic RBF creates a smooth surface and suits well for reconstructing scale-independent non-uniformly sampled data. Due to its global nature, the method can smoothly fill regions of missing data. The biggest drawback of this method is that the runtime highly correlates with the object complexity. A more complex geometry requires more RBF centers to not blur the fine details, which leads to a bigger matrix and can cause numerical issues at some point.

A different global state-of-the-art method is the Poisson Surface reconstruction [KBH06] that also considers all points at once and is therefore highly resilient to scanner noise. Unlike the radial basis function class, the Poisson reconstruction approach allows locally supported basis functions and therefore the solution is reduced to a well conditioned sparse linear system. To reconstruct the surface, they compute the a 3D indicator function χ, defined as 1 at points inside and 0 at points outside the model. Finally the surface is reconstructed by isosurfacing the function. The algorithm basically relies on the relationship between the normals of the scanned geometry and the gradient of the indicator function. Since the normals define a vector field, which is zero everywhere except on the surface, the problem of finding the indicator function reduces to reserving the gradient operator. That means finding a 3D scalar function whose gradient field approximates the normal vector field best. The resolution of the reconstructed mesh can be intuitively controlled by the depth of the used octree, which defines the neighbourhood of a point and correlates directly with the runtime, the used memory and the size of the resulting mesh.

In conclusion, we can say that global methods are generally very robust to noise, but expensive to compute because of the global dense system.

Local methods

On the other hand local methods use only a subset of the total data set at each step. The main advantage is that the computation is very fast, even for large data sets. 1992 Hoppe proposed [HDD+92] the first milestone in local indirect surface reconstruction. The basic idea was to approximate the manifold by locally estimating the tangent plane and using it to compute the SDF. It was one of the first methods, which does not assume the surface topology or the existence of boundaries in advance. The algorithm fails in surface areas with high curvature, because the tangent plane does not approximate the surface very well. A naive solution to this problem would be to increase the resolution of the grid and reduce the search radius until the surface is similar to a plane again. The clear drawback beside the increasing computational costs is that the algorithm is more influenced by noise and also requires a dense sampling to still be able to compute the plane orientation.

The currently most successful meshless geometry representation, motivated by the fact, that detailed geometry needs a lot of small primitives, which contributed less than a pixel, is the Point Set Surface (PSS) proposed by Alexa in [ABCO+01]. PSS uses the moving least squares (MLS) to approximate a smooth manifold surface from a set of points close to the original surface. The degree of the approximation easily adapts to the noise level of the scanned geometry. With the proposed resampling technique they also tackle problems like noise and redundancy reduction.

For rendering point set surfaces interactively they propose to use their upsampling method, if the point resolution is too small. The point resolution should be adapted with respect to the screen space resolution. However, this method does not discretize the zero set of the implicit function to a mesh and keeps points, as the name suggests, as the surface primitives.

Another very popular method is the Algebraic Point Set Surfaces [GG07], that presents a new Point Set Surface definition based on moving least squares fitting of algebraic spheres. The APSS method is much more robust in terms of low sampling rates or regions with high curvature. In consequence of the sphere fitting procedure, the method returns the mean curvature of the surface for free, which corresponds to the fitted sphere radius. In case of oriented points the algorithm takes the normals into account for the minimization and punishes the gradient to the reconstructed function to match the direction of the normals. As an extension of their method, they propose a way to handle sharp feature by classification of the two sides of an edge performed during the runtime by the user. The APSS algorithm uses only points from one class at a time to reconstruct the corresponding surface, which does not smooth the edges or corners out and is able to reconstruct sharp feature.

Summarizing, we could say that local indirect surface reconstruction methods are much faster than the global ones. They do not suffer from numerical problems, but can easily get confused by missing geometry if the local neighbourhood is too small.

2.2. Sparse signal reconstruction

In the last years compressed sensing (sparse signal reconstruction) has gained a lot of attraction in many fields like applied mathematics, computer science and signal processing. The main goal of compressed sensing is to find a basis or dictionary in which group of signals can be sparsely encoded and reconstructed. We consider a signal as sparse if most of the signal elements are zero. The fraction between the zero and nonzero elements defines the sparsity of a signal. From a general viewpoint the reduction of dimensions leads to efficient modelling techniques and noise reduction. In 2004 the compressed sensing pioneer Emmanuel Candès proofed in [CRT05] that given the knowledge about the sparsity of the signal and the corresponding basis, the data could be reconstructed with less samples than required by the Nyquist-sampling-theorem. That means underdetermined systems with normally infinite solutions can be solved with a constraint minimization (convex optimization) converging to the correct solution. Generally speaking, compressed sensing takes advantage of the fact that the world is not totally random. As a matter of fact many interesting signals are remarkably sparse if they are analyzed in the right domain. Finding the right domain (basis) for a given class of signals is one of the major research topics in compressed sensing. In the next section we are going to discuss the related work and algorithms for using a learned dictionary as a basis. However, assuming that we have the domain in which the observed signals are sparse, we still need to find a solution for the underdetermined system. The least-squares approach is to minimize the L_2 norm of the energy of the system

$$\min_x ||Ax - y||_2^2 \tag{2.1}$$

where A stands for the basis of the sparse domain and y is the measured signal. This approach usually leads to poor reconstruction results, because the zero entries of the signal will get a nonzero value which leads to the loss of sparsity. Also, in presence of noise the least squares minimization leads to overfitting. To achieve better results and enforce sparsity, one could minimize 2.1 with the L_0 norm instead of the L_2. The L_0 norm counts the number of non zero entries in a vector and is not a proper F-norm, because of its discontinuous definition. Solving this problem would return the optimal solution in terms of sparsity and reconstruction error. But as proved by Dongdong et al. in [GJY10] finding the global minimum of the $L_p(0 \leq p < 1)$ minimization is a strongly NP-hard problem. But again Candès and Romberg proved in [CR05a] that many L_0 minimization problems can be solved by replacing the L_0 norm with the L_1 norm also known as Taxicab norm or Manhattan norm. The L_1 norm is defined as the sum of the absolute values of the entries of a vector

$$||x||_1 = \sum_{i=0}^{n} |x_i| \tag{2.2}$$

Strictly speaking, it is equivalent, if the coefficient vector x is sparse enough, to solve the much easier L_1 minimization instead of the NP-hard L_0. In [SMW+07] Sharon et al. proposes an algorithm for determining the L_1 - L_0 equivalence for error correction and sparse representation.

Solving the a linear system with L_1 norm can be expressed as a linear program or a basis pursuit denoising.

However, compressed sensing has also found its way to the field of surface reconstruction. Similar to us, in 2010 Avron et al. assumed in [ASGCO10] that common objects, even geometrically complex ones, can typically be characterized by a rather small number of features. This realization led them to the field of sparse signal reconstruction. They propose a global sparse method which uses the L_1 norm and solves the problem in a two-step fashion. First they solve the orientation under the assumption that a smooth surface has smooth varying normals. The computed orientations are used to compute the positions. To achieve a lower sparsity than L_1 sparsity they use the iterative reweighted version [CWB08].

$$\min_x ||Wx||_1 \; s.t. \; ||Ax - y||_2^2 \tag{2.3}$$

where W is a diagonal weighting matrix. The basic idea is to solve the equation (2.3) iteratively. In the first iteration W can be represented as the identity matrix and in next iteration the weights w_i are defined inversely proportional to the true signal magnitude. However, their global sparse method achieves reasonable reconstruction, with concentrated error at the corners or edges, because points lying directly on these spots have no clearly defined orientation. This problem leads to spikes sticking out of the model.

Recently, another two-step method which uses the reweighed L_1 was published in 2014 by R. Wang in [WYL+14]. Their main focus lies in on decoupling noise and features of a given geometry. In the first phase they approximate a base mesh with a global Laplacian denoising scheme. They prove that if the number of samples tends to infinity, the base mesh converges to the underlying surface with probability 1. In the second phase they rely on the discovery that sharp features can be sparsely represented by a dictionary constructed by the pseudo inverse

of the Laplacian matrix. Recovering of those features is done by a progressive reweighted L_1 minimization. The results of the method are quite good, if it is applied to meshes, because meshes in contrast to point clouds contain the basic information of the surface topology.

2.3. Dictionary learning

Dictionary learning is one of the most studied fields in machine learning. Sparse dictionary learning uses the coefficients form some signals in a given basis to compute a new dictionary D in which the signals can be represented sparsely. Learning a dictionary is an NP-Hard problem [Til14], which is even very hard to approximate. The most popular approximation algorithm is the K-SVD algorithm, which we are also going to use.

Dictionary learning has been used in several fields of science. In computer graphics it is mostly used in image compression or inpainting. Recently, dictionary learning has also found its way to 3D geometry processing. In 2014 J. Digne proposed in [DCV14] method, which exploits the self-similarity of underlying shapes. The algorithm locally resamples the point cloud and uses the new samples, which they call centers, to compute a dictionary of the local height field discrete cosine transform (DCT) coefficients. Finally, they use the dictionary and the centers to compress the point cloud without losing important information.

Another method for robust surface reconstruction using a dictionary was proposed in 2014 in [XZZ+14b]. They use a dictionary consisting out of the vertices of the reconstructed triangular mesh and a sparse coding matrix, which defines the connectivity of the vertices. They minimize for the optimal triangulation of the input data, while taking many factors into account. As a cost function, they define the distance between the reconstructed mesh and the input point set. The results are quite good and outperform a few state-of-the-art methods, but their optimization model is nonconvex, which means they can not guarantee convergence against a global minimum. Also, in case of missing data the algorithm has problems to fill those, because of the missing samples in that region.

3

Reconstruction model

In the following section we are going to describe our approach more in detail and explain the used mathematical models and their properties. We start with a basic surface reconstruction algorithm and along the section extend it. We begin by clearly formulating the discussed problem.

3.1. Problem definition

Our problem can be characterized as follows: Given a point set containing n oriented 3D points $\mathbb{P} = \{p_0, p_1, \cdots, p_n\}$, where $p_i \in \mathbb{R}^3$ with corresponding normals $\mathbb{N} = \{n_0, n_1, \cdots, n_n\}$, $n_i \in \mathbb{R}^3$ and $||n_i||_2 = 1$ sampled from a closed manifold surface S. We try to find a discrete mesh $\mathbb{M}$ that is defined as a set of vertices and faces $\mathbf{M} = \{\mathbb{V}, \mathbb{F}\}$. The vertex set $\mathbb{V} = \{v_0, v_1, \cdots, v_n\}$ defines the position of the geometry in space. The face set for triangular meshes is defined as $\mathbb{F} = \{f \mid (v_i, v_j, v_k) = f \wedge v_i \neq v_j \wedge v_j \neq v_k \wedge v_i \neq v_k, v_i, v_j, v_k \in \mathbb{V}\}$. The reconstructed mesh should approximate the surface S so that the numerical and visual error is as small as possible.

3.2. Moving least squares

Since the whole algorithm will build on moving least squares, we will start with the basic definition proposed in 1981 [LS81] for smoothing and interpolating data. At first a brief explanation of the least squares problem is given.

3.2.1. Least squares

The least squares method is a very common method to approximate the solution of overdetermined systems. The standard example is fitting a line in a point cloud. In a more abstract way that means that we want to compute a global function $f(x)$ from N points p_i and their function values y_i that the least squares error between $f(p_i)$ and y_i is as small as possible. Consequently we need to minimize the following statement

$$\min_f \sum_{i=0}^{N} ||f(p_i) - y_i||_2^2 \tag{3.1}$$

where f is the reconstructed function and d is the dimension of the domain, in which the function is defined in. Since the function f consists not of arbitrary elements, but is defined with respect to some basis we can write (3.1) as

$$\min_{c \in \mathbb{R}^d} \sum_{i=0}^{N} ||b(p_i) \cdot c - y_i||_2^2 \tag{3.2}$$

where $b(p_i)$ is the basis for the point p_i and c is the unknown coefficient vector, we are trying to compute.

Equation (3.2) can be solved analytically by computing the partial derivatives and setting them to zero. If we write (3.2) in matrix form we get

$$LS_{error} = \min_{c \in \mathbb{R}^d} ||B(p)c - y||_2^2 \tag{3.3}$$

where B is the basis matrix. Now we need to compute the gradient ∇LS_{error} and set it to zero to find a minimum.

$$\nabla LS_{error} = c^T B^T Bc - 2y^T + Bc + y^T y \stackrel{!}{=} 0 \tag{3.4}$$

Since the last term of the equation does not depend on c, we remove it. Finally we need to solve

$$c = (B^T B)^{-1} - B^T y \tag{3.5}$$

This operation could be problematic, because the matrix $B^T B$ is not necessarily easy to invert. However, for a more detailed derivation, we refer to [Fed15].

3.2.2. Weighted Least squares

The problem of using a least squares fit is that all points p_i get treated equally, which is not what is preferred, because points which are far away from the evaluation point x get the same weight as points really close to it. To address the issue, we introduce a weighting function $\theta(d)$, depending on the Euclidean distance between x and p_i.

$$\min_{f_x \in \mathbb{R}^d} \sum_{i=0}^{N} \theta(||x - p_i||_2)||f_x(p_i) - y_i||_2^2 \tag{3.6}$$

In matrix form we get

$$WLS_{error} = \min_{c \in \mathbb{R}^d} W(x,p)||B(p)c - y||_2^2 \tag{3.7}$$

where $W(x,p)$ is a diagonal matrix containing the weights on the diagonal. Again we can solve the function analytically.

$$c = (B^T W B)^{-1} - B^T W y \tag{3.8}$$

Weighting function

The choice of the weighting function depends on the application. A standard weighting function to gain smooth varying weights with local support is a Gaussian function.

$$\theta(d) = e^{-\frac{d^2}{\sigma^2}}$$

where the additional parameter σ controls the width of the weighting function.

Another weighting function is the tricube weight function defined as

$$\theta(d) = \begin{cases} (1 - |d|^3)^3 & \text{if } |d| < 1 \\ 0 & \text{else} \end{cases}$$

that has a very compact support and is used in LOESS [Cle79].

3.2.3. Moving weighted least squares

The moving least squares (MLS) is basically a weighted least squares which gets moved over the domain $\mathbb{R}^d$. The MLS function only takes the points inside a given radius r into account, which is the reason for the local support and the efficient evaluation. This problem reduction is possible, because the weighting function gives points further way than the radius a weight close to zero and they become neglectable for the local result.

$$\min_{f_x \in \mathbb{R}^d} \sum_{i=0}^{k} \theta(||x - p_i||_2)||f_x(p_i) - y_i||_2^2 \tag{3.9}$$

where k is the number of points inside the radius r. The reconstructed function is continuously differentiable if and only if the weighting function θ is continuously differentiable [Lev98].

3.2.4. MLS for surface reconstruction

All the explained methods are widely used in computer science. To get back to surface reconstruction we are going to describe the work flow for reconstructing a surface using MLS. Since we work with computers we need to discretize the 3d domain. Usually the best way is to create a grid inside the bounding box defined by the sampled points. An advantage of this method is that computed results could be directly used as an input for the marching cubes algorithm

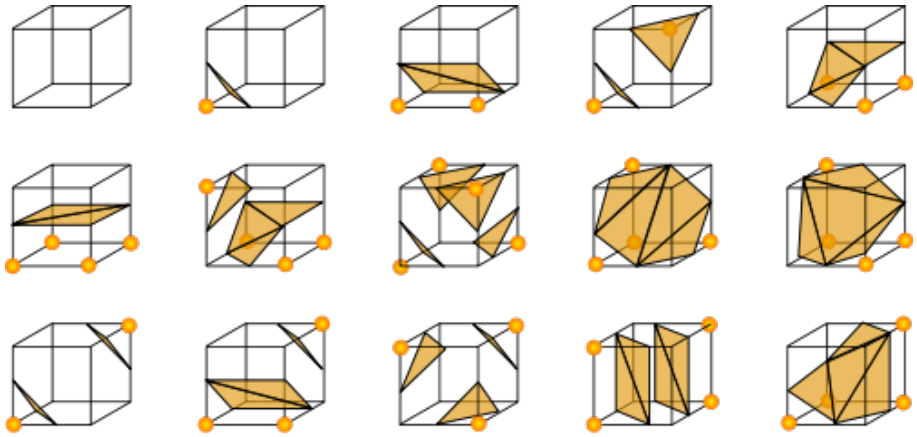


Figure 3.1.: *The originally published 15 cube configurations*

[LC87]. Another useful feature is the intuitive correlation between surface resolution and grid resolution. However, using a grid to approximate the continuous space has the downside that a lot of computation, like neighborhood queries, is wasted because some grid points can be far away from the surface and therefore do not contribute to the surface reconstruction. The signed distance value for every grid point is computed, using MLS with some suitable basis. The resulting output is directly used to extract the zero set of the implicit function, which defines the surface. The isosurfacing is done by the marching cubes algorithm. It basically takes 8 neighbours at a time, which form an imaginary cube (3.1), and returns the polygons that are necessary to represent the surface passing trough the cube. The result of the marching cubes algorithm is the result of the whole surface reconstruction procedure. A briefer description can be found in Algorithm 1.

A good overview of the least squares problems can be found in [Nea04].

3.3. Compressed sensing and surface reconstruction

We now have a basic model that we can extend. As previously mentioned we want to combine the classical MLS definition with a sparse signal reconstruction. Since real world surfaces are basically characterized by a few features, we are expecting to reduce noise by a sparse reconstruction. However, there are multiple ways to define a minimization problem that has a sparse solution. As mentioned above, we are going to use the L_1 instead of the L_0 norm to reduce the complexity of the problem.

3.3.1. Sparse minimization with equality constraints

Given noise free data the optimal way to gain a sparse solution is to solve

$$\min_{x \in \mathbb{R}^d} ||x||_1 \; s.t. \; ||Bx - y||_2^2 = 0 \tag{3.10}$$

Algorithm 1 Algorithm for extracting the surface from a point cloud

```
                                        ▷ % Compute the bounding box from the given points%
boundingBox = computeBoundingBox(inputPoints)
                                        ▷ % Compute grid inside the bounding box with a given resolution%
grid = createGrid(boundingBox.min, boundingBox.max, resolution)
for i = 0 to grid.length do
                                        ▷ % get the point inside a given radius%
    neighbors = getNeightborsInRadius(inputPoints, grid[i].position, radius)
    if (neighbors.length == 0) then
                                        ▷ % If there are no neighbors the grid does not contribute%
        grid[i].value = NULL
        continue
    end if
                                        ▷ % compute the implicit function value at the current grid point%
    grid[i].value = computeImplicitFkt(grid[i].position, inputPoints)
end for
                                        ▷ % run the marching cubes algorithm to extract the iso surface%
[vertices, faces] = computeMarchingCubes(grid)
return [vertices, faces]
```

where B is a suitable basis matrix in which the signal y can be sparsely encoded. This definition is not very suitable for us, because we work with a floating point representation and possibly noisy data. We can modify (3.10) to not solve $Bx = y$ directly but introduce only a small error ϵ.

$$\min_{x \in \mathbb{R}^d} ||x||_1 \; s.t. \; ||Bx - y||_2^2 < \epsilon \tag{3.11}$$

This definition can handle noise better than the previous definition and can be controlled by the parameter ϵ. However, the sparsity is not a hard constraint. That means we want a sparse solution for the linear system, but it is more important that the solution accurately solves the system $Bx = y$. This convex problem is typically solved by linear programming.

The smoothed L0 algorithm proposed in 2009 [MBZJ09] tires to minimize the L_0 norm directly. Therefore, it approximates the L_0 norm by a smooth function f_σ depending on σ.

$$SL0(X) = \sum_{x_i \in X} f_\sigma(x_i) \text{ where } f_\sigma(x) = \begin{cases} 1 & \text{if } x > \sigma \\ 0 & \text{else} \end{cases} \tag{3.12}$$

The variable sigma determines the smoothness of the function. The smoother the function is, the worse is the approximation for the L_0 norm. Finally, the goal is to minimize f_σ with a small sigma. To not get stuck in local minima, the algorithm uses a graduated Non-convexity procedure and starts with a large σ and decreases it in every iteration, to find the global minimum.

3.3.2. Least squares minimization with sparsity constraints

Another slightly different formulation is

$$\min_{x \in \mathbb{R}^d} ||Bx - f||_2^2 \; s.t. \; ||x||_1 < t \tag{3.13}$$

where we minimize the least squares system with a hard sparsity constraint. This method is preferred if the sparsity of the signal x is known. With the parameter t the sparsity of the signal is directly controlled.

This problems are typical solved by greedy algorithms like matching pursuit (MP) [MZ93]. MP computes the best nonlinear approximation with respect to some basis or dictionary. This approximation is stored in a sequence and repeated iteratively. A light extension is the orthogonal matching pursuit [TG07] which requires that after every step the extracted coefficients are projected onto the selected basis. This procedure can create better results than the standard MP formulation, but the computation is also more expensive.

3.3.3. Sparse regularized least squares

Another possible way to formulate the problem as a L_1 regularized least squares program:

$$\min_{x \in \mathbb{R}^d} ||Bx - f||_2^2 + \lambda ||x||_1 \tag{3.14}$$

where λ is a weight to decide the importance of the sparsity of the solution. It is inverse correlated to the parameter t mentioned in formulation before. The class of the techniques is called LASSO (Least Absolute Shrinkage and Selection Operator) and tries to combines L_2 and L_1 minimization, with the aim to get the advantages of both: A sparse solution with a small mean squared error. The problem could be remodeled as a convex quadratic problem and solve by an interior point method.

A specialized interior point algorithm for solving large scale L_1 regularized systems was proposed in [KKB07]. They use the preconditioned conjugate gradients method to compute the search direction for the next iteration step.

3.3.4. Sparse moving least squares

We remodel our MLS minimization to be sparse:

$$\min_{c \in \mathbb{R}^d} \sum_{i=0}^{k} \theta(||x - p_i||_2) ||b(p_i) \cdot c - y_i||_2^2 + \lambda ||c||_1 \tag{3.15}$$

And in matrix notation

$$\min_{c \in \mathbb{R}^d} W(x,p) ||B(p)c - y||_2^2 + \lambda ||c||_1 \tag{3.16}$$

where we can regulate the sparsity of the solution with λ. To gain a sparse solution with a small mean squared error the signal needs to be sparse in the the domain of the basis B. Assuming so, the parameter λ can be used to smooth out the surface and reduce noise. We propose to choose the parameter λ accordingly to the expected noise level.

3.4. Designing the basis

Finding the right basis or dictionary for a given class of signals is an open research problem in compressive sensing. In signal processing, wavelets often have been used as sparse basis for oscillating signals or image compression. The basic idea is to express the data in a domain, where most of it can be represented as a sparse combination of the basis or atoms. If the number of used basis tends to infinity the expressed signal will perfectly match the original one.

On the other hand, a learned dictionary also serves well as a domain, where the representations would be sparse. A so called data-depended dictionary needs to be trained on data sets from the same cluster, to compute the atoms of the dictionary. Finally, the signal from the same cluster can be represented as a linear combination of the atoms. With an increasing number of training data and dictionary atoms the reconstruction error should converge to zero. If the dictionary was trained on data with fine details like high frequencies, it is possible to reconstruct those also in case of under sampling. Furthermore, if the training signals where basically noise free, it is also possible to remove noise during the reconstruction. The dictionary training is an offline process, which is very expensive in terms of memory usage and CPU time, but only needs to be done once.

3.4.1. Gaussian height field

In our case we want to use a Gaussian height field (GHF) as basis. The GHF is defined by a number of Gaussian centers and a corresponding σ value. But before we can define the GHF we need to find a good reference coordinate system, which fits the local points well.

Local coordinates system

Since we iterate over the grid, representing the discrete approximation of the continuous 3D space, we only use the k points inside a sphere around the grid point x with radius r to estimate the local frame. If the radius is small enough the given points form a proper height field. To extract the height field we need to compute the principal directions of the points P inside the sphere. To extract those we first need to center the points by subtracting the mean point $\bar{p}$ from every point p_i:

$$\tilde{P} = [p_0 - \bar{p}, p_1 - \bar{p}, \cdots, p_k - \bar{p}] \tag{3.17}$$

The next step is to compute the singular value decomposition of $\tilde{P} \in \mathbb{R}^{3\times k}$.

$$U\Sigma V^T = SVD(\tilde{P}) \tag{3.18}$$

where the matrix $U \in \mathbb{R}^{3\times 3}$ contains the principal directions of P. To gain a height field we need to express the points P in U, which is done by computing $P_{local} = U^T\tilde{P}$. Since we want to center our coordinate system at our evaluated grid point x, we need to transform x to the local coordinate system $x_{local} = U^T(x - \bar{p})$ and subtracted the resulting value from every point:

$$P_{localatx} = [p_{local_0} - x_{local}, p_{local_1} - x_{local}, \cdots, p_{local_k} - x_{local}]^T \tag{3.19}$$

The points in the matrix $P_{localatx} \in \mathbb{R}^{k\times 3}$ define now the height field.

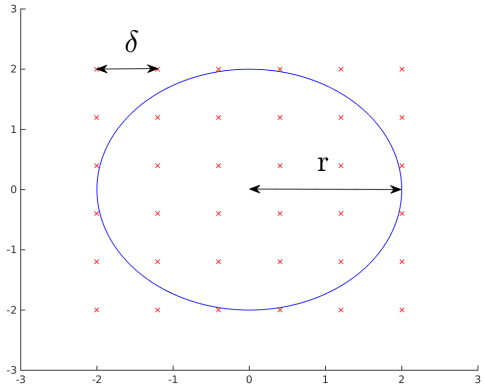


Figure 3.2.: *Distribution of the Gaussian centers.*

Flipping problem

The principal components from the SVD are defined up to the sign. For the first two principal components, this is not a problem, because they are interpreted as the x and y directions and the Gaussian centers are symmetrically distributed around the origin. But for the third principal component, representing the z axis and the orientation of the local plane, the sign matters. We need the z axis to point in the same direction as the point normals to have a consistent inside outside definition. Strictly speaking, we need to check if the plane normal n_p is pointing in the same direction as the weight average normal of the points inside the neighbourhood.

$$n_{avg} = \sum_{i=0}^{k} \frac{\omega_i n_i}{k} \tag{3.20}$$

where ω_i is the weight of the normal n_i and should depend on the distance between p_i and x. We propose to use a simple Gaussian weighting function. After the average normal is compute, we check with the dot product with the plane normal is pointing in the same direction.

$$n_{plane} = \begin{cases} -n_p & \text{if } n_p \cdot n_{avg} < 0 \\ n_p & \text{else} \end{cases} \tag{3.21}$$

Defining the Gaussian matrix

Since we have the points transformed to their local coordinate system with a consistent z axis (plane normal) orientation, we can create the Gaussian basis matrix K. Therefore, we need to create a uniform grid $\Delta \in \mathbb{R}^{m \times 2}$ of m Gaussian center positions with the size $2r \times 2r$ centered around the origin (see Figure 3.2).

The Gaussian matrix $K \in \mathbb{R}^{k \times m}$ is defined in the following way.

$$K = \begin{bmatrix} e^{-\frac{||\Phi_0 - \Delta_0||_2^2}{2\sigma^2}} & e^{-\frac{||\Phi_0 - \Delta_1||_2^2}{2\sigma^2}} & \dots & e^{-\frac{||\Phi_0 - \Delta_m||_2^2}{2\sigma^2}} \\ e^{-\frac{||\Phi_1 - \Delta_0||_2^2}{2\sigma^2}} & e^{-\frac{||\Phi_1 - \Delta_1||_2^2}{2\sigma^2}} & \dots & e^{-\frac{||\Phi_1 - \Delta_m||_2^2}{2\sigma^2}} \\ \vdots & \vdots & \ddots & \vdots \\ e^{-\frac{||\Phi_k - \Delta_0||_2^2}{2\sigma^2}} & e^{-\frac{||\Phi_k - \Delta_1||_2^2}{2\sigma^2}} & \dots & e^{-\frac{||\Phi_k - \Delta_m||_2^2}{2\sigma^2}} \end{bmatrix} \tag{3.22}$$

where $\Phi \in \mathbb{R}^{k \times 2}$ contains the xy coordinates of the transformed points $P_{localatx}$. The parameter σ is defined depending on the radius r and the number of Gaussian centers along the first dimension.

The final minimization problem we need to solve is:

$$\min_{c \in \mathbb{R}^m} W(x,p)||K_{m,r}(p)c - f||_2^2 + \lambda||c||_1 \tag{3.23}$$

where f are the height values, which is equal to the third column of the transformed points $P_{localatx}$,

Signed distance value

After solving the equation (3.23) we need to compute the implicit function value for the grid point x:

$$SDF(x) = \begin{bmatrix} e^{-\frac{||\vec{0} - \Delta_0||_2^2}{2\sigma^2}} & e^{-\frac{||\vec{0} - \Delta_1||_2^2}{2\sigma^2}} & \dots & e^{-\frac{||\vec{0} - \Delta_m||_2^2}{2\sigma^2}} \end{bmatrix} c \tag{3.24}$$

We can use $\vec{0}$ as position for x because we transform the local coordinate system to the grid point x.

3.4.2. K-SVD

Definition

For learning the dictionary we are going to use the K-SVD algorithm proposed in [AEB06]. The algorithm creates a dictionary for discrete signal representation via sparse combinations of the dictionary atoms. The basic algorithm works in the following way.

Given a training matrix $T \in \mathbb{R}^{d \times n}$ of signals, where d is the signal dimension and n is the number of samples. The goal of the algorithm is to compute an overcomplete dictionary out of the a atoms, where every signal can be represented as a sparse linear combination of the atoms

a_i. That means every signal t_i (column in the training matrix T) can be reconstructed by solving sparse minimization systems.

$$\min_{x_i \in \mathbb{R}^d} ||Dx_i - t_i||_2^2 \; s.t. \; ||x_i||_0 < T_0 \tag{3.25}$$

where T_0 defines the maximum number of non zero elements and thus determining the sparsity of the signal. We can write the previous equation in matrix form to solve the minimization problem for all signals simultaneously.

$$\min_{X \in \mathbb{R}^{d \times n}} ||DX - T||_F^2 \quad \forall i \quad ||x_i||_0 < T_0 \tag{3.26}$$

The K-SVD method solves the problem iteratively, by alternating between sparse coding of the training signals based on the current dictionary and a process of updating the dictionary atoms to better fit the data. In the sparse coding stage the algorithm uses a pursuit algorithm like Basis pursuit or Orthogonal matching pursuit to compute the sparse coding. At the first iteration the initial dictionary is created out of randomly selected training signals. In the updating state the method updates every atom a_i from the current dictionary. K-SVD determinates all training signals which use a_i in their reconstruction and compute the reconstruction error E_i on the dictionary D without a_i. Afterwards E_i is decomposed in $E_i = U\Sigma V^T$. The first column in the U matrix defines the new atom a_i. The first column of the V matrix multiplied by the first singular value $\Sigma_{0,0}$ defines the updated coefficients. The K-SVD pseudocode can be found in algorithm 2.

Learning the dictionary

To learn the dictionary we iterate over the domain and use the coefficients from the reconstruction of the signal as test data. This signal are the columns of the training matrix T. The size of T depends on the radius and the number of grid points. Finally we need to choose a compression rate and a threshold for the sparse representation with the atoms. Usually the dictionary size is disproportional to the threshold. That means if the dictionary size increases, we need less atoms to represent a signal precisely.

Usage in surface reconstruction

The final minimization system we work with is:

$$\min_{c \in \mathbb{R}^d} W(x,p)||K(p)Dc - y||_2^2 + \lambda||c||_1 \tag{3.27}$$

where D defines the learned dictionary. We solve this problem with an alternating direction method of multipliers (ADMM), which is designed to solve convex optimization problems by breaking them into smaller pieces. The algorithm was proposed by N. Parikh and S. Boyd in [BPC+11] and first defines a canonical problem form called graph form of the optimization. In the next step the method performs a graph projection splitting, a form of Douglas-Rachford splitting or the alternating direction method of multipliers, to solve graph form problems serially. This algorithm is implemented in the POGS framework [Fou14] by Chris Fougner, which we are going to use to solve 3.27.

4

Experiments

In the following section we are going to describe the experiments we carried out. At first we are only doing one dimensional tests, to get an overview of the behaviour of the solvers and the reconstruction. After getting successful results we are moving on to the two dimensional case and trying to create the base for the three dimensional case. Finally, we use the gained knowledge to implement a system for 3D reconstruction. In all dimensions we analyse the run time and the resulting reconstruction error of different methods. For our experiments we used the numerical computing environment Matlab [MAT12] and the object oriented programming language C++.

4.1. 1D case

4.1.1. Compress sensing experiments

Because our whole algorithm builds up on sparse solvers, we begin by testing different solvers. For that purpose we create a random Gaussian orthonormalized matrix $A \in \mathbb{R}^{128\times256}$ with mean of 0 and standard deviation of 1, which we are going to use to create a measurement vector. For the measurement vector we create a 256 parameter long signal x_0, which contains 25 non zero elements see Figure 4.1. Together with matrix A we create a measurement vector y by multiplying A and x_0

$$y = Ax_0 \tag{4.1}$$

The measurement vector y is used to reconstruct x_0 with different solvers. After the reconstruc-

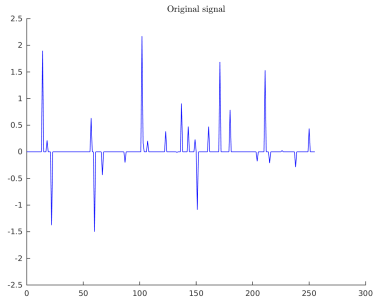


Figure 4.1.: *256 coefficients of the original signal.*

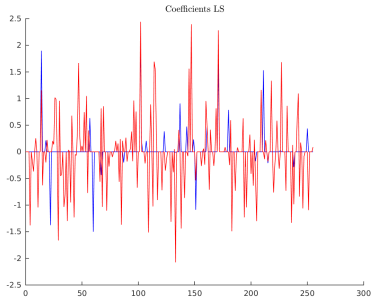


Figure 4.2.: *Least squares reconstruction (red).*

tion we measure the reconstruction error between x_{rec} and x_0. However, to show the importance of a sparse solver, we compute the coefficients with a standard least squares solver and compare them. As we can see in Figure 4.2 the resulting coefficients are not even close to the original ones.

Least squares with sparsity constraint in C++

To minimize the following energy we use the portable C++ compressed sensing framework Kl1p [Geb12].

$$min\ ||Ax - y||_2^2\ s.t.\ ||x||_1 < \epsilon \tag{4.2}$$

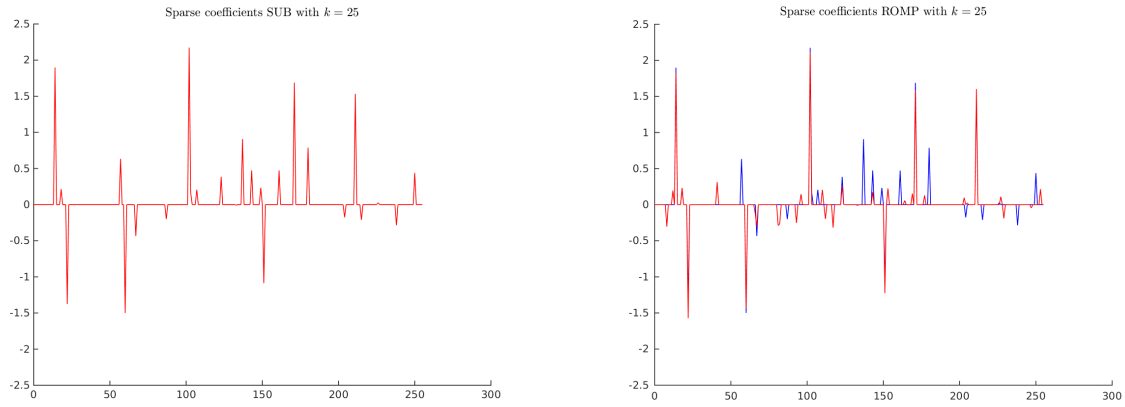
KL1p implements a lot of pursuit algorithms for solving equation (4.2). The subspace pursuit algorithm has the best reconstruction quality, with a mean squared error of 1.1719e-14 and a run time of 17 milliseconds. While the Regularized Orthogonal Matching Pursuit algorithm has the fastest run time of 4 milliseconds, but also a big reconstruction error of 0.0143. The reconstructed signal can be seen in Figure 4.3. For detailed mean squared error and run time information see Detailed run time information of different sparse solvers in the appendix.

L_1 minimization with quadratic constraints in Matlab

For minimizing equation 3.10 we use function l1qc from the Matlab toolbox l1-Magic [CR05b]. The resulting mean squared error is 2.2507e-08 with a run time of 66 milliseconds. Worth mentioning is that the run time of the code is measure in Matlab, which is usually slower than the compiled C++ code. For the l1qc function we used $\epsilon = 0.001$. Figure A.2 (a) shows the reconstructed signal.

L_1 regularized least squares problems in Matlab

For minimizing equation 3.14 we use function l1ls from the Matlab toolbox [KKB08] by Stephen Boyd. The resulting mean squared error is 1.4291e-07 with a run time of approximately 358 milliseconds. Also, this solver is implemented in Matlab and therefore slower than the



(a) (b)

Figure 4.3.: *1D reconstruction.* (a) *Subspace pursuit reconstruction.* (b) *Regularized Orthogonal Matching Pursuit reconstruction.*

C++ solvers, but it is also much slower than the l1qc method. For the l1ls function we used $\lambda = 0.0001$. Figure A.2 (b) shows the reconstructed signal.

Results

Overall the solvers implemented in C++ outperformed the Matlab implementations in terms of run time and mean squared error. However, we still going to use both Matlab solvers due to simplicity and its sufficient performance for the one and two dimensional reconstruction tests.

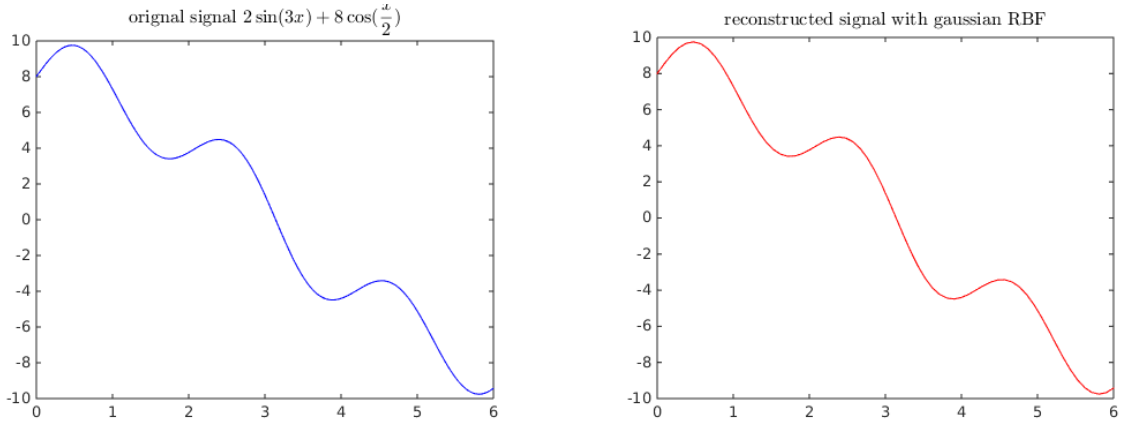
4.1.2. Sparse function reconstruction with Gaussian RBF

L_1 minimization with quadratic constraints

To get a better feeling for the L_1 solving algorithms in terms of reconstruction and their parameters, we will start with the 1D reconstruction and a trigonometric function serving as a signal. At first we will solve the following L_1 minimization with quadratic (L_2 norm) constraints.

$$min\ ||c||_1\ s.t.\ ||Kc - y||_2^2 < \epsilon \tag{4.3}$$

where K is the Gaussian matrix as defined in (3.22) and y are the function values of the trigonometric function $f(x) = 2\sin(3x) + 8\cos(\frac{x}{2})$. The original signal was sampled uniformly with 64 samples in the interval from 0 to 6 see Figure 4.4 (a). For the Gaussian matrix we use $\sigma = 0.187$ and a tolerance of $\epsilon = 1e - 3$. As a sparse solver we used the l1qc algorithm from the l1-Magic [CR05b], which uses the log barrier algorithm for the solving. Since the reconstruction is on really small scale, we use all points at once and use the x values from the sampled points as centers for the Gaussians.



(a) (b)

Figure 4.4.: *1D reconstruction.* (a) *The original signal.*
(b) *The reconstruction using a Gaussian height field.*

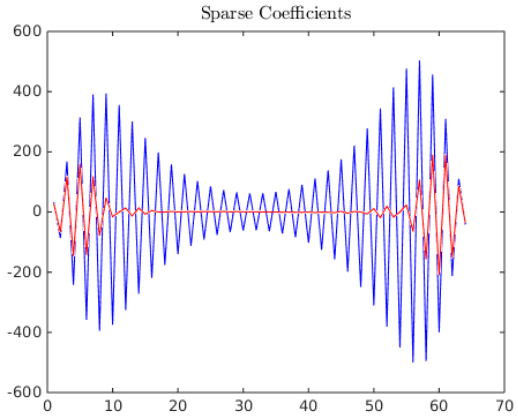


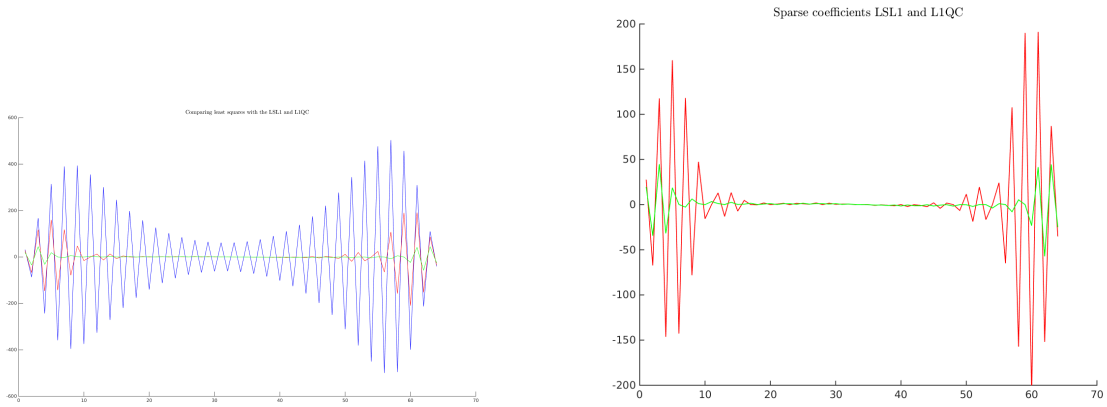
Figure 4.5.: *Least squares compared to sparse coefficients for the reconstruction.*

We can clearly see in Figure 4.4 (b) the reconstructed signal matches the original quite well. The introduced error by the reconstruction is $1.0e-06$ and the reconstruction took 0.027668 seconds.

Figure 4.5 shows the values of 64 coefficients for the RBF. The blue line shows the coefficients, if we solve the problem with a simple least squares solver $min||Kc - y||_2^2$ and enforce no sparsity. The red line shows the coefficients, if we use the sparse solver. It is visible that the coefficients with a small magnitude, in the middle of the signal, close to 0.

Results

The reconstruction with the l1qc was quite fast, nearly 50 percent faster than the Matlab least squares solvers, but also very sensitive to parameters. For the wrong σ value the sparse coefficients for the reconstruction are equal to the least squares solution and return no sparsity. This unstable behaviour is not intuitive for the user and therefore it is not very convenient.



(a) (b)

Figure 4.6.: *Comparing the* (a) *LS (blue), LSQC (red) and the L1LS (green) coefficients and the* (b) *LSQC (red) and the L1LS (green) coefficients from the reconstruction.*

L_1 regularized least squares problems in Matlab

Our next step is to test the L1LS solver on the reconstruction problem. We reformulate the problem description (see 3.14) to match the required input for the solver. All previous parameters are the same as in the section above and the value for the new parameter λ is $1e-6$. The reconstructed signal (see Figure A.3) match visible the original signal quite well. The introduced error is 0.0109 and the reconstruction took 2.237272 seconds, which is a little bit slower as the L1QC method, but the method is much more stable and finds a solution most of the time. Also when we take a look at the resulting coefficients see Figure (4.6), we can observe that they are much sparser, even for such a small λ, than the coefficients from the L1QC method.

To get a better feeling for the λ parameter, which directly corresponds to the sparsity of the reconstruction, we computed the same reconstruction with increasing values from 0.0001 to 0.01 for λ. For every result we compute the resulting reconstruction error see Figure (4.7). We notice that with increase of λ the sparsity increases, but also the reconstruction error. This comes from the fact, that the weight on least squares part in the minimization is reduced.

Adding noise

As we could see, the reconstruction worked well. Now we would like to test the behaviour of the algorithm with added Gaussian white noise. Therefore we created a noise vector v with a signal to noise ratio $SNR = 20$ and added it to the signal $f_{Noisy} = f + v$. We then tried to reconstruct the original signal with Least squares and with LSL1 (Figure 4.8 (a)).

We can observe the expected behaviour in Figure 4.8 (b). The least squares algorithm tries exactly to match the data, but in our case (noisy data) we don't want to fit the data exactly. The LSL1 algorithm visibly performs better (Figure 4.8 (c)). Instead of fitting the data exactly the algorithm is forced by the parameter $\lambda = 0.8$ to compute a sparse solution, which reduces the noise level. The error of the reconstruction is 2.3469, which is nearly the half of the error of the

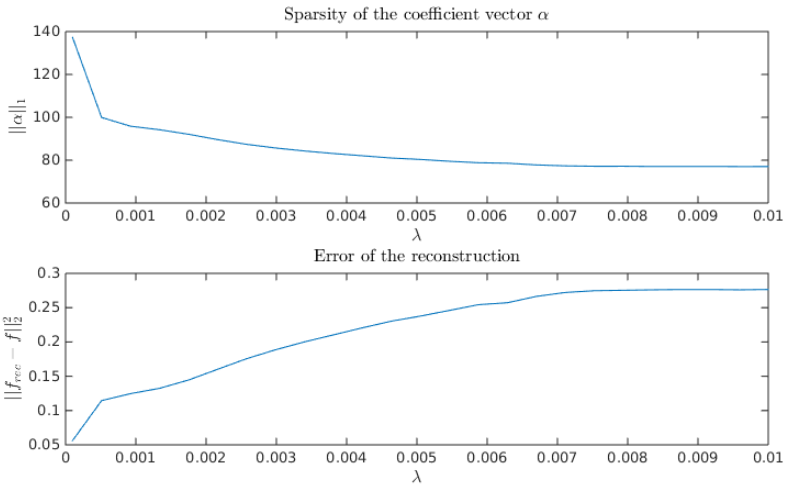
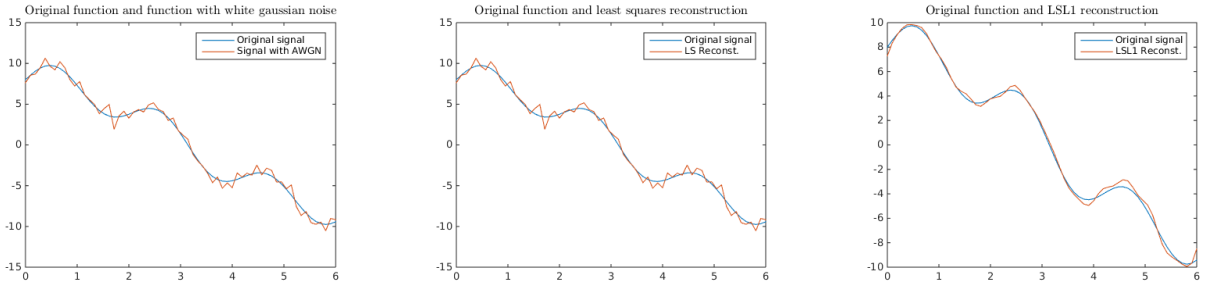


Figure 4.7.: *The behavior to the parameter λ in the L1LS reconstruction.*



(a) (b) (c)

Figure 4.8.: *Reconstruction with Gaussian white noise.*
(a) *Original signal and noisy one*
(b) *Original signal and Least squares reconstruction*
(c) *Original signal and L1LS reconstruction*

least squares reconstruction (4.5274).

Finding the optimal σ

To find the optimal relation between the radius, the number of Gaussians and the their σ we created a test. As a test signal we use the same signal as in 4.4 (a). We first start to iterate over the number of Gaussian centers from 7 to 19. For every number of centers we compute the reconstruction error with different division factors d_i, from 0.1 to 4.

$$\sigma_i = \frac{\frac{2 \cdot \text{radius}}{\text{number of gaussians}}}{d_i} \tag{4.4}$$

Afterwards we return the d_i with the lowest reconstruction error as the optimal factor. In Figure 4.9 (a) we can see the error for different number of Gaussians, in Figure 4.9 (b) we can see the used division factor, Figure 4.9 (c) shows the resulting number of zero elements form the coefficient vector for $\lambda = 0.01$ and the Figure 4.9 (d) show the basis components (8) multiplied by the reconstruction coefficients without adding them up. Summarizing we can say the best division factor is around 0.6.

4.2. 2D case

As we could see in the previous section the 1D reconstruction worked quite well. In this section we are going to try to reconstruct a 2D function and other figures. Because of the robustness of L1LS we are going to focus on that solver.

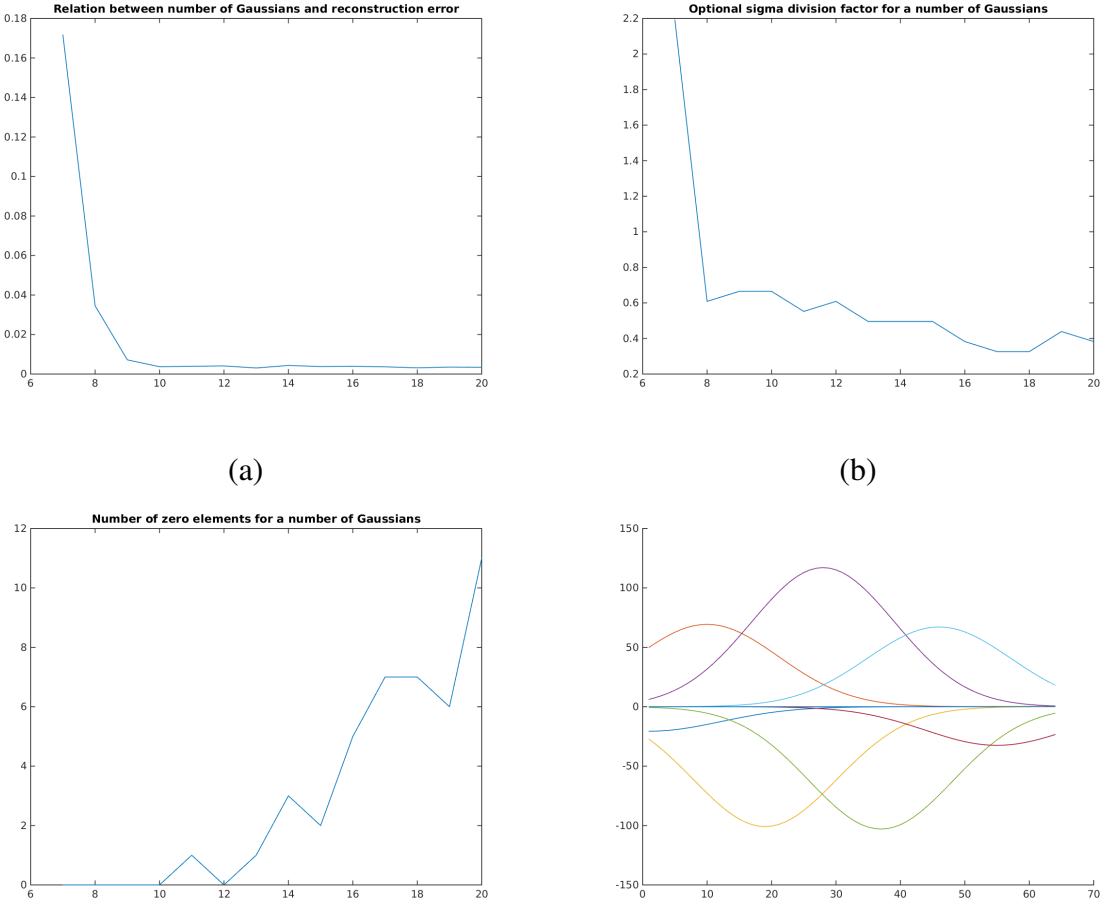
4.2.1. Testing on a function

We first need to define a 2D function $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R} \times \mathbb{R}$ where we can sample from:

$$f(\theta, r) = \begin{pmatrix} \cos(\theta) \cdot (r + \sin(5\theta)) \\ \sin(\theta) \cdot (r + \sin(5\theta)) \end{pmatrix} \tag{4.5}$$

We set $r = 5$ and the function is 256 times uniformly sampled. To reconstruct the function we use 10 centers and the neighbouring points inside a radius of 2. In Figure 4.10 (a) we can see the function points with normals and also the centers. The corresponding neighbourhood queries and their circles can be seen in Figure 4.10 (b).

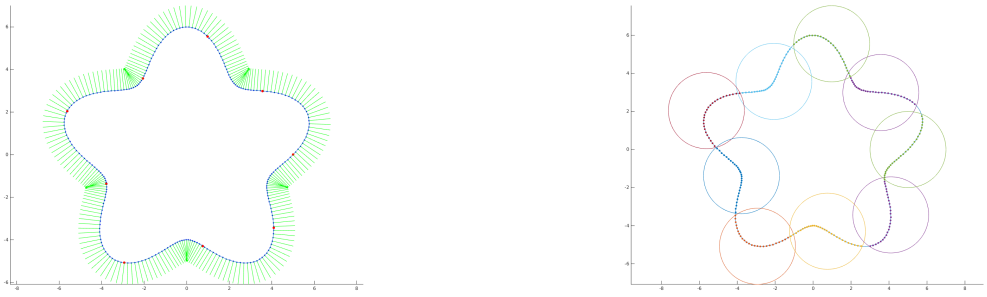
As mentioned in section 3.4.1 we need to compute the local coordinate system for every center. In this case we define the origin of the local coordinate system as the center of mass. In Figure 4.11 we can see what happens if we do not take the normal flipping problem into account. After computing the correct coordinate system, the local points are transformed into it and the reconstruction is computed see Figure 4.12. For this reconstruction we used 16 uniformly distributed Gaussian centers and $\sigma = 0.85$ and $\lambda = 1e - 5$. The local reconstruction can be transformed back to the global space with the inverse of the local basis and the mean point.



(a) (b)

(c) (d)

Figure 4.9.: (a) *Reconstruction error for different amount of Gaussians*
(b) *Optimal division factor for different amount of Gaussians*
(c) *Number of zero elements for different amount of Gaussians*
(d) *Basis multiplied by coefficients (No adding up)*



(a) (b)

Figure 4.10.: (a) *The function 4.5 and its normals.*
(b) *The center points and their neighbouring points*

(a) (b)

Figure 4.11.: (a) *The principal components of each centers neighbours* (b) *The correctly oriented axis.*

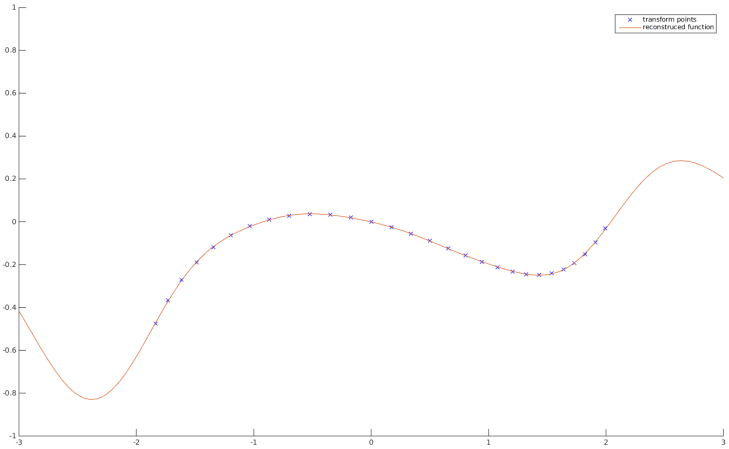
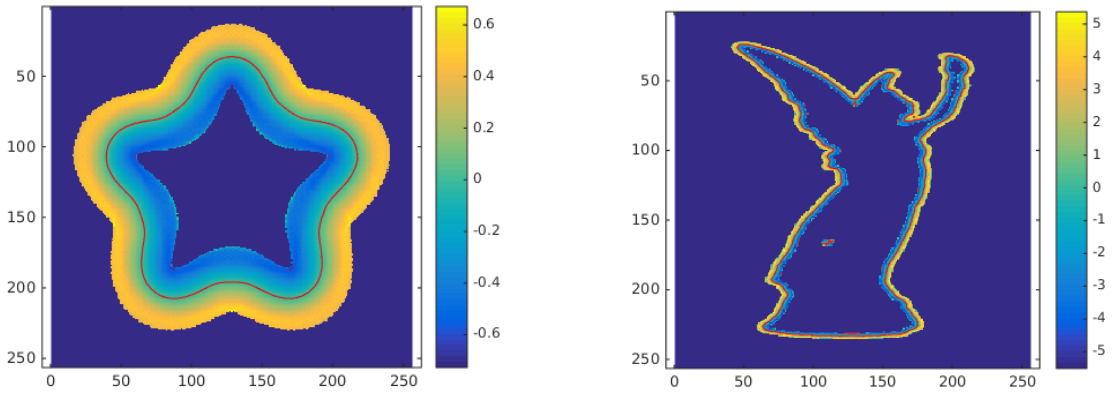


Figure 4.12.: *The local reconstruction of the function*



(a) (b)

Figure 4.13.: *Signed distance and zero set plots*
(a) *Reconstruction of the function 4.5*
(b) *Reconstruction of the 2d Lucy model*

4.2.2. Reconstruction with implicit function

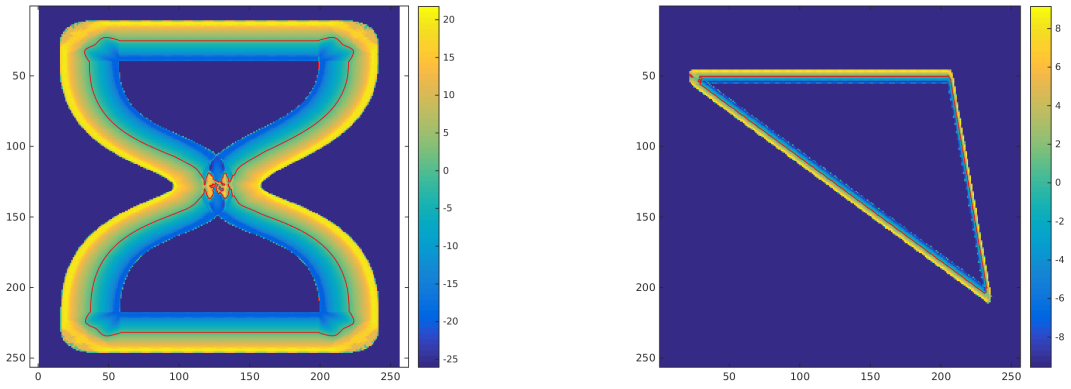
To get close to the 3D case we are going to reconstruct the function by isosurfacing the implicit signed distance function. Therefore, we create a grid as described in section 3.2.4. For extracting the zero set iso-line we use the marching squares algorithm, which is the simpler version of the marching cubes algorithm for the 2D case. The reconstruction of the function from the previous section can be seen in Figure 4.13 (a). Another reconstruction on a 2D version of the Lucy model from the Stanford scanning repository can be seen in Figure 4.13 (b). The 2D data is computed by first separating the foreground from the background and then using the marching squares algorithm to detect the contour of the foreground. The discrete representation is used to compute the required normals. Some examples can be found in A.4 and their reconstructions under A.5.

4.2.3. Results

In conclusion we would say that the 2D reconstruction with a Gaussian height field works quite well, but there are two main problems. At first, if the sampled curve points are not topologically equivalent to a line. This causes problems, because the resulting height field is very spiky and not smooth. This could appear if different parts of the curve are close to each other, see Figure 4.14 (a). The second problem occurs when the function has very sharp corners and the number of centers are too low. The results is that the corner get smoothed out see Figure 4.14 (b).

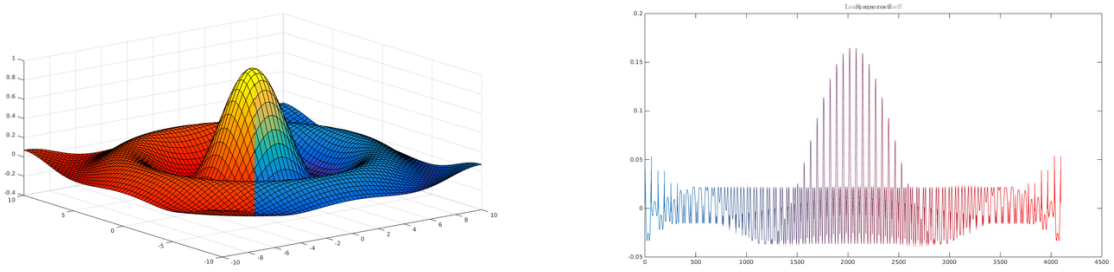
4.2.4. Testing on a height field

In 3D we are going to work with local height fields, and so we define a trigonometric function $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$.



(a) (b)

Figure 4.14.: (a) *Problem with a thin object part*
(b) *Problem with a really sharp corner.*



(a) (b)

Figure 4.15.: (a) *Reconstruction of the function 4.6 (blue) and original surface (red)*
(b) *Fading between LS coefficients (blue) and sparse ones (red)*

$$f(x,y) = \frac{\sin(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}} \tag{4.6}$$

which gets sampled uniformly 4096 times in range -10 to 10 to define the height field.

L_1 minimization with quadratic constraint

Based on the good performance in the 1D case we try to get the solution with the L1QC method. For the Gaussian matrix we use $\sigma = 0.3$. The reconstruction (see Figure 4.15 (a)) takes 167.298803 seconds and the total error is $1.0e-03$, which are quite good results. But the if we take a look at the resulting coefficients (Figure 4.15 (b)), we can see they are complete equal with the least squares solution and are not sparse any more.

L_1 regularized least squares minimization

For testing the L1LS method we use different values for λ ranging from 0.1 to 5. The other parameters are the same as in the section above. The reconstructed function and the comparison between obtained reconstructed coefficients and the least squares solution can be found in Figure A.6. It is clearly visible that the small frequencies of the function disappear and only the big peek stays. In conclusion we would say the L_1 regularized least squares solver outperforms the L_1 minimization with quadratic constraint solver, but it is much slower.

4.2.5. Adding the dictionary

The next step is to add the dictionary to the minimization. We create a 32 by 32 grid with 16 Gaussian centers and compute the reconstruction with the resulting training matrix $T \in \mathbb{R}^{16\times 673}$. The number of columns of T depends obviously on the number of grid points and on the radius. To find the best parameters we tried 12, 18, 24, 30 and 36 as dictionary sizes and for each the sparsity threshold of 3, 6, 9 and 12 see Figure A.8 (a). Finally, we chose a dictionary size of 30 and a sparsity threshold of 9. Figure A.8 (b) shows the convergence of the KSVD algorithm after 25 iterations. As a signal we used 1024 samples from $f(x) = \frac{\sin(8x)}{5} + 0.5 * \cos(x)$, which defines a valid height field. For testing the reconstruction with the dictionary, we use a small subset (46 points) of the samples and try to reconstruct the function. To simplify the problem we set the local coordinate axis to the global ones, which works in this case because the signal is already a height field. In Figure 4.16 (a) we can see the global signal (blue dots) with the subset (red crosses) and the query point with the radius. Figure 4.16 (b) shows the local reconstruction with and without a dictionary. The reconstruction error without a dictionary is 6.0032 and with a dictionary 2.4269.

Local coordinate system problem

If we use the local coordinate system for the reconstruction, we face the following problem. The computed local coordinate system of the subset inside the circle could significantly differ from the one from all points. In Figure 4.16 (c) we can seen the original points transform with respect to the local coordinate system of the subset (green) and the original points transform with respect to their own coordinate system (blue). The red line represents the reconstruction with subset of the samples.

Using the global frame

One simple way would be to fix the local coordinate frame to the global one. That would solve the problem with the local coordinate systems. But if we continue to get the signed distance function value by evaluating the reconstructed function at the query point, we could easily get wrong results, because the shortest distance to the function must not be at 0. In Figure 4.16 (d) you can see an example for that case. To find the shortest distance we sample the reconstructed function multiple times and use linear curve segments to approximate it between the points. Afterwards we use this representation to find the shortest distance to the curve. The sign of the

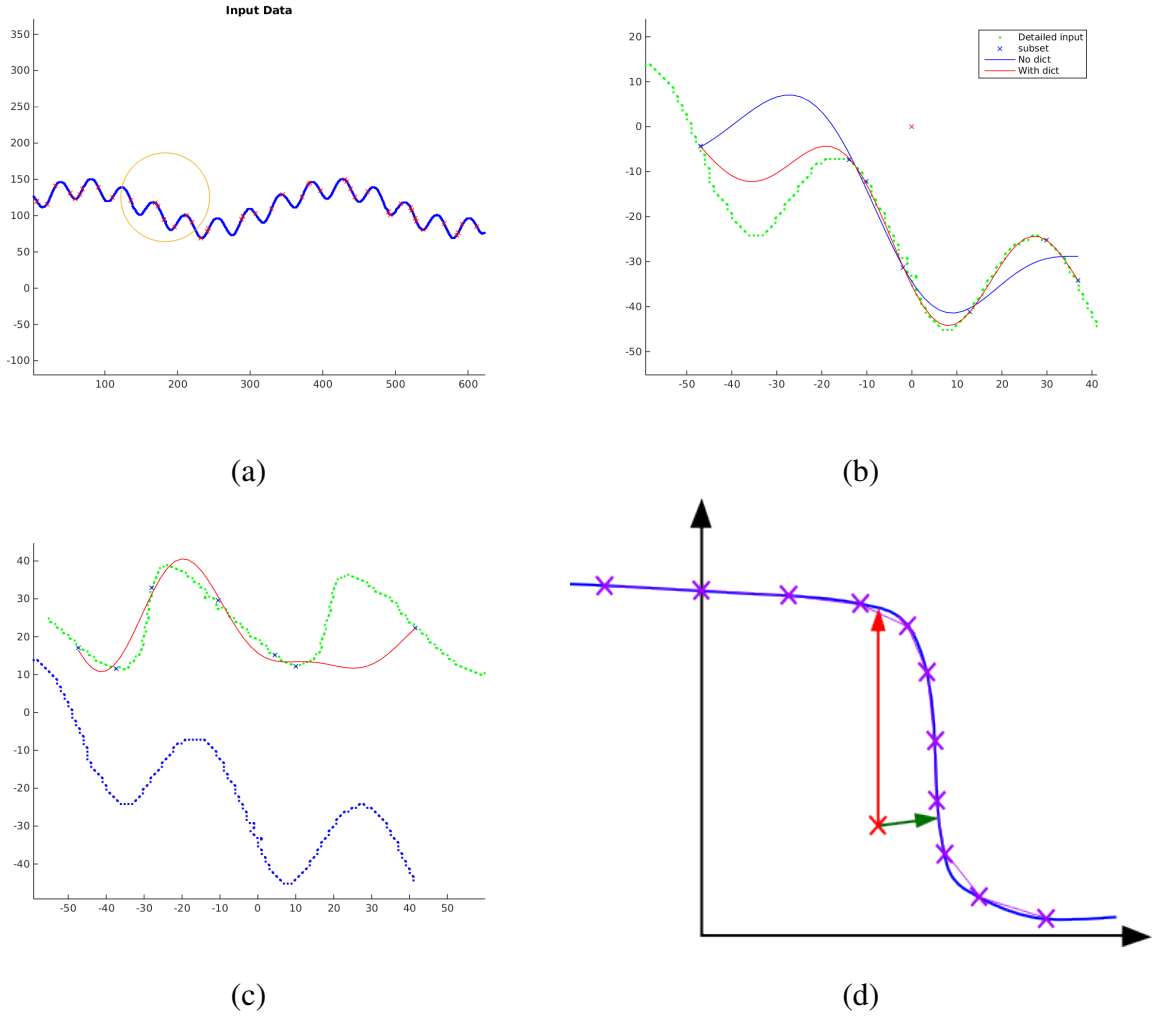
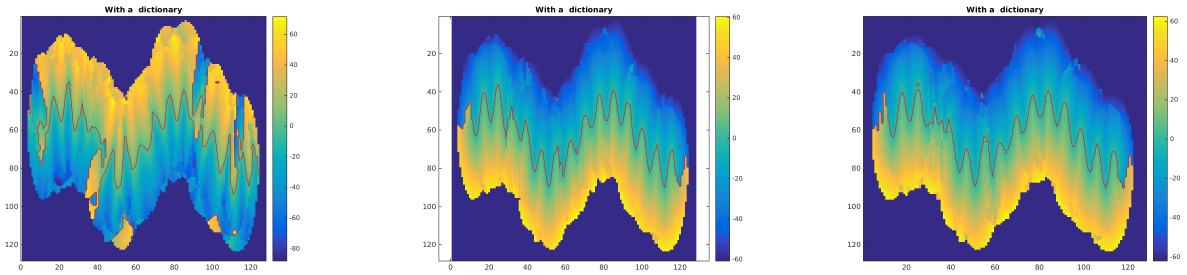


Figure 4.16.: (a) *Original signal (1024) and subset (46)*
(b) *Reconstruction with (err = 2.4269) and without (err = 6.0032) dictionary on subset*
(c) *Reconstruction with local coordinate systems (err = 5.1745)*
(d) *Problem if the local coordinate system is fixed to the global one.*



(a) (b) (c)

Figure 4.17.: (a) *Fixing problem with implicit reconstruction*
(b) *Implicit reconstruction with fixed frames*
(c) *Implicit reconstruction with stored frames*

distance is still defined by the evaluating the function. However, this only works if the signal is already a height field. Another approach is to remember the local frames and use them for the reconstruction with subset of samples.

Implicit reconstruction results

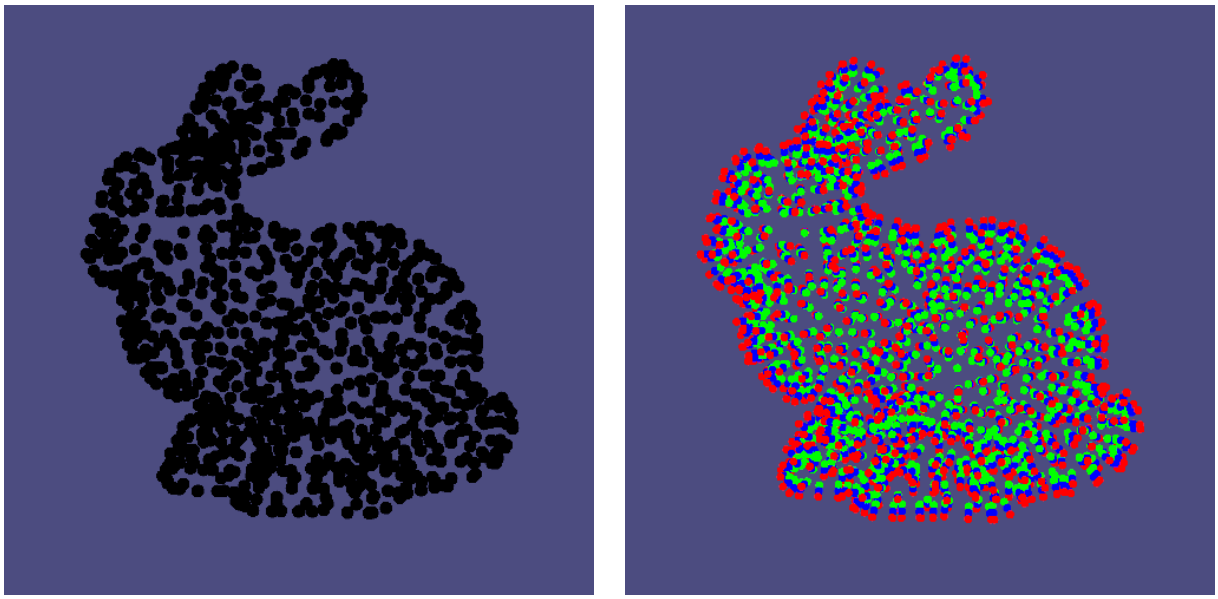
In Figure 4.17 (a) we can see the result if we don't take the fixing problem into account. In Figure 4.17 (b) we can see the reconstruction with fixed frames and Figure 4.17 (c) shows the reconstruction if we remember the local frames.

Results

The reconstruction with a precomputed dictionary works quite well. It is possible to reconstruct a signal with 95,5 % missing data. The runtime of the whole reconstruction process increase roughly speaking by a factor of 1.5. Notwithstanding the local coordinate system problem, we are going to lift the approach up to the 3D case.

4.3. 3D case

At this point we are going to test our approach in 3D. The implementation is done in C++ with the libigl framework [JP+15]. At first we are going to implement a normal moving least squares with different weighting function and polynomial basis to an arbitrary degree. To be able to use some Matlab routines we are going to create an interface, through which it is possible to communicate with the Matlab engine. Afterwards, we are going to optimize the code. Finally, we are going to add the local Gaussian height field and the dictionary.



(a) (b)

Figure 4.18.: (a) *1000 points from the Stanford bunny.*
(b) *The constraint points red - outside, blue - on the surface, green - inside*

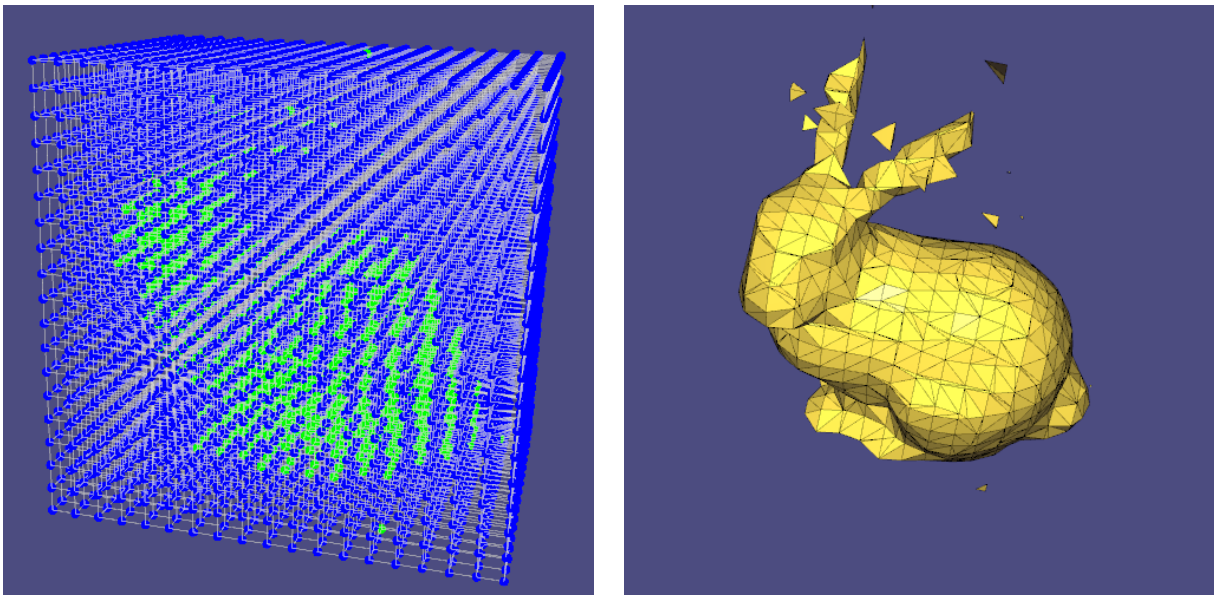
4.3.1. 3D Moving least squares

For the 3D MLS we created a simple acceleration structure, which divides the space in to a given number of cells (bins) and adds all points to the cell in which they lay in. For neighbouring queries we just need to find the bins inside or intersecting the query circle to get the points. To not get the trivial solutions trough the minimization procedure we have two possible constraints. First we add for every sample 2 new points in the direction of the normal with distance ϵ. The sample point gets the constraint value 0 and the two new points get minus or plus ϵ, depending on if they are inside or outside (see Figure 4.18). The second one is to enforce that the gradient of the reconstructed function and the surface normals match.

As described in 1 we create a grid, which is slightly bigger than the bounding box of the input sample points. For every grid point we compute the signed distance value. After all values are computed the we run the marching cubes algorithm to get the vertices and faces (see Figure 4.19 (a)). If there are reconstruction artifacts like small objects which are not connected to the main reconstruction, we can post process the result and extract the biggest connected component see Figure 4.19 (b). Some reconstructions can be found under A.7.

4.3.2. Optimization

With the aim to compute the results faster, also for bigger meshes, we considered different ways to optimize the basic framework. In the end of the section you can see a bar chart which compares the new implementation with the old one without any optimization in terms of run time.



(a) (b)

Figure 4.19.: (a) *Marching cubes grid with SSD value blue - positive and green - negative.*
(b) *The bunny reconstructed with resolution 20 and reconstructions artefacts.*

PCA

Some meshes are not really aligned the Euclidean axis, it makes sense to get the principal components and transform all input points and normals to get better results with the Marching cubes algorithm. Therefore we implemented a transform as an optional preprocessing step.

Matlab in C++

To be able to call Matlab routines we implemented an interfaces to the Matlab engine. Through that interface we can directly evaluate Matlab code or call function. This makes it possible to use Matlabs solvers to get a sparse solution of a linear system. Note that this does not contribute to the speed up, on the contrary solving system with Matlab is around 10 times slower than using C++ methods.

KD-tree

As mentioned in the previous section, we added simple uniform binning as an acceleration structure. This type of acceleration is not optimal for every kind of mesh, because it does not adapt to the distribution of the points. We replaced the bin with a KD-tree. The KD-tree is constructed during the initialization and is quicker for queries on large datasets, but also works well on small ones.

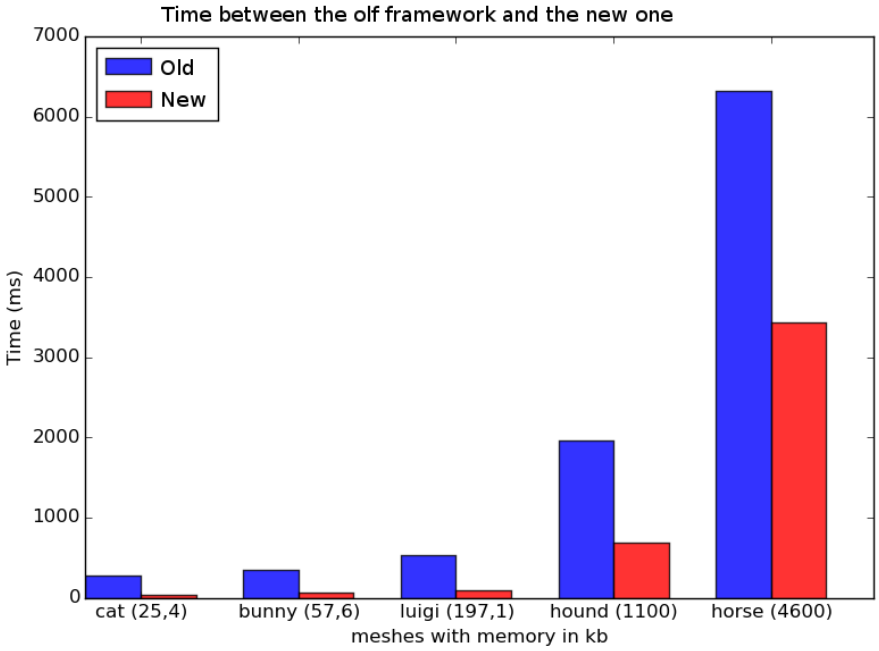


Figure 4.20.: *The run time before and after the optimization.*

Precomputing the basis

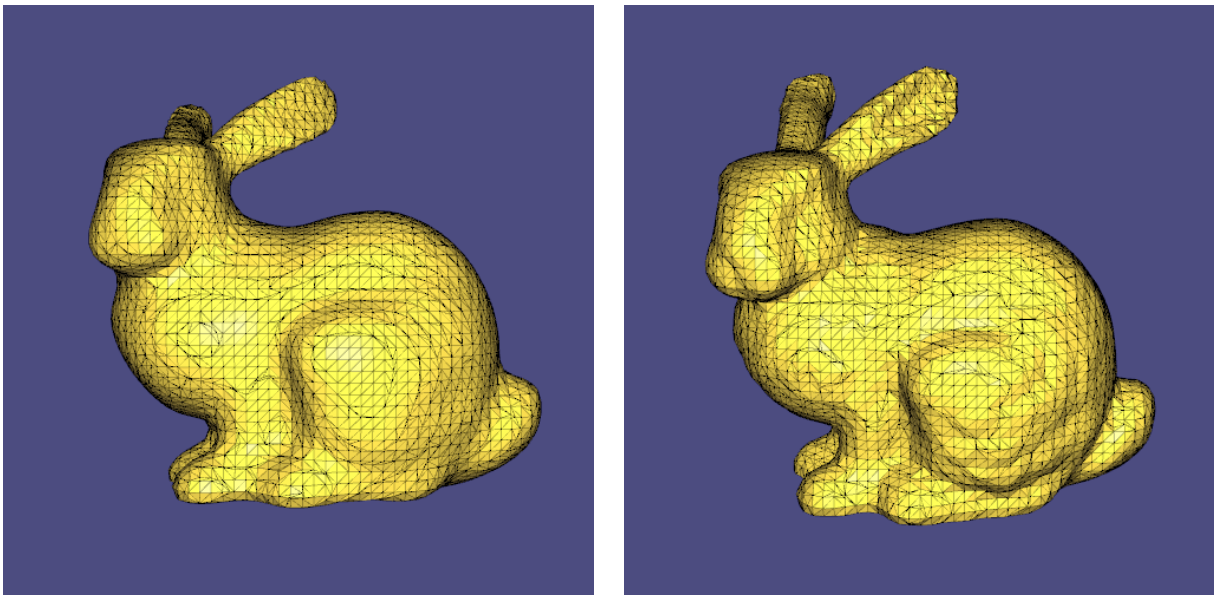
In the basic framework we compute the basis matrix every time after solving the MLS system. It is faster to precompute the basis for every vertex as a preprocessing step. This would be a big speed up if the grid for the marching cubes is really dense and surface points get used multiple times.

Thread building blocks

The last optimization is to evaluate the MLS function for every grid point in parallel. Because the grid point evaluation is independent of other grid points, it could be easily parallelized. In Figure 4.20 we can see the speed up for input of different sizes.

4.3.3. Gaussian height field

The MLS with a Gaussian height field (GHF) works in the same way as the MLS algorithm, the only difference is the computation of the implicit function value. The GHF method doesn't need any constraints. Our experiments have shown that the GHF method only works well if the surface doesn't contain thin parts. To avoid this problem the radius has to be smaller than the thinnest part and the grid resolution must also increase, which requires a denser sampling. However, in Figure 4.21 we can see the reconstruction using 50 as grid resolution with the MLS algorithm and normal constraint compared to the GHF reconstruction. The runtime for the GHF is 12871 milliseconds, which is about 6 times slower than the MLS method (2185) with polynomial degree 2.



(a) (b)

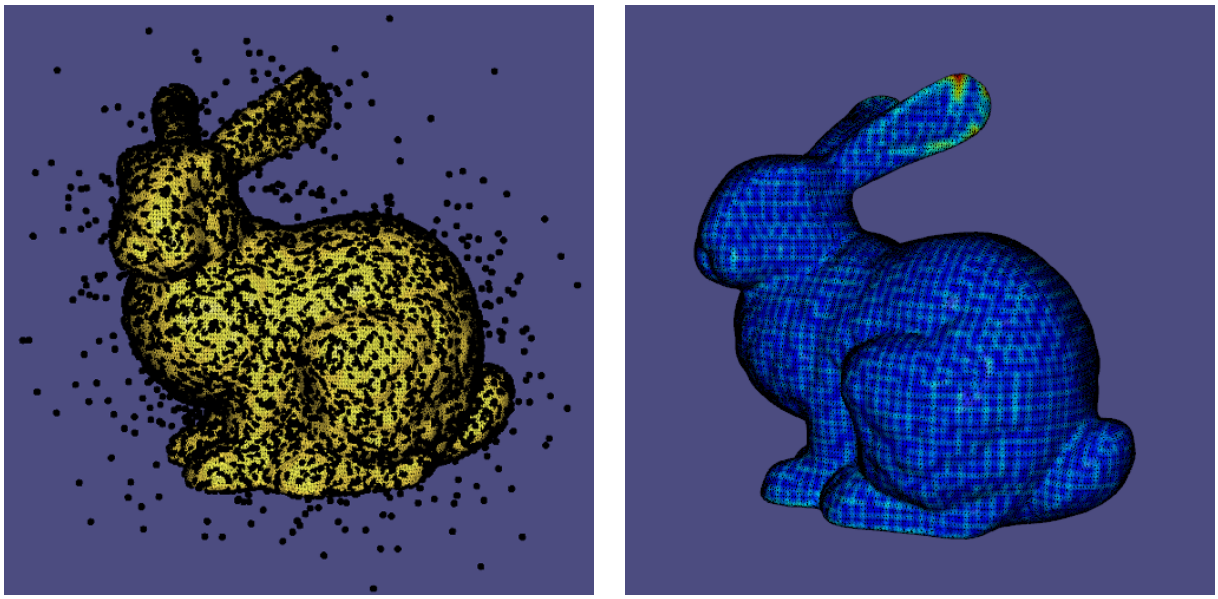
Figure 4.21.: (a) *Moving least squares reconstruction with normal constraint and polynomial basis of degree 2* (b) *Gaussian height field reconstruction with 16 center uniformly distribute on a rectangle.*

4.3.4. Adding the dictionary

similarly as in the 2D case we also add the dictionary to reconstruction. The dictionary is created in the same fashion as mentioned before. We still keep the problems, which appeared in 2D, in mind for the process.

Reconstruction with own dictionary

In the beginning we start simple and compute a dictionary of 30 atoms with a sparsity threshold of 9. Afterwards we resample our initial model with only 5 % of the amount of the initial samples. With the new test set and the dictionary we try to reconstruct the model. In Figure A.10 we can see that the reconstruction with dictionary much closer to the original model than the one without. Generally speaking, the reconstruction is also much faster, than without a dictionary. That comes from the fact, that the resulting linear system is already in a better coordinate system and the solver needs less iterations. In section A.2 we can see the detailed distribution of the runtime for evaluating one grid point without a dictionary. In Figure A.9 we can see the runtime for a different amount of samples with and without dictionary. In Figure A.13 we can see the surface patches corresponding to the 5 biggest eigenvalues (A.12).



(a) (b)

Figure 4.22.: (a) *Sampling of the Stanford bunny with 10000 points and 5 percent outliers and 10 percent noise*
(b) *The Stanford bunny coloured with the error scalar field*

4.3.5. Creating the data

Real world scanners do not fit our testing purposes since we cannot easily adapt their noise level. We created a small program to sample a discrete mesh. The program randomly chooses a triangle T according to the surface area and uniformly samples a point p_i inside T, by computing s and t with 2 random numbers $r_1$and r_2.

$$\begin{aligned} s &= 1 \cdot \sqrt{1 - r_1} \\ t &= (1 - s) \cdot r_2 \end{aligned} \tag{4.7}$$

Afterwards we compute the edges e_1 and e_2 outgoing from point p_0. We compute the final position in the following way:

$$p_i = p_0 + se_1 + te_2 \tag{4.8}$$

For the normal at point p_i we use Barycentric coordinates and the vertex normals to interpolate the normal n_i. It is also possible to add noise and outliers along the normal direction with a given probability see 4.22.

4.3.6. Results

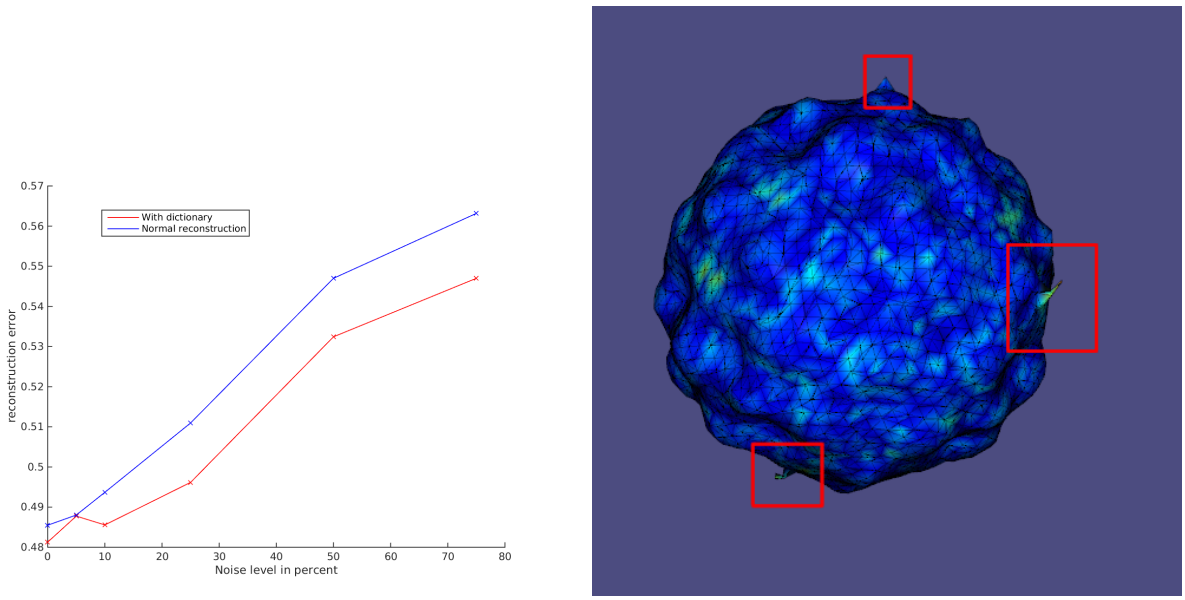
Summarizing, we can say, that the reconstruction works also good in the 3D case. To get more scientific error measurements, we wrote a small program, which computes the distance of each reconstructed vertex to the original mesh. The final error measurement is the mean distance of all vertices. We also used the computed scalar field to color the reconstructed mesh and see where the reconstruction is mostly off.

Under sampled data

As described in 4.3.4 we can reconstruct under sampled models, as long there are enough points in the the grid neighbourhood to compute a valid local coordinate system. If this condition is not fulfilled the reconstructed mesh is not a closed manifold any more. We can also observe that regions with high curvature get less samples, since we reduce the samples randomly, and the reconstruction error is much higher at those regions. In Figure 4.22 (b) we can see the original Stanford bunny color with the coloured with the error scalar field. We can see that most of the error appears at the ears of the bunny.

Noisy data

To see the benefit of the dictionary we added different amounts of noise to the data set and reconstructed the model with and without dictionary. In Figure 4.23 (a) we can clearly see that the error of the reconstruction with the dictionary is smaller. Additionally, we have to say that the error by its own does not really tell much, since the size of the point cloud is unknown. To get a better feeling for the point cloud dimensions, we provide the length of the diagonal of the bounding box. Also worth mentioning is that the reconstruction without dictionary creates after 25 % of noise unwanted handles (see 4.23 (b)) and other reconstruction artefacts, which are very visually disturbing. The original model and the reconstruction with a dictionary can be seen in A.11.



(a) (b)

Figure 4.23.: (a) *Reconstruction error plot (bounding boy diagonal 37.88875)*
(b) *Reconstruction artefacts without dictionary*

5

Conclusion and Outlook

We introduced a new surface reconstruction technique which combines classical dictionary learning and compressive sensing. First, the method computes a grid containing the signed distance to the scanned data and extracts the zero level surface. Our approach makes reconstruction with undersampled or noisy point clouds possible.

In chapter 1 we started with a quick introduction and explained the importance of surface reconstruction generally. In chapter 2 we gave a brief overview of the related work in the field of surface reconstruction, compressive sensing and dictionary learning. We discussed the advantages and disadvantages of the state-of-the-art methods and classified them into categories. In the next chapter we explained the theoretical background of our new reconstruction technique and step by step built the final minimization formulation. In our experiments we compared different sparse solvers in terms of run time, reconstruction error and sparsity. We found out that the L_1 regularized least squares minimization mostly fitted our needs. We started to test the reconstruction from the one dimensional to the three dimensional case and documented problems we encountered during this process and how we solved them. Furthermore, we did a lot of testing to find the best relationship between the available parameters and found those few intuitive ones for the user. Finally, we can say that our method can keep up with the state-of-the-art methods with some requirements, like remembering and using the local coordinate frames. We contribute a modular reconstruction framework implemented in C++, which allows easily to change or extent different parts, like the weighting function or the solver, from the reconstruction process.

Future work tackles the problem of the local frame approximation. To get more stable reconstructions, the local frames should not vary much if the number of points is decreased or noise is added to the model. A possible way would be to take the normals into account and use them to compute the local coordinate system. Furthermore, the neighbourhood querying needs to be improved, since in thin areas we get very spiky height fields resulting from the fact, that the

local surface is not topologically equal to a disc. This problem could be solved by taking the nearest point as a start point and using very small sphere radii for the queries to incrementally add points to the local points, if they are still in the demanded search radius. This approach would solve the problem, but also drastically increase the run time. Another suggestion to this problem would be to classify this situation as a problematic scenario and only use the samples of the closer side. However, also in sharp regions like in corners or edges the method fails to represent those and causes oversmoothing. To tackle this problem a different set of basis could be used, which can represent sharp features better than Gaussians. Finally, we want to create a better dictionary, which is constructed with training data from a big amount of different point clouds with different features and find out if we could get rid of some problems by optimizing the used dictionary.

A

Appendix

A.1. Detailed run time information of different sparse solvers.

Table A.1.: *Detailed run time and mean squared error table of different sparse solvers.*

Nr	Algorithm	Run time (ms)	MSE	programming language
1	Least squares	3	0.3774	Matlab
2	L_1 minimization with quadratic constraint	66	2.2507e-08	Matlab
3	L_1 regularized least squares	358	1.4291e-07	Matlab
4	Approximate Message Passing	9	3.5591e-09	C++
5	Basis Pursuit	96	7.8125e-13	C++
6	Compressive Sampling Matching Pursuit	14	1.5228e-10	C++
7	Expectation Maximization Belief Propagation	6	2.1085e-09	C++
8	Orthogonal Matching Pursuit	22	1.2734e-12	C++
9	Regularized Orthogonal Matching Pursuit	4	0.0143	C++
10	Smoothed L0	98	2.5474e-08	C++
11	Subspace Pursuit	17	1.1719e-14	C++

Algorithm 2 Learning a dictionary with the K-SVD algorithm (3.26)

▷ % Initialize the dictionary $D^{(0)}$ with normalized columns%

J = 1

▷ % Compute grid inside the bounding box with a given resolution%

while not converged or max iterations reached **do**

▷ % Sparse coding state%

for $i = 0$ to N **do**

▷ % Use a pursuit algorithm to compute x_i%

$\min_{x_i \in \mathbb{R}^d} ||Dx_i - t_i||_2^2 \; s.t. \; ||x_i||_0 < T_0$

end for

▷ % Dictionary update state%

for every column k in $D^{(j-1)}$ **do**

▷ % Update atom d_k%

▷ % Find all signals, which use d_i%

$\omega_k = \{i | 1 \leq i \leq N, x_T^k \neq 0\}$

▷ % Compute the overall representation error matrix%

$E_k = T - \sum_{j \neq k} d_j x_T^j$

▷ % Restrict E_k by choosing only the columns corresponding to ω_k%

E_k^R take subset from E_k

▷ % Apply SVD decomposition%

$E_k^R = U \Delta V^T$

▷ % Set d_k to the first first column of U%

$d_k = U(:, 1)$

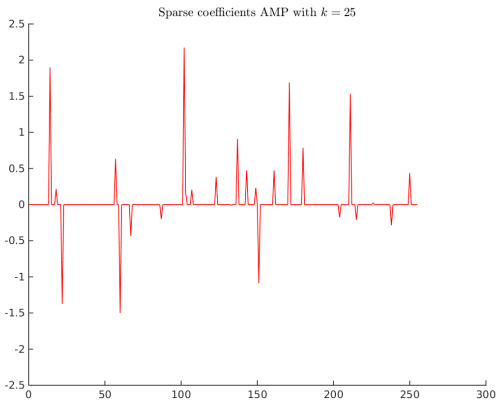
▷ % Set x_R^k to the first first column of V multiplied by the biggest eigenvalue%

$x_R^k = \Delta(1,1) \cdot V(:, 1)$

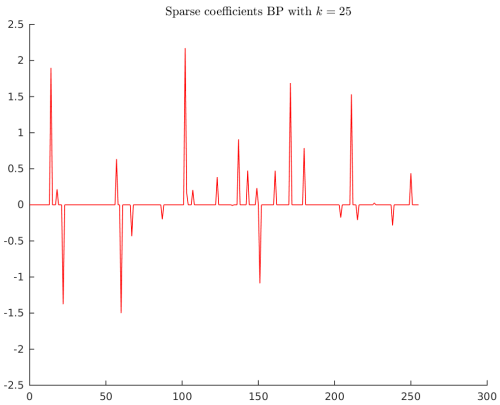
$J = J + 1$

end for

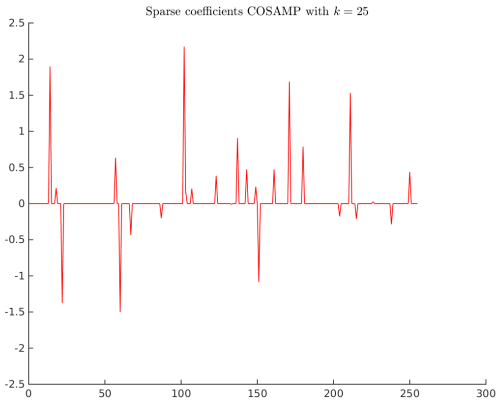
end while



(a)

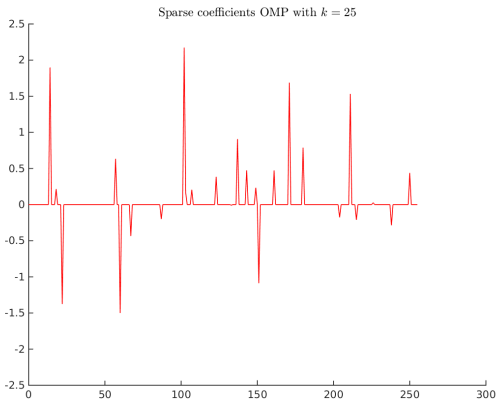


(b)

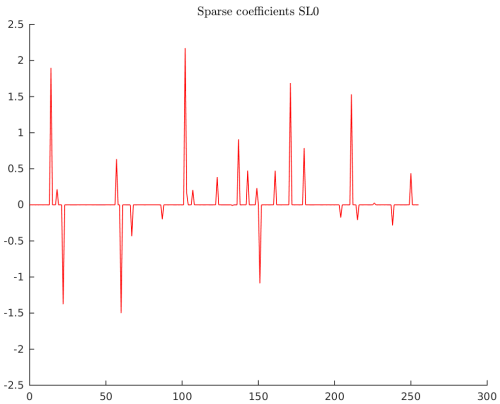


(c)

(d)

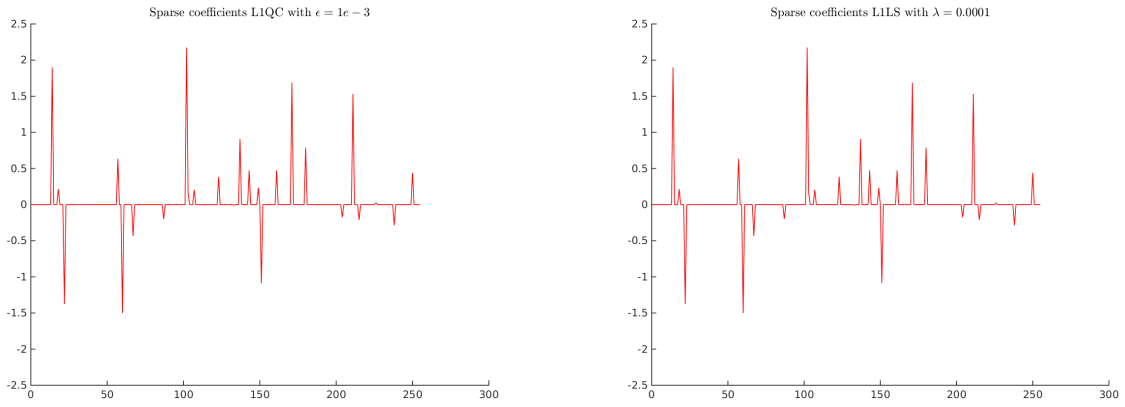


(e)



(f)

Figure A.1.: *1D reconstruction.* (a) *Approximate Message Passing* (b) *Basis Pursuit* (c) *Subspace pursuit reconstruction.* (d) *Compressive Sampling Matching Pursuit* (e) *Expectation Maximization Belief Propagation* (f) *Smoothed L0*



(a) (b)

Figure A.2.: *1D reconstruction.* (a) *L1 minimization with equality constraint* (b) *L1 regularized least squares*

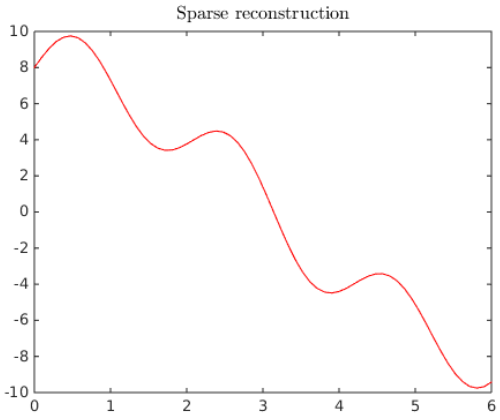
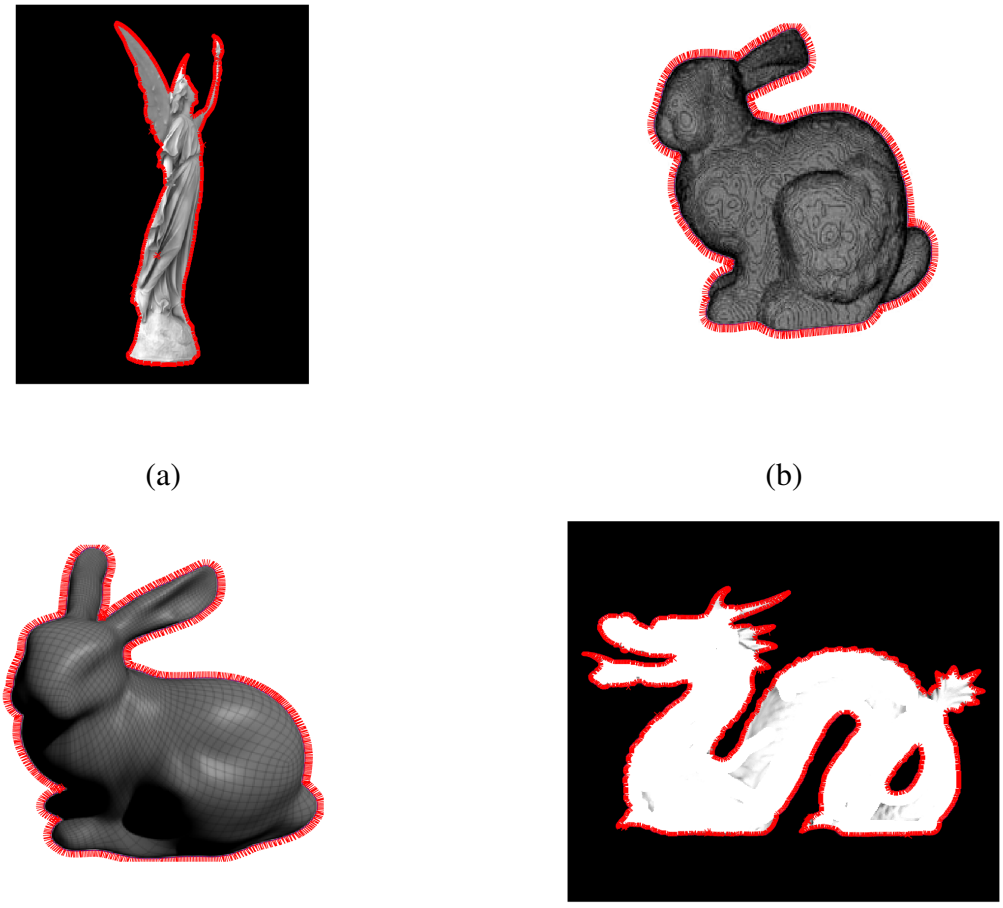


Figure A.3.: *L_1 regularized least squares reconstruction*



(a) (b)

(c) (d)

Figure A.4.: *Computing 2D data and normals out of images.* (a) *Lucy* (b) *Bunny* (c) *Bunny from different view point* (d) *Dragon*

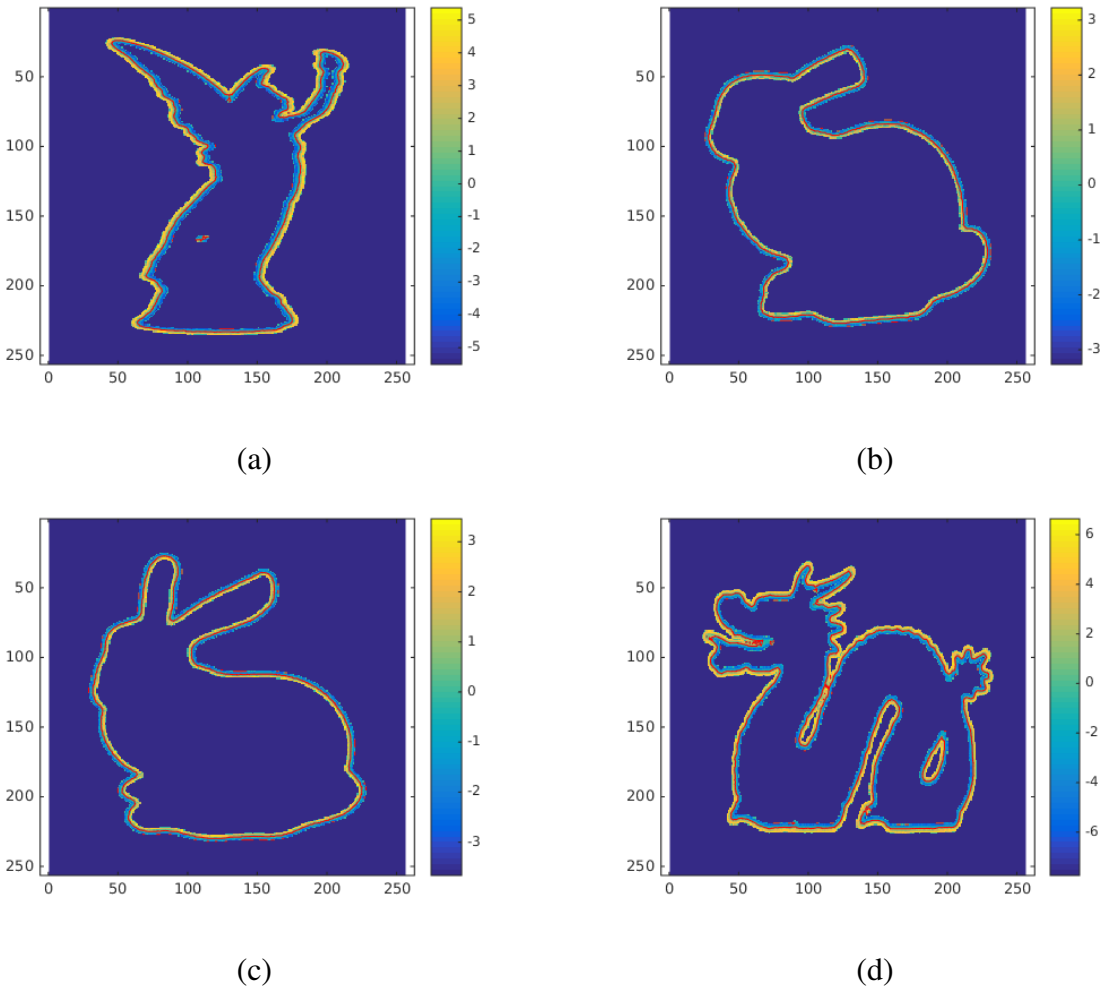


Figure A.5.: *Reconstructing the models.* (a) *Lucy* (b) *Bunny* (c) *Bunny from different view point* (d) *Dragon*

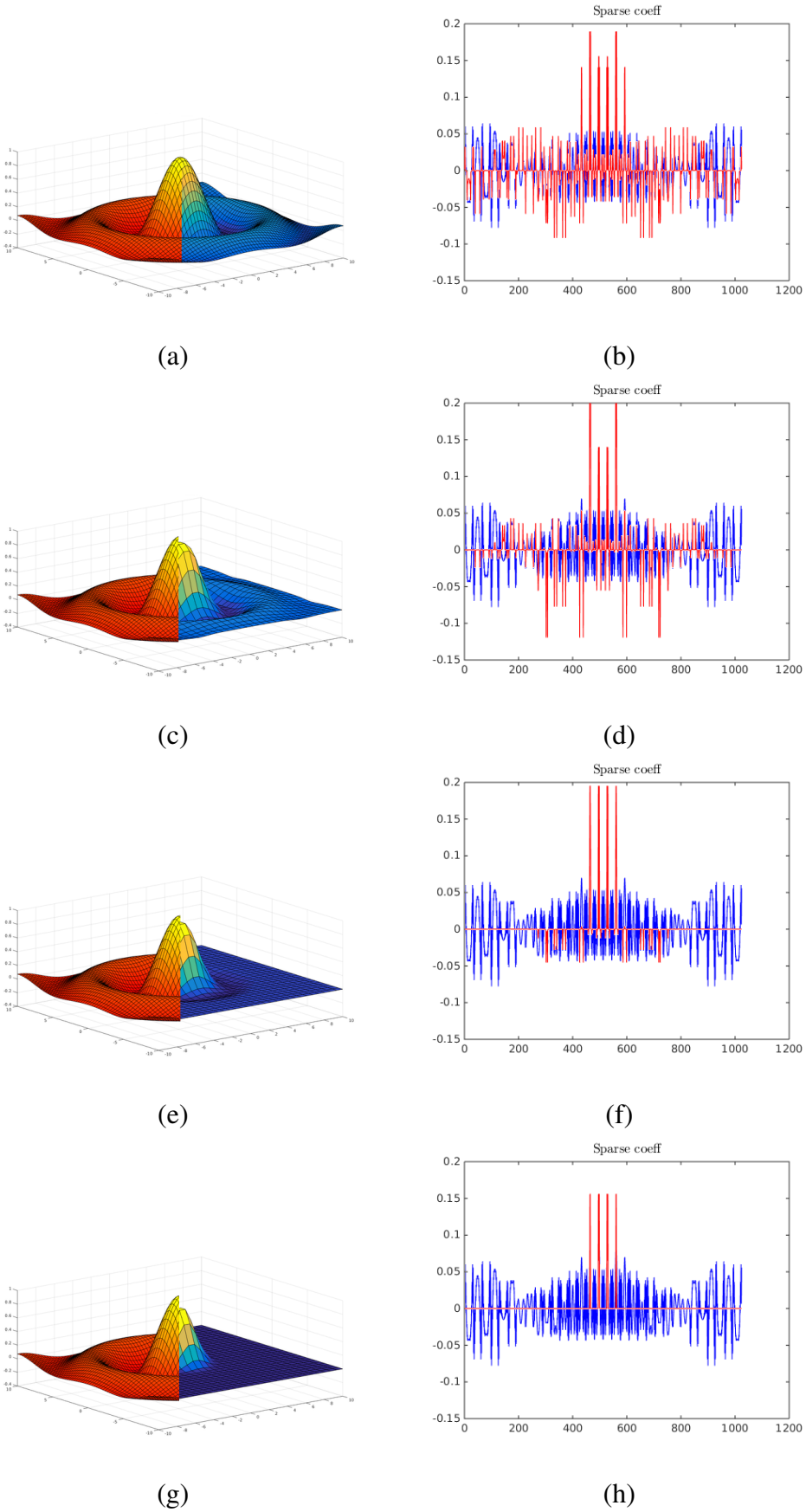
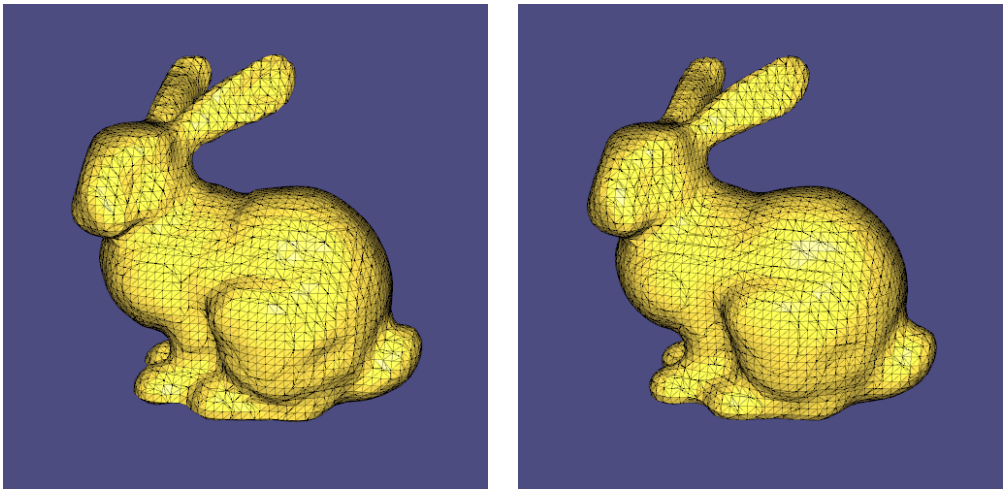
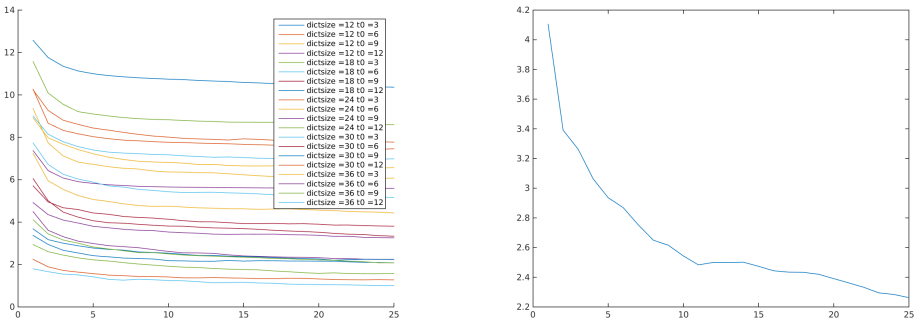


Figure A.6.: *Height field reconstructions with L1LS minimization.* (a) *Reconstruction (blue) and original (red)* (b) $\lambda = 0.1$ *error = 0.2772* (c) *Reconstruction (blue) and original (red)* (d) $\lambda = 1.0$ *error = 1.5574* (e) *Reconstruction (blue) and original (red)* (f) $\lambda = 2.5$ *error = 2.8596* (g) *Reconstruction (blue) and original (red)* (h) $\lambda = 5.0$ *error = 3.5322*



(a) (b)

Figure A.7.: (a) *Reconstruction with additional points constraint* (b) *Reconstruction with normal points constraint*



(a) (b)

Figure A.8.: (a) *Convergence for different parameters* (b) *Convergence after 25 iterations for dictionary size 30 and sparsity threshold 9*

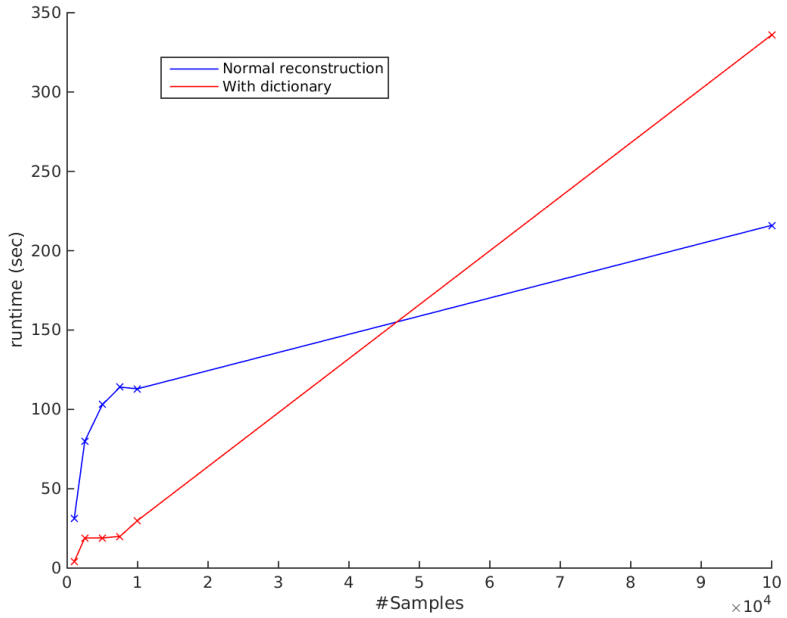


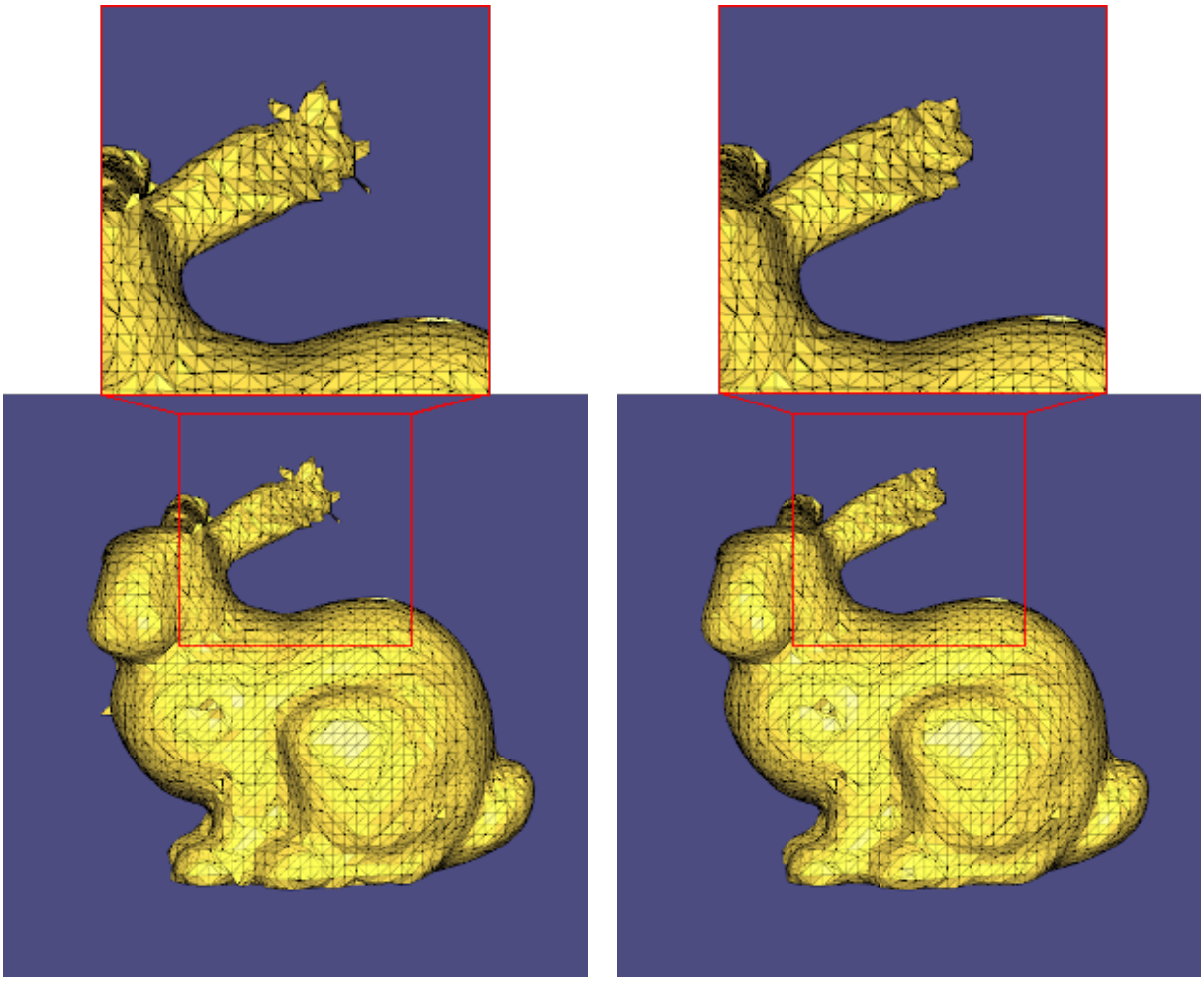
Figure A.9.: *Runtime for the reconstruction with(red) and without (blue) dictionary for a different amount of samples.*

A.2. Detailed run time information for the reconstruction

The following table shows the runtime for the different part of the reconstruction function.

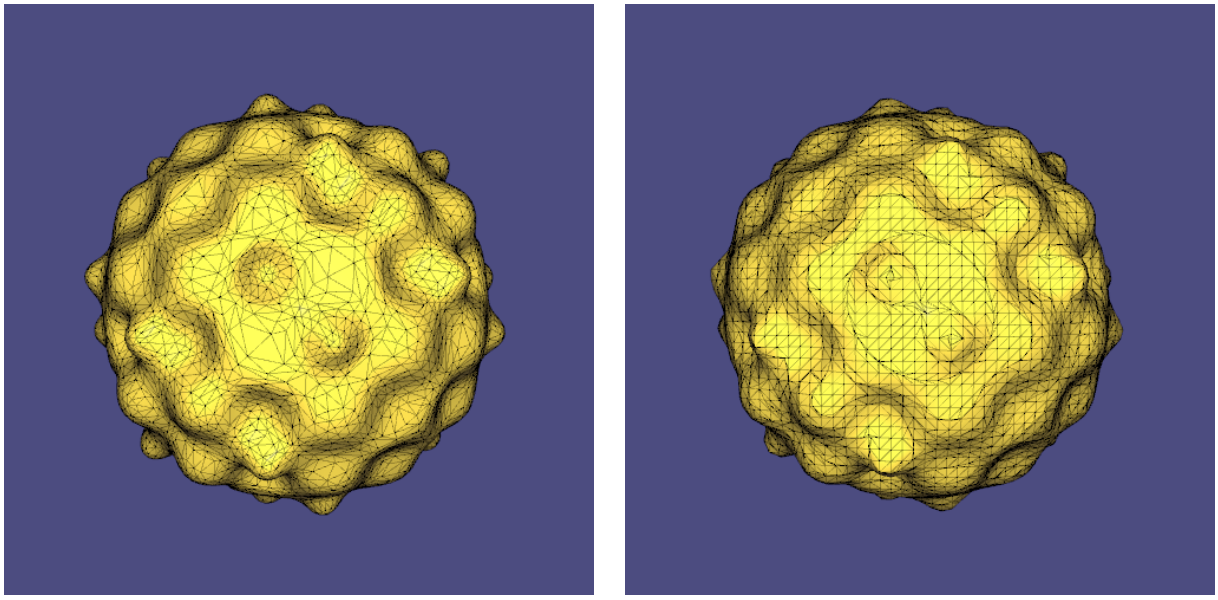
Table A.2.: *Detailed run time table of reconstruction process.*

Part	Milliseconds (avg)	percent
Allocate the memory	1.51	0.032
Computing the weights	2.97	0.064
Transforming to local frames	24.01	0.52
Computing the Gaussian matrix	84.13	1.82
Setting up the linear system	1138.46	24.7
Solving the system with L_1 norm	3356.6	72.864



(a) (b)

Figure A.10.: (a) *Reconstruction with 5 % percent of initial points without dictionary*
(b) *Reconstruction with a dictionary*



(a) (b)

Figure A.11.: (a) *Original mesh*
(b) *Reconstruction with dictionary*

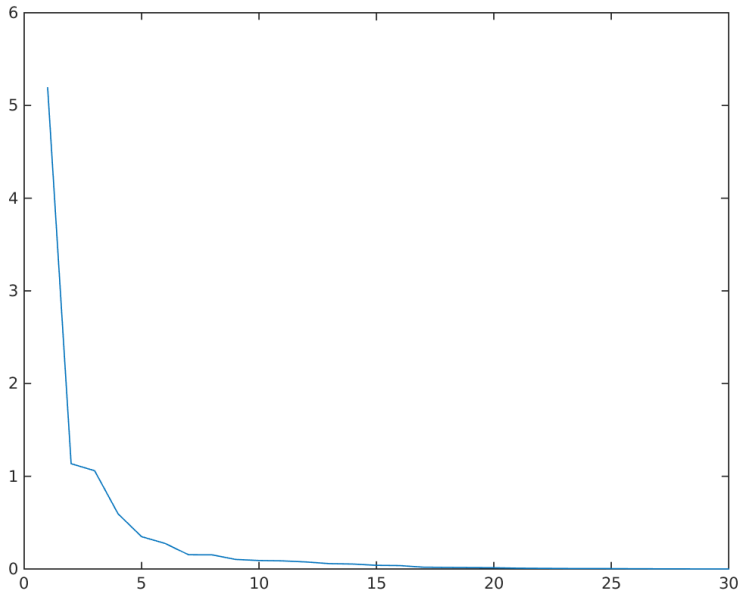


Figure A.12.: *The eigenvalues of the dictionary created on the bunny data set.*

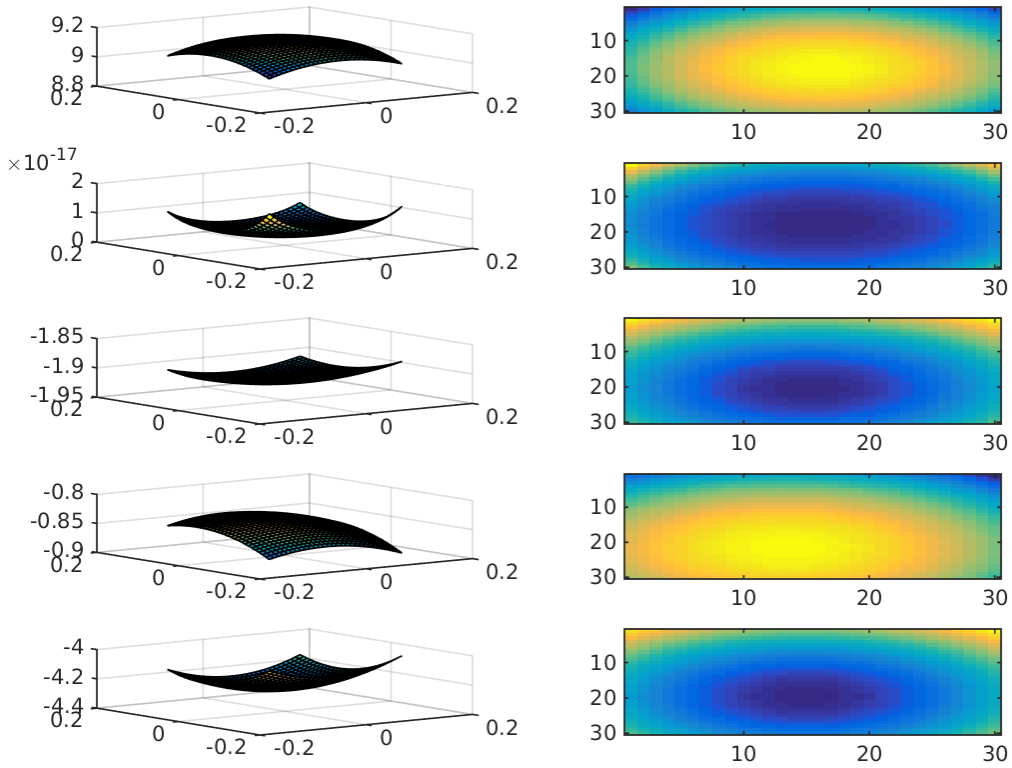


Figure A.13.: *Patches from the bunny data set corresponding to the 5 biggest eigenvalues A.12(top to bottom). Please notice, that every plot has its own coordinate system.*

Bibliography

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

[AEB06] M. Aharon, M. Elad, and A. Bruckstein. Svdd: An algorithm for designing overcomplete dictionaries for sparse representation. *Trans. Sig. Proc.*, 54(11):4311–4322, November 2006.

[ASGCO10] Haim Avron, Andrei Sharf, Chen Greif, and Daniel Cohen-Or. l1 sparse reconstruction of sharp point set surfaces. *ACM Trans. Graph.*, 29(5):135:1–135:12, November 2010.

[BPC+11] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3(1):1–122, January 2011.

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

[CG06] Frederic Cazals and Joachim Giesen. Delaunay triangulation based surface reconstruction: Ideas and algorithms. In *EFFECTIVE COMPUTATIONAL GEOMETRY FOR CURVES AND SURFACES*, pages 231–273. Springer, 2006.

[Cle79] William S. Cleveland. Robust locally weighted regression and smoothing scatter-

plots. *Journal of the American Statistical Association*, 74:829–836, 1979.

[CR05a] Emmanuel C and Justin Romberg. l1-magic: Recovery of sparse signals via convex programming, 2005.

[CR05b] Emmanuel Candes and Justin Romberg. l1-magic: Recovery of sparse signals via convex programming. 4, 2005.

[CRT05] Emmanuel C, Justin Romberg, and Terence Tao. Stable signal recovery from incomplete and inaccurate measurements, 2005.

[CWB08] Emmanuel C, Michael B. Wakin, and Stephen P. Boyd. Enhancing sparsity by reweighted l1 minimization, 2008.

[CX04] Long Chen and Jinchao Xu. Optimal Delaunay triangulations. *Journal of Computational Mathematics*, 22(2):299–308, 2004.

[DCV14] J. Digne, R. Chaine, and S. Valette. Self-similarity for accurate compression of point sampled surfaces. *Computer Graphics Forum*, 33(2):155–164, 2014. Proceedings of Eurographics 2014.

[DG01] Tamal K. Dey and Joachim Giesen. Detecting undersampling in surface reconstruction. In *Proceedings of the Seventeenth Annual Symposium on Computational Geometry*, SCG '01, pages 257–263, New York, NY, USA, 2001. ACM.

[DMSL11] Julie Digne, Jean-Michel Morel, Charyar-Mehdi Souzani, and Claire Lartigue. Scale space meshing of raw data point sets. *Computer Graphics Forum*, pages no–no, 2011.

[Fed15] Prof. Federico. The mathematical derivation of least squares, Jul 2015. `http://isites.harvard.edu/fs/docs/icb.topic515975.files/OLSDerivation.pdf`.

[Fou14] Chris Fougner. Pogs âĂŞ proximal operator graph solver. 2014.

[Geb12] Rene Gebel. A portable c++ compressed sensing library. 2012.

[GG07] Gaël Guennebaud and Markus Gross. Algebraic point set surfaces. *ACM Trans. Graph.*, 26(3), July 2007.

[GJY10] Dongdong Ge, Xiaoye Jiang, and Yinyu Ye. A note on complexity of lp minimization, 2010.

[HDD+92] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. *SIGGRAPH Comput. Graph.*, 26(2):71–78, July 1992.

[JP+15] Alec Jacobson, Daniele Panozzo, et al. libigl: A simple C++ geometry processing library, 2015. http://libigl.github.io/libigl/.

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

[KKB07] Kwangmoo Koh, Seung-Jean Kim, and Stephen Boyd. An interior-point method for large-scale l1-regularized logistic regression. *J. Mach. Learn. Res.*, 8:1519–1555, December 2007.

[KKB08] Seung-Jean Kim Kwangmoo Koh and Stephen Boyd. Simple matlab solver for l1-regularized least squares problems. 2008.

[KSO04] Ravikrishna Kolluri, Jonathan R. Shewchuk, and James F. O'Brien. Spectral surface reconstruction from noisy point clouds. In *Symposium on Geometry Processing*, pages 11–21. ACM Press, July 2004.

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

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

[Lev03] D. Levin. *Mesh-independent surface interpolation*. Spinger-Verlag, 2003.

[Lev04] Marc Levoy. The digital michelangelo project: 3d scanning of large statues. 2004.

[LS81] P. Lancaster and K. Salkauskas. Surfaces generated by moving least squares methods, 1981.

[MAT12] MATLAB. *version 8.04.0 (R2012a)*. The MathWorks Inc., Natick, Massachusetts, 2012.

[MBZJ09] Hosein Mohimani, Massoud Babaie-Zadeh, and Christian Jutten. A fast approach for overcomplete sparse decomposition based on smoothed l0 norm. *Trans. Sig. Proc.*, 57(1):289–301, January 2009.

[MZ93] S.G. Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *Trans. Sig. Proc.*, 41(12):3397–3415, December 1993.

[Nea04] Andrew Nealen. An as-short-as-possible introduction to the least squares, weighted least squares and moving least squares methods for scattered data approximation and interpolation. *Computer Graphics Forum*, 3(2), 2004.

[SMW+07] Yoav Sharon, Student Member, John Wright, Student Member, Yi Ma, and Senior Member. Computation and relaxation of conditions for equivalence between âĎŞ 1 and âĎŞ 0 minimization,âĂİ csl. Technical report, 2007.

[TG07] J. A. Tropp and A. C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Trans. Inf. Theor.*, 53(12):4655–4666, December 2007.

[Til14] Andreas M. Tillmann. On the computational intractability of exact and approximate dictionary learning. *CoRR*, abs/1405.6664, 2014.

[WYL+14] Ruimin Wang, Zhouwang Yang, Ligang Liu, Jiansong Deng, and Falai Chen. Decoupling noise and features via weighted l1-analysis compressed sensing. *ACM Trans. Graph.*, 33(2):18:1–18:12, April 2014.

[XZZ+14a] Shiyao Xiong, Juyong Zhang, Jianmin Zheng, Jianfei Cai, and Ligang Liu. Robust surface reconstruction via dictionary learning. *ACM Transactions on Graphics (Proc. SIGGRAPH Aisa)*, 33, 2014.

[XZZ+14b] Shiyao Xiong, Juyong Zhang, Jianmin Zheng, Jianfei Cai, and Ligang Liu. Robust surface reconstruction via dictionary learning. *ACM Trans. Graph.*, 33(6):201:1–201:12, November 2014.