LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

3D Surface Modeling From Point Clouds

Examiners: Professor Joni Kämäräinen
Supervisor: Arto Kaarna D.Sc. (Tech.)

ABSTRACT

Lappeenranta University of Technology
Department of Information Technology

Jukka Lankinen

3D Surface Modeling Using Point Clouds

Thesis for the Degree of Master of Science in Technology

2009

66 pages, 43 figures, 3 tables and 2 appendices.

Keywords: 3D model, reconstruction, point clouds, NURBS

The goal of this thesis is to implement software for creating 3D models from point clouds. Point clouds are acquired with stereo cameras, monocular systems or laser scanners. The created 3D models are triangular models or NURBS (Non-Uniform Rational B-Splines) models. Triangular models are constructed from selected areas from the point clouds and resulted triangular models are translated into a set of quads. The quads are further translated into an estimated grid structure and used for NURBS surface approximation. Finally, we have a set of NURBS surfaces which represent the whole model.

The problem wasn't so easy to solve. The selected triangular surface reconstruction algorithm did not deal well with noise in point clouds. To handle this problem, a clustering method is introduced for simplificating the model and removing noise. As we had better results with the smaller point clouds produced by clustering, we used points in clusters to better estimate the grids for NURBS models. The overall results were good when the point cloud did not have much noise. The point clouds with small amount of error had good results as the triangular model was solid. NURBS surface reconstruction performed well on solid models.

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Tietotekniikan osasto

Jukka Lankinen

3-ulotteisen pinnan muodostus pistejoukoista

Diplomityö

2009

66 sivua, 43 kuvaa, 3 taulukkoa ja 2 liitettä.

Hakusanat: 3D-mallit, pinnanmuodostus, pistepilvet, NURBS

Diplomityön tavoitteena on tehdä sovellus, joka kykenee tekemään 3D-malleja pistejoukoista. Pistejoukkoja saadaan erilaisilta laitteilta, kuten stereokameroilta, yksikameraisista järjestelmistä ja laser-skannereilta. 3D-malleja, mitä työssä pyritään luomaan, ovat kolmiopohjaiset mallit ja NURBS-mallit. Kolmiomalleja muodostetaan tietyiltä alueilta pistejoukoista ja tätä kolmiomallia käytetään edelleen luomaan neliöpohjainen malli yhdistämällä kolmioita toisiinsa. Nämä neliöt muutetaan arvioiduksi verkkorakenteeksi. Tälle verkkorakenteelle sovitetaan NURBS-pinta ja kaikki NURBS-pinnat yhdessä luovat NURBS-mallin.

Työn ratkaisuun vaadittiin useita toimenpiteitä, sillä valittu kolmiointialgoritmi ei toimi häiriölliselle pistejoukolle. Tätä ongelmaa ratkaisemaan toteutettiin klusterointi-menetelmä, joka vähentää pinnan epätasaisuutta. Kun pistepilviä on pienennetty klusteroinnilla, saamme entistä parempia tuloksia kolmiointimenetelmällä. Parempi tulos kolmioinnissa johtaa myös laadultaan parempaan NURBS-malliin, koska mallissa on vähemmän reikiä, jotka haittaavat NURBS:n neliömallin tekoa.

CONTENTS

APPENDICES

ABBREVIATIONS AND SYMBOLS

AMLS	Adaptive Moving-Least-Squares
CAD	Computer-aided design
CGAL	Computational Geometry Algorithms Library
EKF	Extended Kalman Filter
EMST	Euclidian minimum spanning tree
GPU	Graphics processing unit
NURBS	Non-Uniform Rational B-Splines
OFF	Object File Format
PAT	Patch Adjacency Table
POV-ray	Persistence of Vision Raytracer
RANSAC	RANdom SAmple Consensus
SLAM	Simultaneus Localization and Mapping
UI	User interface
VRML	Virtual Reality Markup Language

$\mathbb{E}^3$	3-dimensional Euclidian space
e', e	Epipoles
$\mathbb{X}$	3D Point in Euclidian space
x, x'	Points in 2 diffrent stereo images pointing to the same point $\mathbb{X}$
l'	Epipolar line
$\mathbb{F}$	Fundamental matrix
$\mathbb{C}$	Covariance matrix
S	Surface
p	Point $p \in \mathbb{E}^3$ on Surface S
P	Point set P or Projection matrix
V_P	Voronoi diagram of point set P
D_P	Delaunay triangulation of point set P
V_p	Voronoi cell at point p
C_p	Cocone at point p
p^+,P^-	Pole vectors in Cocone algorithm
v_p	Surface normal estimate by point p and p^+
α	The angle of the Cocone C_p
ρ	The ratio to define flat points in Cocone algorithm
$S(u, v)$	NURBS surface function
$P_{i,j}$	NURBS control grid

$w_{i,j}$	NURBS weights
$N_{i,p}$	Non-rational B-spline basis functions
U, V	NURBS knot vectors
d	Knot span
$Q_{k,l}$	Data points for the NURBS surface approximation
$\mathbb{R}$	Target surface in NURBS surface approximation

1 INTRODUCTION

The surfaces we see in nature are presented differently in computers. Objects in the nature are physical and we can get infinite number of measurements. Many objects can also be presented as a virtual models in computers and they must be defined somehow. A common way is to define the surface by defining discrete points in space and interpolating a plane through them. Another way is to represent the surface with a mathematical model with a few parameters. The mathematical model can present the surface points with infinite accuracy like in nature.

3D surface modeling is increasingly important for generating surfaces from the point clouds. The points clouds can be captured from real objects or scenes, often by range scanners or by other computer vision techniques. The devices and techniques are constantly becoming more common and applications are needed to handle the point clouds. One of the most common way to describe data acquired from a 3D scene is a point cloud. Point clouds are just collections of points in 3D. In many cases, point clouds do not represent objects accurately enough. That's why interpolation of points to create 3D surface models is sometimes needed. In CAD (Computer-Aided Design) industry it is sometimes needed to model something already created. One way to model an existing object, is to acquire a point cloud from it and turn it to a 3D model. Some people may also want to archive historical objects like statues in 3D and similar methods can be applied. 3D surface modeling is also widely applied in industry and in many medical applications as the layered x-ray images can be modeled in 3D.

We created a program for creating 3D models from point clouds. This software takes care of surface modeling from points to the surface generation. There are many different approaches to create 3D surfaces from point clouds. Many of these approaches are shortly discussed and explained in this thesis. We concentrate on point clouds captured by the stereo cameras but we also take synthetized point clouds into account to measure performance of the methods used. Additionally some processing like clustering is applied to the point clouds before and after surface reconstruction to produce better results. Finally, the selected 3D surface reconstruction method creates a triangulated model which is turned into a NURBS model.

1.1 Research problem

The research problem is divided into multiple parts. The first and most important part is to find out how to visualize point clouds as 3-dimensional models. Then, one acquires points from a stereo camera progressively and must be able to construct 3D models preferably on-line. The third objective is to fit a NURBS (Nonuniform Rational B-spline) surface to the resulting triangle mesh. NURBS can describe the 3D model's surface as smooth and accurate.

The selection of the used methods is justified with various indicators including speed and performance. This thesis only deals with Delaunay-based algorithms for the surface reconstruction. For the NURBS surface fitting a combined approach is used, where 3D model is converted to a grid to be used in NURBS surface reconstruction. We use existing triangulated 3D model to create NURBS surfaces as the surface approximation is already done with the triangulated model.

1.2 Outline of the solution

In this thesis, we introduce an application that reads point clouds from a camera device or from a file. The application is able to create models and show them on the screen. The software is controlled from a UI (user interface). The software reads points from a file and constructs a 3D surface model. Before the 3D surface reconstruction, a point cloud is clustered to remove noise and some outliers. Whenever a 3D model is created, user can convert it to the NURBS surface. User may also save results as an OFF (Object File Format) file. The application is developed on Linux platform using C/C++ with CGAL (Computational Geometry Algorithms Library) for geometric algebra functions such as Delaunay triangulation. CGAL is the defacto library for geometrical algorithms.

We propose to use a COCONE-based algorithm [1] for triangulated 3D surface reconstruction and further use triangulated 3D model to create the NURBS surface model. COCONE algorithm was selected to the task because it performed relatively well and was easy to implement. It is also fast as it only needs the calculation of Delaunay triangulation and a small amount of filtering. Also the incremental nature of the Delaunay triangulation seemed as a possibility to create an incremental algorithm. COCONE can be applied to small point clouds almost in real-time, which is one of the goals of this thesis.

NURBS models are created from the triangular model by merging triangles into a quadrilateral model. Each quad is then divided into a grid structure which is estimated by clusters connected to that quad. The resulted models can furher be used as a solid visualization of the point cloud or the models can be further processed in some other modeling tool. Even though the model might be erronous, it can be fixed in the external modeling tool.

The performance of the resulted application is measured by the time for surface reconstruction and the correctness of the resulting surface. The correctness of the surface is in some situations measured by eye, when no ground truth was available.

1.3 Structure of the Thesis

The structure of the thesis is as follows: first some background to the subject is given in Chapter 2. In Chapter 3 we'll go in-depth to the algorithms used in the implementation. In that chapter, we will also consider how NURBS and other post-processing methods are applied to this kind of problem. The application itself is presented in Section 3.3. The experimental data and the acquisition methods are presented in Chapter 4. The results are presented in Chapter 5. Finally, we will have discussion and conclusions of the work itself in Chapter 6.

2 BACKGROUND

All the fundamental topics for surface reconstruction are discussed in this chapter. The thesis concentrates on geometric modeling and therefore, understanding of fundamental concepts is needed. In addition, it is good to know how the stereo cameras work and how they produce stereo images and point clouds. Finally, some details about interpolation and NURBS surfaces are discussed as they are needed in post-processing of the model. Also a great deal of the earlier work is introduced. Some of the work inspirated our solutions. Others are introduced to show advancement in this area of science.

2.1 Fundamentals of geometric modeling of point clouds

All the objects in 3D modeling are some set of points in $\mathbb{E}^3$, which is the 3-dimensional Euclidian space [2]. In 3D surface modeling, the points in point clouds are usually presented as a set of triplets (x, y, z), where $(x, y, z) \in \mathbb{E}^3$. These triplets are usually called as vertices in 3D modeling. Points may or may not be organized in some way. In this thesis we focus on the unorganized point sets, so each triplet is independent from another. A 3D surface is presented as a set of triangles $(\vec{x}_1, \vec{x}_2, \vec{x}_3)$, where $\vec{x}_1, \vec{x}_2$ and $\vec{x}_3$ are triplets. A single triangle can be called as a face. These surfaces can be connected to each other in some manner and in our case each triangle is aware of all neighboring triangles. The generalized version of a triangle is a polygon that can have any number of vertices. A polygon with four vertices is called as a quad and is used in post-processing in this thesis. Two triangles can be merged together to create a quad as one edge of the triangles is connected. Further, a tetrahedron, which is a set of four triangles and thus, creates a volume. The more generalized version of a tetrahedron is the polyhedron that can have any number of faces with any number of vertices. These concepts are visualized in Figure 1. All triangles are *convex* as their every internal angle is less than 180 degrees and every line segment between two vertices in a triangle remain inside the *convex hull* [2]. The convex hull of a set of points is the boundary of the smallest convex domain containing all the points in the set. Some polygons might be *concave* as one or more of the internal angles are larger than 180 degrees. Such polygons can be divided into a smaller convex polygons -namely triangles -and this procedure is called as a *tesselation*. These concepts are visualized in Figure 2. The tesselation can also be called as a triangulation procedure as it produces triangles [2]. For these primitives a wide variety of mathematical operations are available and all forth-coming discussion is based on these primitive concepts.

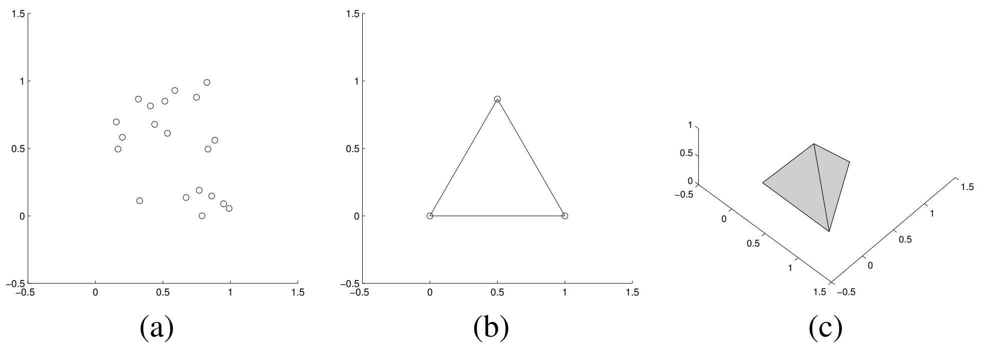


Figure 1. Fundamental geometric concepts: (a) 2D Points, (b) a triangle in 2D, and (c) a tetrahedron in 3D.

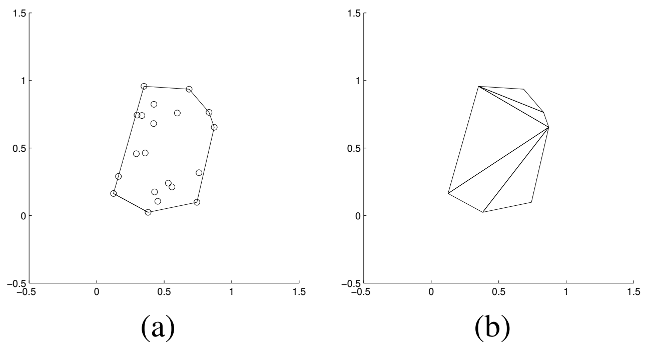


Figure 2. Some fundamental properties of points and polygons: (a) Points and their convex hull and (b) triangulation of the convex hull.

E.g. the 3D model creation algorithm used in this thesis is heavily based on Delaunay triangulation algorithm, which is described in Chapter 3.1.

2.1.1 Stereo cameras and epipolar geometry

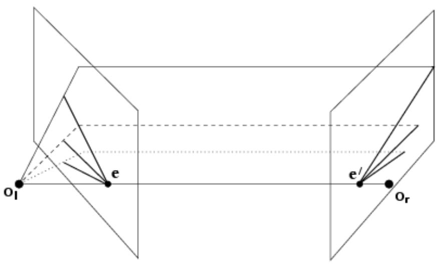
Currently, point clouds are mostly acquired by stereo cameras or by range scanners [3]. The stereo cameras are more prone to measuring errors as they need to create point clouds based on two images instead of the pure range data. However, stereo cameras are more cheaper and more common than range scanners. Often the range data is used to evaluate performance and correctness of surfaces created by reconstruction algorithms.

As this thesis concentrates mainly on stereo cameras, one needs to know how these point clouds are constructed and how much error the point clouds might contain. The stereo cameras are usually binocular, having two separate cameras from two different but known positions. So, the both cameras take a picture of a same object at the same time but from slightly different angle.

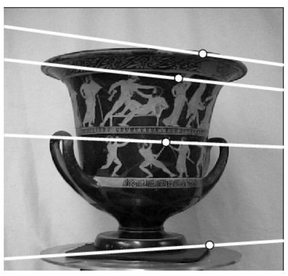
Hartley and Zisserman [4] describe how the stereo imaging is based on the epipolar geometry. The epipolar geometry itself deals with the relations between 2D images and the corresponding 3D points. With the epipolar geometry one can find the corresponding features from each image and then calculate the 3D coordinates for the points in the scene. Epipolar geometry itself is based on the camera model such as a simple pin-hole camera.

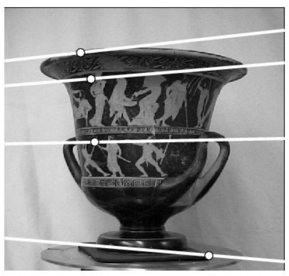
Figure 3 describes the basics of epipolar geometry. The rectangles in (a) are Epipolar planes, points e and e' are Epipoles and visible lines are Epipolar lines. The line connecting optical centres o_l and o_r is the baseline. Baseline describes a common line for both views and is the basis of finding correspondencies between images. Notice that Epipoles are not shown in visible images (b) and (c) as it's the point where Epipolar lines cross the baseline.

Rectification is the part where we define the cornrespondencies for pixels. Rectification translates Epipolar lines to the baseline making correspondent pixels appear at the same position in both images. This works as long as two images are not too different from each other. [4]



(a)





(b) (c)

Figure 3. Converging cameras. (a) Epipolar geometry for converging cameras; (b), (c) a pair of images with epipolar lines [4].

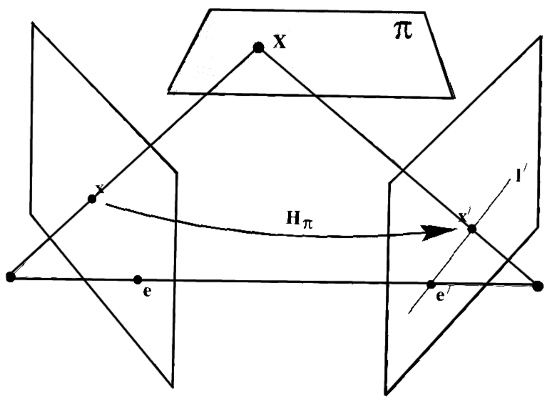


Figure 4. Constrained epipolar case. A point x in one image is transferred via the plane π to a matching point x' in the second image. Epipolar line through x' is obtained by joining x' to the epipole e' [4].

The most simple case of epipolar geometry is the case where the cameras' intrinsic and extrinsic parameters are known (a calibrated camera case) [4]. Figure 4 describes this situation. Here the points x and x' are points in both images of the 3D point X lying on a plane π. As such, there is a 2D homography H_π mapping each x_i to x'_i.

The epipolar line l' can be constructed by using the given point x' and the epipole e'. [4] This can be written as $l' = e'xx'$. Further, the fundamental matrix F can be defined. The cameras' intrinsic parameters can be encapsulated as a fundamental matrix F. This fundamental matrix is the algebraic representation of epipolar geometry and is used to map 2D points in image to the 3D points in space. Algebraically said, for each point in one image, there exists a corresponding epipolar line l'. When selecting corresponding point x' from the second image, it must lie on the epipolar line l'. This mapping $x \mapsto l'$ can be represented by the fundamental matrix F. This fundamental matrix can be determined automatically by homography estimation using RANSAC (RANdom SAmple Consensus) [5] and point correspondency or by hand [4].

Fundamental matrix can now be processed into the camera projection matrices P and P'. In addition to these two matrices we have two points x and x' in the two images. These points lie on the same epipolar line, satisfying the epipolar constraint $x'^T Fx = 0$ [4]. Using the camera matrices and the corresponding two points, we may calculate rays that lay on the plane. The point in 3D is the point where these rays coincide.

Using this approach 3D points can be formed by using a stereo pair [4]. It's also good to notice that the epipolar geometry not only apply for stereo cameras but also for monocular systems. On monocular systems we have only a single camera but we take multiple images in different locations. However, use of only one camera makes it more difficult to calculate the fundamental matrix F because the position for another frame acquired by the same camera is unknown.

For this thesis some point sets were created by an on-line GPU implementation of a stereo algorithm used for 3D reconstructions [6] inspired by Ruigang et al. [7]. The system uses EKF-based (extended Kalman filter) SLAM (Simultaneous Localization and Mapping) to detect interest points from multiple camera frames and determine the current camera position. The selected frames from the camera are chosen, left and right image are chosen and rectified. The rectified images are further processed into a dense depth map and when we know the current camera location, we can extract 3D coordinates from the scene. [8] The details of the system can be found at the project homepage [6].

2.1.2 Clustering

In this thesis, there are point clouds varying from 400 points to even 1.4 million points. Most surface reconstruction algorithms can handle less than thousand points relatively fast (on-line reconstruction), but the reconstruction of over ten thousand points can take a very long time to finish. This is why some simplifications are needed for the model. One way to do the simplifications is to utilize a some sort of clustering method [9].

With clustering one can classify a set of samples in clusters. Classifying or partitioning is decided by the attributes of the samples leading to situation where all similar samples are part of the same cluster [9]. For point clouds clustering methods can be used -such as hierarchical clustering. After clustering, each cluster centroid is used as a single 3D point for the reconstruction. In the end we have an estimation where cluster centroids represent the underlying 3D model.

2.1.3 Nearest neighbors and triangulations

To be able to connect nearby points to create a surface, one must be able to find these neighboring points. One may always calculate distances in the Euclidian space between every point and use this information to find the nearest neighbors and connect to them. This is slow, because the complexity of the calculation increases rapidly when point set increases. Fortunately, other, faster alternatives are possible as presented in this thesis [2]. Another problem is to define the underlying topology of a point cloud. Without any priori information of the object, the topology must be extracted from existing data in some manner.

The reconstruction algorithm used in this thesis is based on Delaunay triangulations and Voronoi diagrams to segment the point clouds. Delaunay triangulation is an algorithm to turn a set of points to a set of triangles [2]. In Delaunay triangulation it's defined that each circumcircle of a triangle is *empty*. Empty circumcircles do not contain points other than the three that define the triangle mentioned. Voronoi diagrams are duals of Delaunay triangulation meaning that one can always generate a Voronoi diagram from a Delaunay triangulation and vice-versa. The duality is demonstrated in Figure 5.

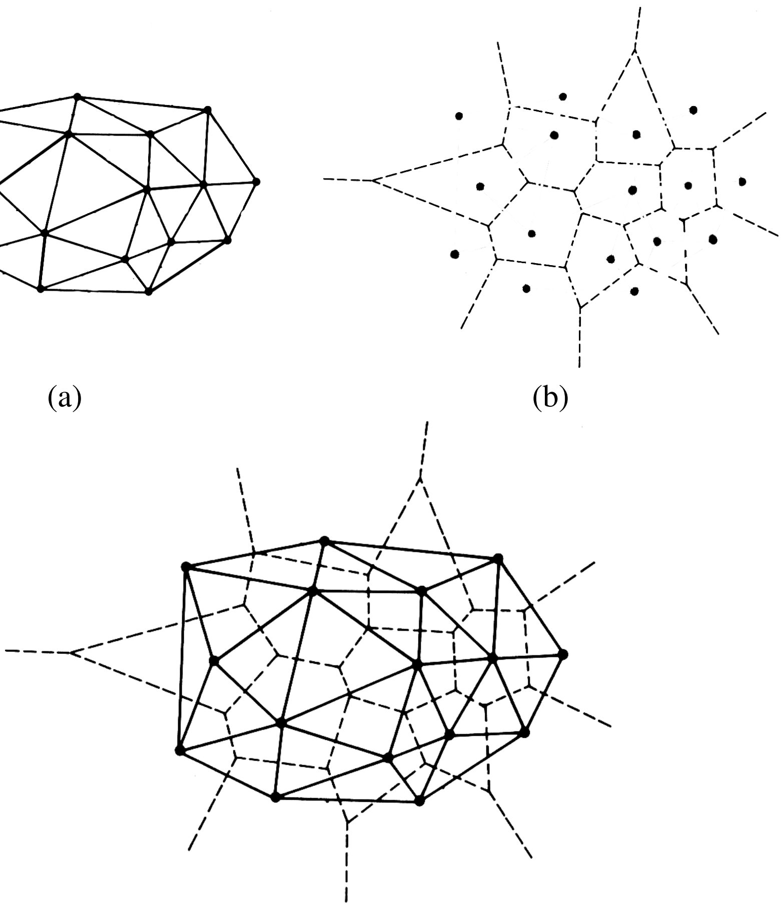


Figure 5. Duality of Delaunay triangulation: (a) Delaunay triangulation, (b) dual Voronoi diagram and (c) both in one figure [2].

Another approach for surface reconstruction is to estimate the surface S with various interpolation methods. These methods usually find the closest set of points for every point and interpolate the surface from them [10]. These approaches are briefly discussed in Chapter 2.2.

2.1.4 NURBS

Nonuniform Rational B-splines (NURBS) are mathematical models usually used in computer graphics, especially in product desing and CAD (Computer-aided Design) models [11]. NURBS can describe the surface of a model in a compact form. These surfaces are basically functions of two independent parameters u and v for a surface in 3D. For these surfaces there are control points which affect to the resulting NURBS surface. In addition to this each control point has a weight to tell how much it affects to the NURBS surface. In NURBS surface generation knots and order for both u and v must be defined. The knot vectors determines where and how the control points affect to the NURBS surface [11]. The number of the knots is equal to the number of the control points plus order plus two. The knot vector divides the NURBS surface construction into many intervals. These intervals are referred as knot spans. Every time the value in the knot vector enters a new knot span, the new control points become active and the old ones are discarded. The values in a knot vector need to be in non-decreasing order because otherwise the algorithm selects a wrong knot.

The order parameter defines the degree of the polynomial used to represent the surface. The order is one less than the degree of the polynomial. Hence, second-order curves are linear, third-order quadratic, fourth-order cubic and so on. The number of control points must be greater than or equal to the order of the curve. The construction of the basis functions using all these parameters are described in detail in NURBS book [11] as well as briefly in Chapter 3.2.

In Figure 6 one point set is presented. The points are connected to each other as shown. These data points are the target set of points into which we try to fit a NURBS surface. The surface also describes the correlation between the data points. In this example we have 12x10 data points and these data points are to be fitted with the NURBS surface as it can be seen in Figure 7. Basically we can now describe those 12x10 data points with just the 5x5 control points defined for the NURBS surface. As such, the whole NURBS surface is described by the control points, the knot vector and the order in u and v direction. Control points define the shape of the surface. NURBS also features a scalar weight for each control point. The higher the weight the more a surface point is attracted by the control point. The example in Figure 7 used uniform weights for all points to make the NURBS surface fitting algorithm a linear problem.

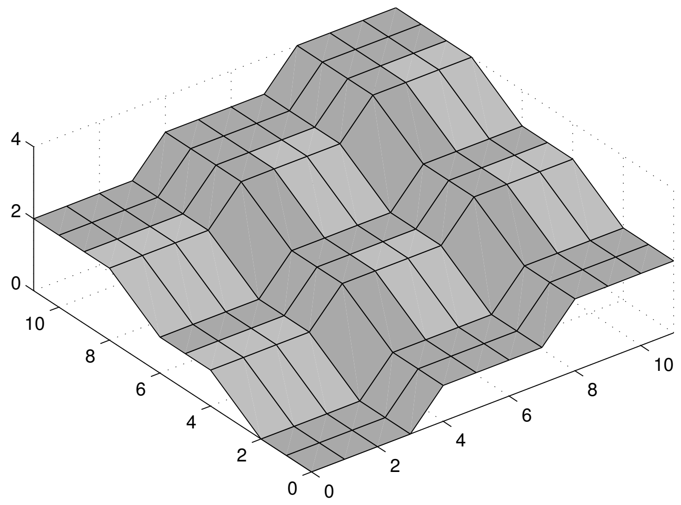


Figure 6. Point set used for NURBS surface fitting. The points set is presented as a surface to demonstrate the connectivity between the points.

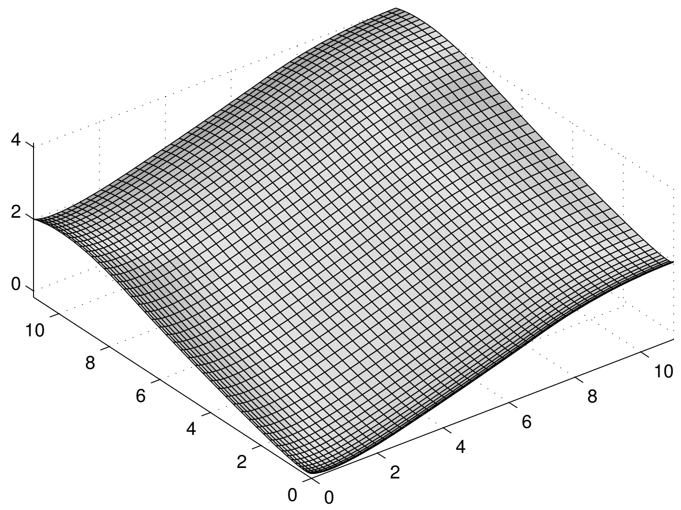


Figure 7. The fitted NURBS surface with 5x5 control point grid.

2.2 Earlier work on surface reconstruction

Most of the surface reconstruction research concentrates on creating a surface as correct as possible in a reasonable time. In general, reconstruction methods can be divided by the way how the resulting surfaces are constructed. Jean-Daniel Boissonnat [12] represents four different categories fo surface reconstruction. The first approach is inspired by differential geometry, the surface is considered as a graph of a function. This function is further approximated in some manner into a triangulated surface. Such methods might be the moving least-squares or the moving projection plane. The second main approach deals with considering the surface as an elastic membrane. At the beginning, a larger membrane encloses the point set and deformation process is applied as long as the energy is minimized to local minimum. The third approach uses some manner of combinatorial methods, which usually construct a geometric data structure, such as the Delaunay triangulation, of the point set. This data structure is further used to extract resulting triangulation of the surface. The fourth category is to define a signed distance function and to compute its zero-set. The surface is therefore regarded as a level surface of an impicit function defined over the entire embedding space.

Hoppe et al. [10] presented the first functional approach using tangent planes for surface reconstruction. The algorithm takes a set of unorganized points as an input and outputs the resulting surface. The algorithm first estimates tangent plane for each point and then use this tangent plane to define a *signed distance function* $f(p)$. A signed distance function is a function estimating the distance from point to the surface. Zero-set of signed distance functions means that all the points are located on the surface. Estimation for the tangent plane is done via k-nearest neighbor clustering. The distance to the tangent plane is considered as signed distance function. The signed distance function can then be used for contour tracking. In this case they used *marching cubes algorithm* to track the resulting surface. The algorithm used EMST (Euclidian minimum spanning tree) to describe the relations between the tangent planes. The overall complexity of the algorithm was $O(n^2) + O(n + klogn) + O(n)$, where n was the number of points in a point cloud and k was the predefined nearest neighbors for each point. As such, it is computationally expensive on large point sets.

Another similar approach was presented by Jean-Daniel Boissonnat in 2000 [12]. Instead of using tangent plane interpolation the method uses natural neighbor interpolation. Natural neighbors can be extracted from Voronoi diagram of the point cloud. For each point the natural neighbors are calculated and the signed distance function is defined from them.

The set of these distance functions are further interpolated and the triangulation is done with the dual of the used Voronoi diagram.

Amenta and Bern presented the first theoretically proved algorithm for the surface reconstruction 3-dimensional points [13]. They used algorithm called CRUST and the 2-dimensional case of the algorithm is presented in Figure 8. The CRUST algorithm used Delaunay triangulations and Voronoi diagrams to reconstruct surfaces. This algorithm takes Voronoi diagram of the point cloud and in the each Voronoi cell, the two furthest points are chosen. After this an union of the original point cloud and the set of newly selected points is created and this union is triangulated. The surface is now generated by triangles, which have the vertices from the original point cloud. The algorithm is simple but it requires a lot of calculation - first calculate Voronoi diagram and then calculate Delaunay triangulation of a three times larger set of points (original point cloud plus the selected furthest points for the each Voronoi cell). It was also noticed that this algorithm did not perform well on sharp edges and produced unwanted triangles. However, some improvements were made by Varnuska et al. to improve the correctness of resulting surface [14]. He applied different manifold extraction, better estimation for surface normals and uniform data filter.

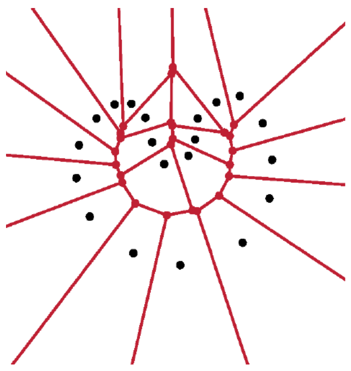


Figure 8. The two-dimensional CRUST algorithm. On the left, the Voronoi diagram of a point set S. On the right, the Delaunay triangulation of S and Voronoi vertices. The CRUST edges are presented in thicker black color. The CRUST edge is presented if at least two points of any triangle belongs to original point set S [13].

After the presentation of CRUST algorithm, many similar algorithms were created. Such algorithms as COCONE by Dey et al. [1] and Power Crust by Amenta et al. [15]. COCONE algorithm (or one-pass CRUST) is based on a single calculation of Delaunay triangulation over the whole point set and by using the dual of the triangulation (Voronoi diagram) it was possible to define triangles which reside on the surface. This algorithm proved to be a way more robust than the CRUST algoritm but still it didn't deal well with sharp edges and boundaries. Improvements made by Varnuska et al. to CRUST algorithm can be applied for COCONE algorithm as well [14].

Amenta et al. also presented improvements to the original CRUST algorithm in form of Power Crust [15]. They improved noise tolerance by introducing *the power diagrams* to estimate medial axis of the object.

There are multiple variations of COCONE algorithm to deal with different problems. Such algorithms are Tight COCONE that reconstucts a water-tight surface by peeling. It was presented in 2004 and it works with point clouds with small amounts of noise but fails with very noisy [16]. Another variation is SuperCOCONE, which works on very large point sets (over one million points) [17]. It clusters the point cloud into areas and applies COCONE algorithm on smaller groups. In the end, all smaller surface reconstructions are tied together to form a very large reconstruction.

Most of these reconstruction algorithms based on Delaunay triangulations did not deal well with noise. It is the reason why some algorithms are created to work in high noise conditions. One such algorithm is RobustCOCONE that uses Delaunay triangulation's *Delaunay balls* to decide which points are taken into reconstruction [18]. This approach is used to find out normals in AMLS (Adaptive Moving-Least-Squares) surfaces. This results of the smoothing in resulting reconstruction [19]. The basic idea can be seen in Figure 9. At the beginning we have noisy points on the surface. Delaunay triangulation is applied and delaunay balls are calculated. A Delaunay ball is a circumscribing ball of a tetrahedron in the Delaunay Triangulation at point p. In figure 9 the second image describes a big delaunay ball with circumcentre c and point p on the surface. v_p is the estimated normal by the points c and p. In the last image the surface is estimated with AMLS function and the rest of the points are projected on the surface in normal direction.

In addition to the triangular surface generation, NURBS surface reconstruction was also researched to compress data and describe it more accurately. W. Ma and J.-P. Kruth first presented a NURBS curve and surface fitting algorithm in 1998 [20]. They used a two-pass algorithm to fit NURBS curves to the point cloud. First they parameterize acquired points for fitting process. Parameterization includes a use of base surface which is approximation of underlying surface. After approximation the final parameterization is acquired by projecting remaining points onto the base surface and assosiating each measured point with its projected point [21]. Secondly they fit NURBS surface on the parameterized surface points with least-squares method. Additionally their algorithm select weights for each point on the surface.

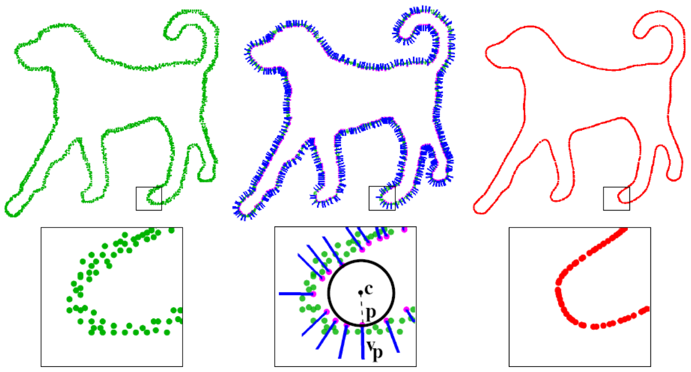


Figure 9. Normal estimation in AMLS. Outward normals are estimated from big Delaunay balls and points are projected with these normals [19].

Another approach for surface fitting was presented by In Kyu Park et al [22]. They used a clustering algorithm to get a smaller point set in association to a larger point set. By using PAT (Patch Adjacency Table) the centers of the clusters are connected, creating a triangulated surface. As In Kyu Park et al. [23] proposes, the quadrilateral domain is generated by first constructing a hierarchical structure from the triangles. The procedure can be seen in Figure 10. Polyhedrons are described as points with triangles surrounding them (cells). Then we can use this structure to combine the triangles at the lowest level, proceeding at the top. The triangles are merged together to create quads. In Kyu Park et al. [23] also mention the problem when the triangles do not fit evenly. As such, they represent three different subgraphs.

As seen in Figure 11, Type I subgraph can merge triangles together without any problems. There are even amount of triangles. However, Type II subgraph doesn't have even amount of triangles so merging with another subgraph with odd amount of triangles is needed. Type III subgraph is the most complicated situation and the extraction of it can be seen in Figure 12. The whole subgraph dimension is odd, meaning that pairwise grouping does not exist. The proposed solution for this is to find the shortest path to another Type III subgraph in hierarchical structure. To do this, subgraph is incrementally expanded as long as common nodes with another Type III subgraph is found. After this the shortest path between these two subgraphs can be defined (path nodes) and further used for grouping. In the path nodes, pairwise grouping of the triangles is performed. This causes Type III subgraphs to be converted into the Type II subgraphs.

One can now apply merging for each subgraphs and the quadrilateral domain is obtained. However, there can be one last triangle to be grouped if the total amount of triangles were odd.

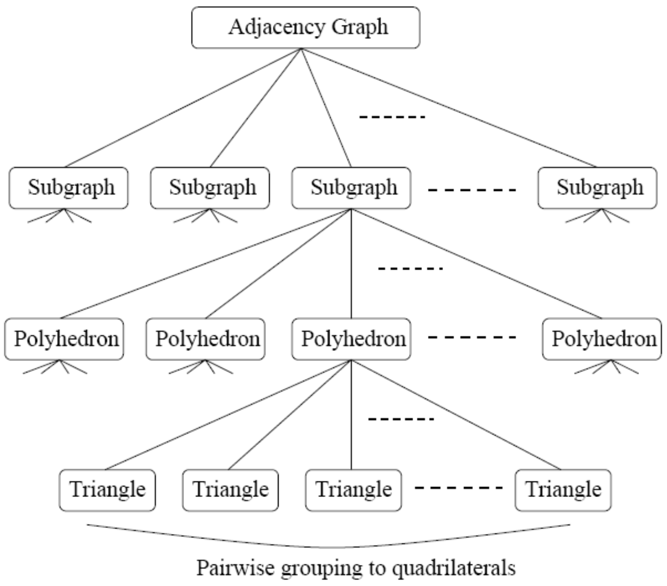


Figure 10. Hierarchical decomposition of the initial 3D model [23].

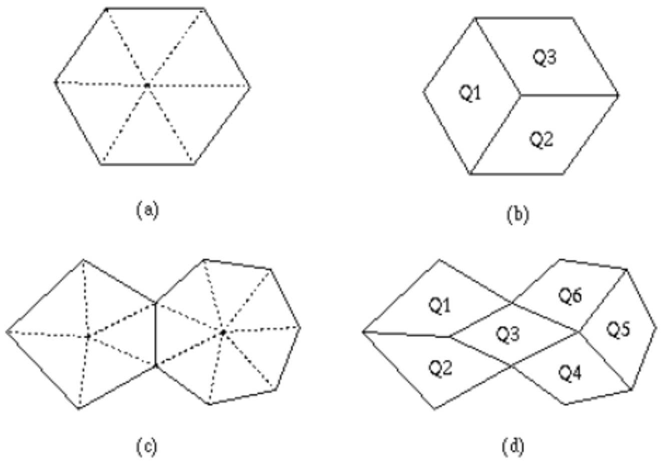


Figure 11. Generating quads with polyhedral constraint. (a) An even-sided polyhedron (Type I), (b) resulted quads, (c) odd-sided polyhedrons (Type II) and (d) resulted quads [23].

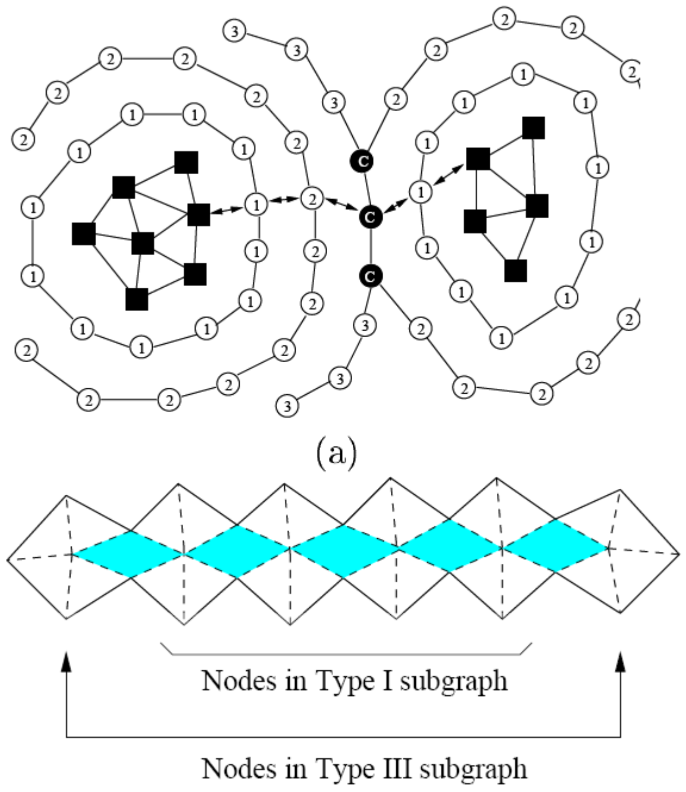


Figure 12. Processing a type III subgraph. (a) finding shortest path, number indicates the level of expansion. (b) The quadrilateral domain creation in the node path [23].

3 METHODS

In this section the selected methods are described in detail. The whole implementation is based on three phases: Pre-processing, triangulation and post-processing. In pre-processing, we simplificate acquired point clouds by clustering. This approach is described in detail in Chapter 4. Triangulation is described in next sections of this Chapter as well as post-processing, which mostly deals with NURBS surface generation from the triangulated model.

3.1 Triangulation of point clouds

As stated in the previous chapters, we concentrate on Delaunay and Voronoi based geometric solutions for 3D model creation. Earlier work within the subject shows that there are many different and well-performing alternatives. COCONE algorithm performs well on smooth point sets with only small error. It is a fast algorithm and easy to implement. As stated in Chapter 2.2, improvements to this algorithm are also useful and robust. Thanks to the pre-processing step, it is also possible to perform COCONE algoritm even for more noisy point sets while losing only some minor details.

3.1.1 The COCONE algorithm

The COCONE algorithm basically includes calculation of a Delaunay triangulation and obtaining a dual Voronoi diagram. After this the algorithm continues by checking all Voronoi cells, selecting the appropriate triangles and extracting the manifold. The extracted manifold is single so, it represents only one possible surface for a point cloud.

The COCONE algorithm can be described in a high-level pseudo-code:

```
1. Obtain point set S
2. Calculate Delaunay Triangulation
3. For each point in S:
3.1 Define Cocones (Voronoi cells with cones)
3.2 Select candidate triangles
4. Extract manifold
```

The Cocone algorithm only requires an unorganized collection of points on from the surface. This collection of points is denoted by S. For this point set, the Delaunay triangulation D_P and the Voronoi diagram V_P are computed.

Defining Cocones

Cocones are areas defined by an angle in each Voronoi cell. The areas define which dual triangles are selected for manifold extraction step. The definition of Cocone goes as follows:

```
1. Obtain Voronoi cell Vp
2. Find 2 farthest points in cell
   (Angle between them needs to be
   more than 90 degrees)
3. For each edge in Vp
3.1 If the angle between endpoints
    of an edge and the angle between two
    farthest points intersect
3.1.1 Mark it checked for that edge
3.1.2 If it has been marked by all other
      adjacent cells for that edege
3.1.2.1 Add the Voronoi edge to the
        candidate set E
3.1.3 Else Proceed to the next edge
3.2 Else proceed to the next edge
```

For each point p there is a Voronoi cell V_p. S denotes the underlying (target) surface and p is the point on that surface as shown in Figure 13. For this cell there exists "poles" which are the two most farthest points in the cell. The poles are noted as p^+ and p^-. The point p^- is the second farthest point in cell such that the two vectors from p to p^+ and p^- make an angle more than $\pi/2$. This means that the points must be located opposite directions.

The poles estimate the surface normal at point p and this pole vector is denoted as v_p. If Voronoi cell is *unbounded* (there are points at infinity), p^+ is taken at infinity, and the direction of v_p is taken as the average of all directions given by unbounded Voronoi edges.

A cocone C_p for the sample point p is defined by the normal n and a double cone with angle θ [13]. Let e be an edge in the Voronoi cell V_p and w_1, w_2 be its two endpoints. Next, we need to check if the angles between the normal and the edge endpoints w_1 and w_2 intersects the range $I = [\pi/2 - \theta, \pi/2 + \theta]$. If they do intersect, we set e as checked and if all adjacent Voronoi cells with the same edge intersect too, we add e in set E. The set E of Voronoi edges will be used to create a *restricted Delaunay triangulation*.

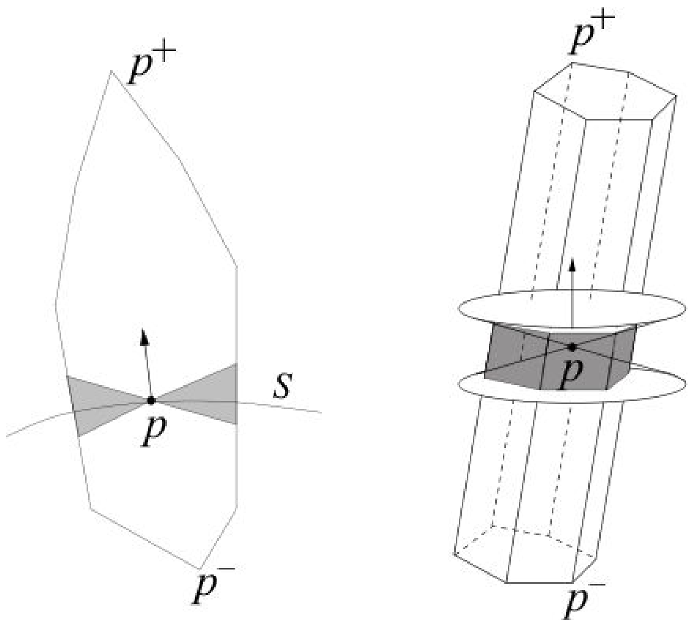


Figure 13. The cocone in 2D (left) and in 3D (right). In left cocone is shaded, in right the boundary is shaded [1].

Now one can define those Voronoi edges in V_p that intersect the obtained cocone. The dual triangles of these Voronoi edges E for each point create restricted Delaunay triangulation and the candidate set of triangles, denoted as T. This filtering is done to all Voronoi cells. After the candidate set of triangles T is obtained, we need to proceed with manifold extraction step.

Additional properties of Cocone

Additionally we can specify Cocone neighbours N_p for point p and other parameters, such as Radius and height for each Voronoi cell [24]. The radius of a Voronoi cell is defined as the radius of Cocone. Basically, the radius is the distance between the sample point and the furthest point in Voronoi cell. Height is the distance between p^- and the sample point

p.

The radius tells us how "fat" the Voronoi cell is and the height tells how "long" it is. With these properties we can define which sample points are so called "flat" points [24]. The flat points are points that cannot be on the boundary. With this the algorithm can detect boundaries and remove triangles adjacent to it. A flat point has a Voronoi cell that's "thin" and the normal is similar to the neighboring Cocones. All such triangles adjacent to non-flat sample points are discarded.

Manifold extraction

The manifold stands for points connected to each other (has an neighbouhood). In our case there are multiple manifolds in candidate set of triangles T. What we want is single-manifold: Only on possible way to connect each triangle together.

The candidate set of triangles T now has triangles which are somewhat parallel to surface S. As such, there are some triangles which are not necessary for the reconstructed model because they all represent different manifold. In manifold extraction we decide which triangles we do want and which we don not want. Dey et al. [24] represents a way of pruning and walking the candidate set of triangles. The pruning step removes triangles incident to sharp edges. An edge is sharp if there are two consecutive triangles incident to the edge such that the angle between them is more than $3\pi/2$. The edges with only one triangle are also sharp.

The walking step extracts the manifold itself. Basically it takes an arbitrary oriented triangle from the candidate set of triangles. This triangle denotes the beginning of the surface manifold extraction. The algorithm takes the edges of this triangle and adds them to a Pending set which denotes edges to be processed. Basically we proceed incrementally to neighboring triangles via edges until the manifold is extracted. The details can be found in [24].

Notes and improvements to the COCONE algorithm

The original idea was to have this software to perform on-line reconstructions. However, the nature of real-life point clouds is somewhat noisy and requires pre-processing, this might be impossible. Fortunately, Cocone algorithm is based on Delaunay triangulation, which is incremental algorithm and thus very appropriate for this case.

In addition to the original Cocone algorithm, some improvements can be applied. Such improvements are simplified manifold extraction and average normal presented by Michal Varnuska et al. [14]. In Figure 14, averaging normals is a way to improve the resulting surface. Cocone algorithm assumes that all Voronoi cells are thin. In that case the poles estimate normals pretty well as it is seen in case (a). The pole p^+ is almost parallel to surface normal. However, so called fat Voronoi cells do not estimate the normal so well because the farthest Voronoi vertices are more widely scattered as in case (b). The pole p^+ is not so parallel to surface normal.

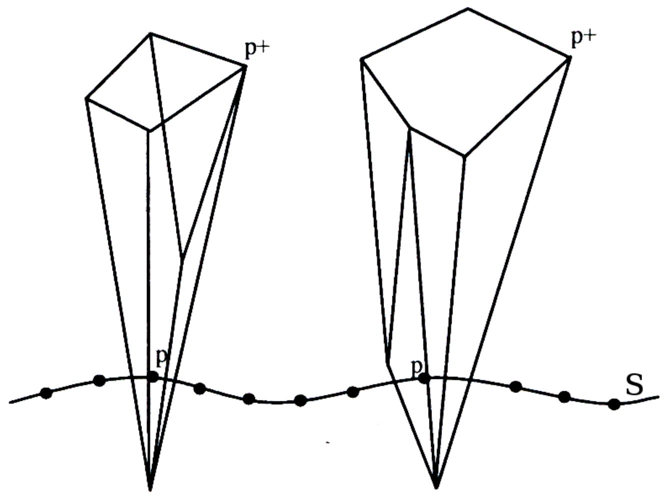


Figure 14. An image of two different Voronoi cells. (a) represents a good and (b) bad Voronoi cell when approximating surface normal [14]. Surface S is presented in 2D for simplification.

To solve the problem of bad Voronoi cells an improvement can be applied. The farthest pole p^+ is taken as a normal for the temporary plane [14]. With this plane a halfspace is defined. All the vectors from point p to each Voronoi vertex are summed as long as they lay in the same halfspace as the positive pole p^+. In Figure 14 it means that all the vertices at the top of the Voronoi cell (like p^+) would be summed together and the ones below the point p would be ignored (outside of the halfspace). This summed vector is the estimated normal for the surface at point p.

3.2 Triangulated model post-processing

We decided not only to create a reconstruction, but also do some post-processing. The major post-processing step would be the NURBS surface reconstruction. Other post-processing steps include applying normal mapping and saving into a commonly used model file by 3D-modeling programs.

As stated before, NURBS surfaces are commonly used for surface modeling in CAD because they can represent smooth surfaces with a few control points. In post-processing step, we try to generate NURBS surfaces from the triangulated model instead of creating it directly from the point cloud as Hoppe et al. [20]. We follow a path similar taken by In Kyu Park et al. [23].

The conversion from a triangulated model to the NURBS surface model requires multiple steps. First, the triangulated model must be parameterizated. After this the NURBS surface reconstruction algorithm can be applied to the parameterized point clouds. At the end we want to export the figure as a common file format for further processing.

3.2.1 Parameterization of a triangulated model

Before the points in a point cloud can be used for NURBS surface reconstruction, it must satisfy requirements stated by NURBS surfaces. First of all, NURBS requires a regular grid of points. This regular grid can be used directly as the control points or as surface points. The surface points need to be approximated in some manner to the grid structure. In Chapter 4 we acquire clusters and this means we have many surface points associated to the each point in triangulated model. We use these cluster points to approximate the grid structure.

From the triangulation, we already have a triangulated surface which represents the topology of the surface. It would be best to convert the existing triangles into the quadrilateral domain and use the data points from each quad as control points for nurbs surface approximation. Kyu In Park et al. [23] proposed a quadrilateral domain extraction method which was briefly described in Chapter 2.2. We will not use their method but do the quadrilateral domain creation a more direct way. Our approach uses breadth-first search to identify the triangles to be merged together. The algorithm can bescribed as follows:

```
1.   Take the first triangle and add it to Pending queue
2.   Do while Pending queue is not empty
2.1    Take the first triangle from Pending queue
2.2    Mark it checked
2.3    For each edge of that triangle
2.3.1    Take a triangle adjacent to that edge
2.3.2    If it is first and marked as not-checked
2.3.2.1     Mark it checked
2.3.2.2     For each edge of that triangle
2.3.2.2.1      If it is marked as not-checked
2.3.2.2.1.1       Mark it checked
2.3.2.2.1.2       Add it to Pending queue
2.3.2.3     Create a quad
2.3.3    Else add the triangle to Pending queue
```

The algorithm is also presented visually in Figure 15. Basically we have Pending set which includes all the triangles needed to be processed and merged to neighboring triangles. Those triangles are presented with numbers in Figure 15. We begin a randomly selected triangle and start extracting quadrilateral domain from there. It'll also need a check list to chech which triangles are already processed. Basically, a triangle is checked when it's merged with another triangle. At that point all its neighboring triangles are added to Pending set as shown in Figure 15. This algorithm linearly processes neighboring triangles and can't make jumps so the model needs to be solid. If the model has holes or boundaries it is possible that a neighboring triangle cannot be found to merge with.

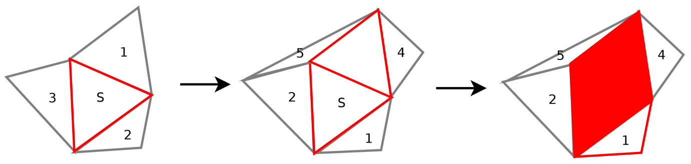


Figure 15. Quad creation process.

The difference between this algorithm and the one presented by Park et al. is that this algorithm does not use hierarchical structure to locate each triangle in higher level diagram. It is a lot faster to not have one but it also has downsides. This algorithm requires that the triangular model is a single manifold. If the model has multiple separate surfaces it will process only one of them randomly. Also this method is known to leave single triangles at boundary areas or areas where there are anomalies in triangulation.

3.2.2 NURBS surface reconstruction

As already stated in chapter 2, NURBS can represent smooth surfaces with few control points and other parameters [11]. A NURBS surface is described with the parameters in the u direction and in the v direction. U and V are the knot vectors of the surface S and p and q are the degrees of basis functions in u and v direction. The mathematical definition of NURBS surface is given in Equation 1.

$$S(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}P_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}} \quad 0 \leq u,v \leq 1 \tag{1}$$

The $P_{i,j}$ form a bidirectional control grid. $w_{i,j}$ are the weights and $N_{i,p}$ are the non-rational B-spline basis functions defined on the knot vectors U and V [11]. If we know the data points on the surface, we can define control points by using the least-squares surface approximation method. NURBS surfaces can only be approximated when the data point values are in reqular grid order. This means that the points can be anywhere as long as they're topologically in order.

The approximation method uses the least-squares minimization to fit the control point. The least-squares tries to minimize the error between the resulting surface and the original data points. In this thesis, to obtain a linear solution for the problem, all the weights $w_{i,j}$ are set to 1. The least-squares approximation needs the number of control points and the number of knots used for fitting. Knots can be defined by a simple algorithm.

Basically, in 2-dimensional case, we need a total of $n+p+2$ knots, where n is the number of control points and p is the degree of the curve. So, there are $n-p$ internal knots and $n-p+1$ internal knot spans denoted as d. For the 3-dimensional case we need to define knots both in both u and v direction.

At first we define the knot span d with the following equation [11]:

$$d = \frac{m+1}{n-p+1} \tag{2}$$

where m is the amount of original points on the surface and n is the number of control points. Now, we can define knots by

$$i = int(jd) \quad \alpha = jd - i \tag{3}$$

$$u_{p+j} = (1-\alpha)\overline{u}_{i-1} + \alpha\overline{u}_i \quad j = 1, ..., n-p \tag{4}$$

where j describes the number of intervals needed. The points on the surface are denoted as $Q_{0,0}, ..., Q_{k,l}$ and are used for surface fitting. The border points are not approximated. Surface S defined by datapoints $\mathbf{Q}$ is approximated in the least-squares sense:

$$min \sum_{m=1}^{M} |Q_{u_m,v_m} - S(u_m, v_m)|^2 \tag{5}$$

where we try to minimize the error between data points and resulting surface. $S(u, v)$ is defined by Equation 1 [11]. Equation 5 can also be written in matrix form defined by equation 6. The matrix $\mathbb{R}$ contains all polymonials to be minimized. The solving process is now presented in detail. To find out the positions of required control points P, the following equation can be used,

$$(N^T N)\mathbf{P} = \mathbf{R} \tag{6}$$

where N is the $(m-1) \times (n-1)$ matrix of scalars. The scalars are the values of basis functions over $(m-1) \times (n-1)$. $\mathbf{R}$ is the target surface defined with equations

$$\mathbf{R} = \begin{bmatrix} N_{1,p}(\overline{u}_1)\mathbf{R}_1 + \cdots + N_{1,p}(\overline{u}_{m-1})\mathbf{R}_{m-1} \\ N_{n-1,p}(\overline{u}_1)\mathbf{R}_1 + \cdots + N_{n-1,p}(\overline{u}_{m-1})\mathbf{R}_{m-1} \end{bmatrix} \tag{7}$$

where

$$\mathbf{R}_k = \mathbf{Q}_k - N_{0,p}(\overline{u}_k)\mathbf{Q}_0 - N_{n,p}(\overline{u}_k)\mathbf{Q}_m \quad k = 1, ..., m-1 \tag{8}$$

Equation 6 can now be solved and we have control points $\mathbf{P}$. Notice that approximation can be done on one direction at a time, because u and v directions might have a different amount of control points and a different degree. The resulting surface does not go precisely through all the original surface points. This is the basis of solving NURBS surfaces on predefined points and is used for the surface reconstruction for each surface patch.

The grid structure for NURBS surface approximation

It's not possible to fit NURBS surface to the whole point cloud at once because of the requirements of NURBS surface approximation. At least this is not possible without somekind of parameterization algorithm. Instead, we will use the existing triangulated surface and turn it to a regular grid structure. Each point in the original 3D model is a center of one cluster created in the pre-processing phase. For each quad we have four cluster centers and each cluster has multiple data points. These data points are used to approximate the regular grid structure, which is constructed for each quad. This creates an approximation of the surface in grid form to be approximated by the NURBS surface fitting.

In Kyu Park et al. [23] proposed a method to create this grid structure. Figure 16 represents the construction process. The dots in the pictures are the associated cluster points and the quad is the one of the generated quadrilateral domain. The quad is constantly divided into a smaller grid until the desired accuracy is acquired [23]. The middle points on quad boundary are computed by averaging the neighboring grid points. In similar manner, the surface normal is acquired. The location of the point on the surface is approximated by projecting the data points on the surface (the current quad) in normal direction. The projected points are then considered as the new grid points. The neighboring data points are also used to approximate the new normal vector. By recursively applying this procedure, we have a regular grid structure that follows the topology of underlying data points. However, this requires enough points to be projected as grid points which is not always the case. There still remains the problem with the continuity on the boundaries of the NURBS surfaces created. The boundaries needs to be modified by setting the control points on the boundaries such as the tangent control points line up in a fixed ratio over the whole boundary. The areas of high curvature need more averaging of adjacent points. Finally, NURBS a smooth surface network is generated.

Inspired by this idea we will create a grid which is divided requrarily into grid points.

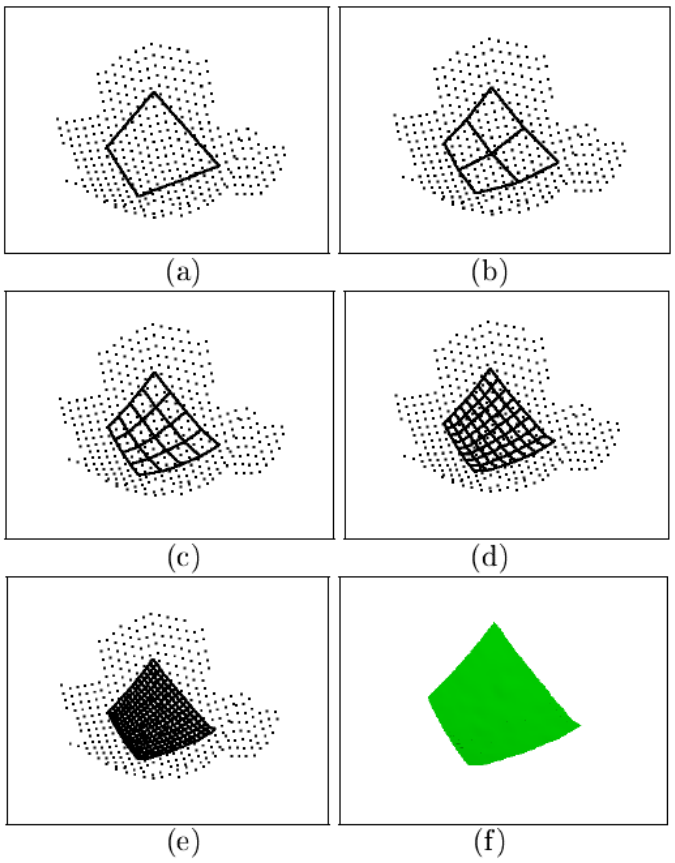


Figure 16. Constructing a regular grid from a quad associated with clusters. The size is the amount of intersections in a grid. (a) initial position, (b) 3x3 grid, (c) 5x5 grid, (d) 9x9 grid, (e) 17x17 grid and (f) the reconstructed NURBS surface [23].

Next we evaluate each grid point with the n-nearest neighbouring points from clusters. All those grid points that are not on the border of the grid are evaluated by all points in all four clusters. In Figure 17 there are four corner points in grid, each of them is the centroid of a cluster ($C1$, $C2$, $C3$ and $C4$). For all those points which do not lie on the border of the surface are approximated by taking the union of all four clusters, finding n-nearest neighbours for a grid point and averaging them. This also deals well with the outliers because we don't use all possible data points in clusters. Even if we do use all data points in clusters and have outliers, the averaging reduces the error because the grid point is in right position.

The border grid points are evaluated with a smaller union of clusters. Border points are evaluated only with clusters assigned for that border. In Figure 17 the top-most grid points are evaluated by clusters $C1$ and $C3$. This fastens the n-nearest neigbour search and additionally the continuity problem is solved. Because neighboring triangles use the same method with the same datapoints we'll have continuous borders between two surfaces without any extra checking.

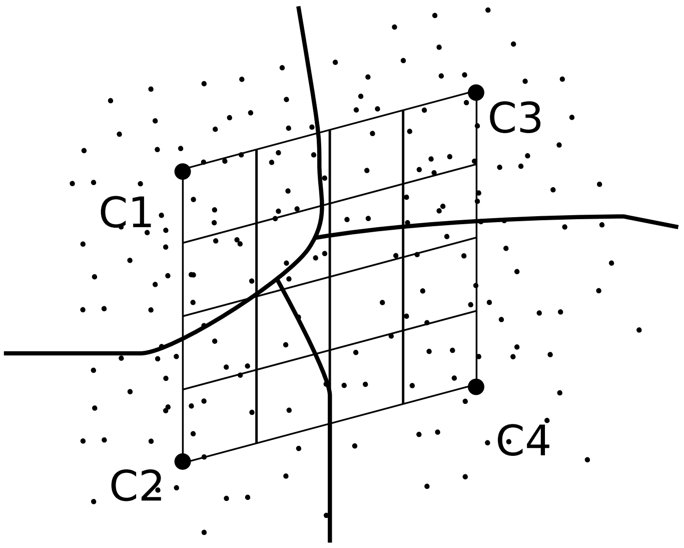


Figure 17. Grid with cluster centroids and clusters.

In the end of gridifying process, we have a data structure that can be converted into NURBS surface by using the least-squares NURBS surface fitting algorithm which was described earlier.

3.2.3 Storing data for further processing

As stated earlier, some manner of further storing method must be applied for the triangulated 3D model and for the NURBS models. The triangulated 3D model is saved into a OFF file format which is provided by the CGAL library. It was also decided to use the file formats provided by Nurbs++ library [25]. Nurbs++ library can save files in VRML (Virtual Reality Modeling Language) and POV-ray (Persistence of Vision Raytracer) format. We save files in VRML format because of the wider scale of programs which can read this file format.

3.3 Implementation details

In this chapter the implementation of the application is described. The application is using previously mentioned methods: reading point clouds from files, selecting desired points with a bounding box, applying incremental clustering and triangulating clusters with the COCONE algorithm. The application itself is implemented using CGAL library [26] for geometric calculations. The application uses Qt4 library [27] for user interface and OpenGL for 3D presentation of point clouds. The whole application is programmed with C++. The basic layout of the software is shown in Figure 18.

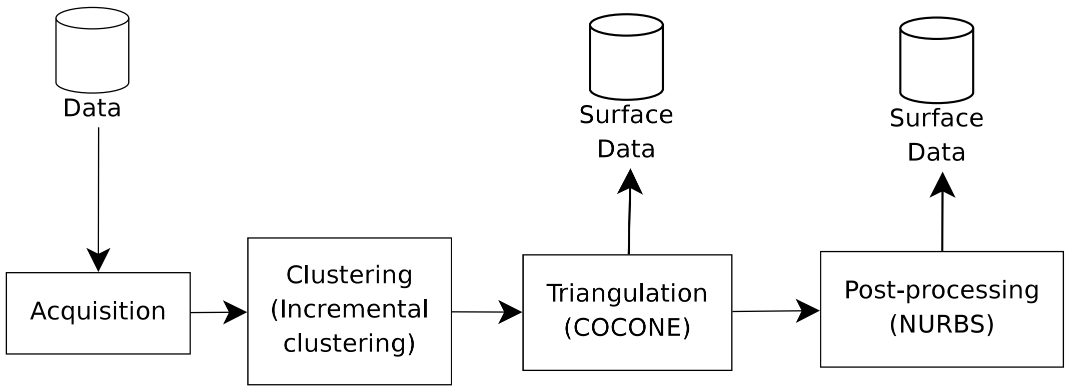


Figure 18. Different phases of the software.

At first we start with the acquisition window, which also presents the currently acquired points in Data storage. Next, we open clustering window. The clustering window is followed by the triangulation window. The triangulation eventually passes the resulting triangulated 3D model to the post-processing, i.e. NURBS surface approximation. The UML (Unified Modeling Language) diagram of the software is presented in Appendix B.

3.3.1 Gui components

In this section the GUI components of the software are briefly described. The software consists of 4 main views: Main window, clustering window, triangulation window and NURBS window.

Main Window

The application starts with a point cloud presentation window with an editable bounding box (Figure 19). Yellow cubes and red wires represents the bounding box. In this window the user can load multiple point clouds and select the desired points with the bounding box. Bounding box is modified by moving the yellow corner boxes with a mouse.

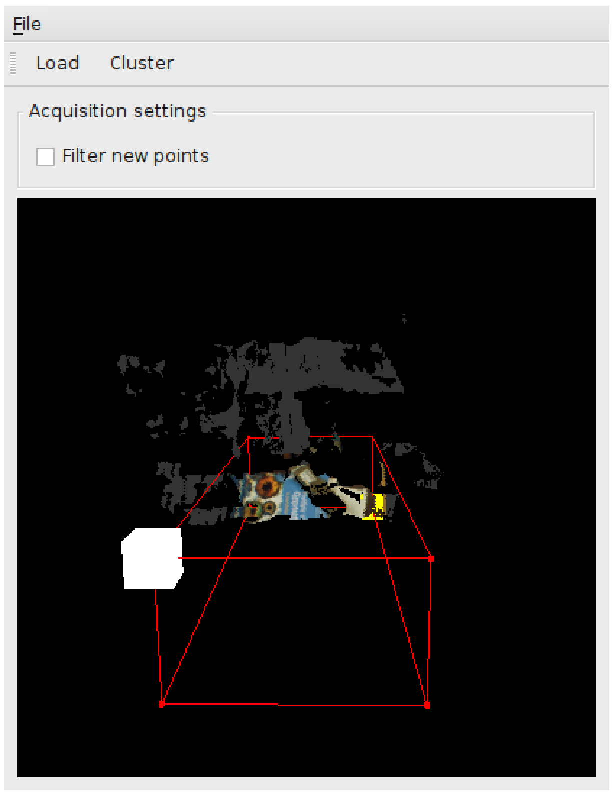


Figure 19. The main GUI.

If the user has loaded a point cloud and loads another one, the bounding box still apply and points outside are ignored. The selected point cloud can be further converted into a clustered version by selecting the "Cluster" option. Points are rendered as simple colored glpoints in space in perspective view.

Clustering Window

In the clustering window, the user has a couple of options to choose. He can affect to the maximum bound of incremental clustering algorithm. A bigger bound means less clusters and a smaller bound mean more clusters. The user may start clustering by pressing the "Cluster" button. The result will be show in image window as it can be seen in Figure 20. When the user is satisfied to the clustering result he may choose to make a triangulated 3D model.

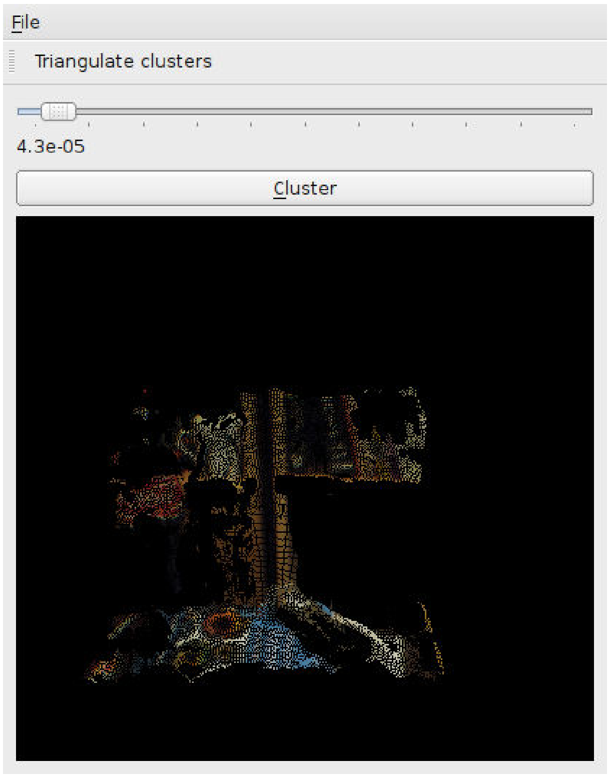


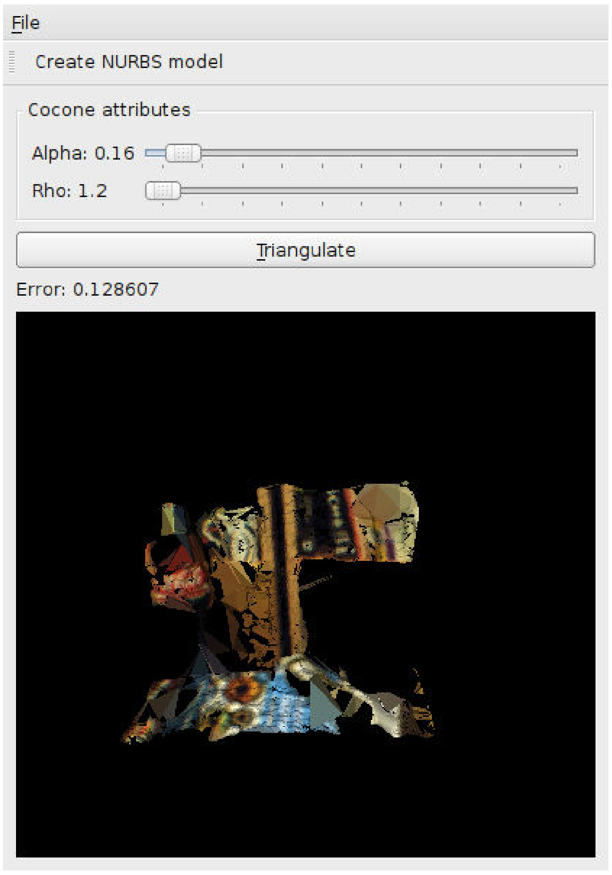
Figure 20. The clustering GUI.

Triangulation Window

The triangulation window shows all possible options possible for the triangulation algorithm. He chooses a desired theta value, which affects to the COCONE cone size. The triangulation result can be viewed in image window below. In Figure 21 the user has selected appropriate alpha and rho value and the result of the COCONE algoritm can be seen below. Alpha value affected to the size of a cocones in the COCONE algorithm. The Rho value was used as the ratio when selecting which points in a triangulation are flat and belong to the surface. The user may save the triangulated model in OFF format or further process it into a NURBS model.

Post-processing Window (NURBS)

The NURBS model creation GUI provides the parameters for NURBS surface reconstruction. The user may specify the grid size, NURBS degrees and number of desired control points. The "Process" button starts the NURBS surface reconstruction. When the



Alpha: 0.16

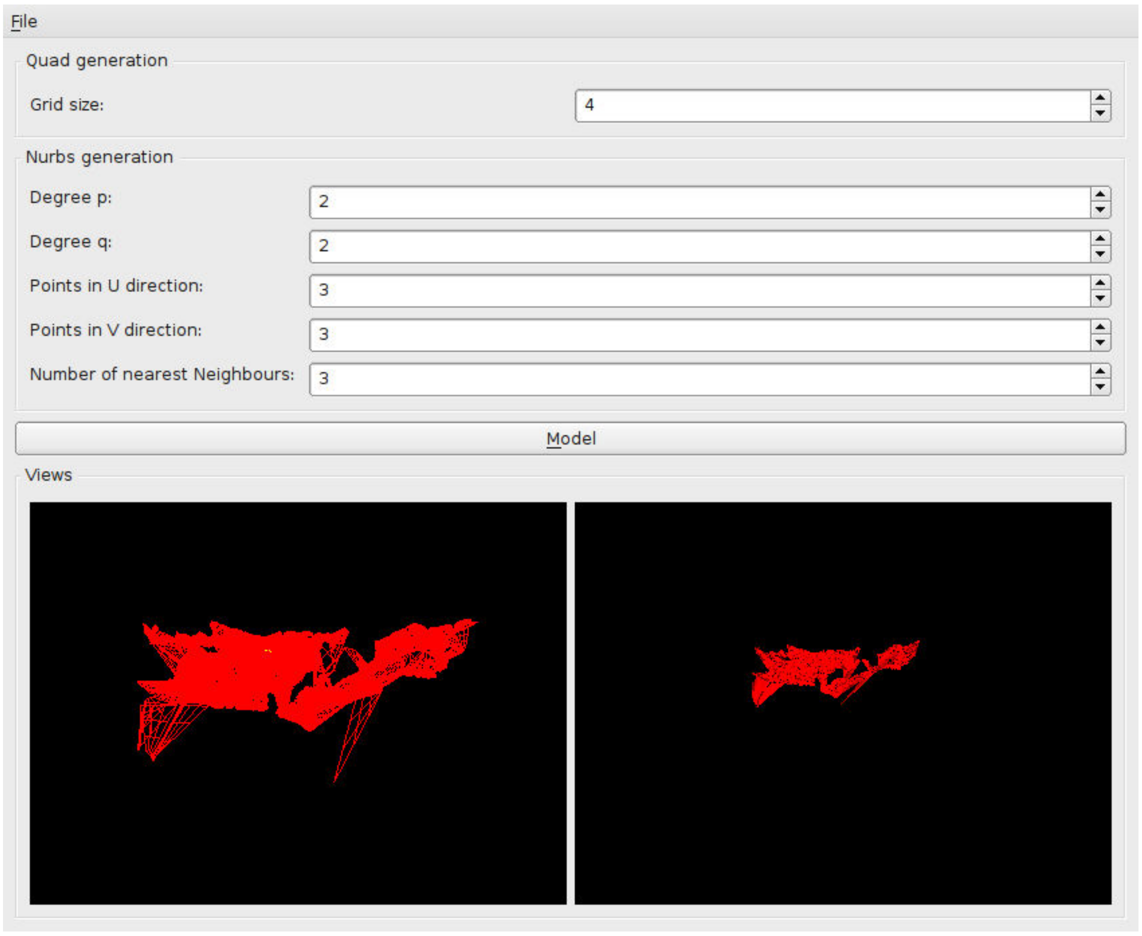
Rho: 1.2

Error: 0.128607

Figure 21. The triangulation GUI.

reconstruction is ready the user may save the result.

The software also provides the means for evaluating the resulting surface. The error of a triangulated model is presented in triangulation GUI. The error estimates the distance from all data points to the triangulated surface. Basically, this means that if you have done a lot of clustering, the resulting surface will most likely to have more error. The point clouds with points in wider area will have a lot bigger error measure. The only reasonable way to evaluate this error measure is to compare it to other measurements.


File

Quad generation

Grid size: 4

Nurbs generation

Degree p: 2

Degree q: 2

Points in U direction: 3

Points in V direction: 3

Number of nearest Neighbours: 3

Figure 22. The NURBS window of the program.

4 EXPERIMENTAL DATA ACQUISITION AND PRE-PROCESSING

The real-life point clouds might not be as good as we hope. Many acquired point clouds may have outliers or there are errors in 3D points (noise). This kind of errors are hard to deal since we really do not know if the underlying surface really contains those points. In addition to real-life data, one can have accurate point clouds done by the laser scanners or generated from a mathematic model. These point clouds rarely have significant errors unless intended.

4.1 Acquiring point clouds

In this section our acquisition methods are described. We have samples from a stereo camera system, a single camera system and some samples from a laser scanner systems provided by Stanford University Computer Graphics Laboratory [28].

The Bumblebee2 stereo camera is used for the real-life measurements. Bumblebee2 is a stereo camera created by Point Gray Research Inc [29]. Camera's specification can be found in Appendix A. Triclops software was used to acquire the point clouds [30]. Triclops software uses an area based correlation with SAD (Sum of Absolute Differences). Also the algorithm is said to be fairly robust and it has multiple steps to reduce noise including sub-pixel stereo vision [30]. The camera is said to be calibrated with RMS error of about a twentieth of a pixel. The point cloud generation method requires some texture. Occlusions, repetitive features and specularities can cause problems, but during the acquisition no such anomalies were found.

4.2 Manual refinement of the point cloud

Figure 4.2 represents the acquisition process in our final application. The application starts with the *visualization of the points*. *Point Acquisition* is started by the user when a appropriate source for the point cloud acquisition is defined. Points are acquired from a file or from a device and added to the visualization part of the program. Further, the user can make a *selection for triangulation* and decide only to get points from the visualization within the selection. The selection can be used to filter out the unwanted (the points outside the selection) points in the acquisition phase and save us a lot of calculation time. Whenever user decides he can further send points to the next phase, namely clustering and triangulation.

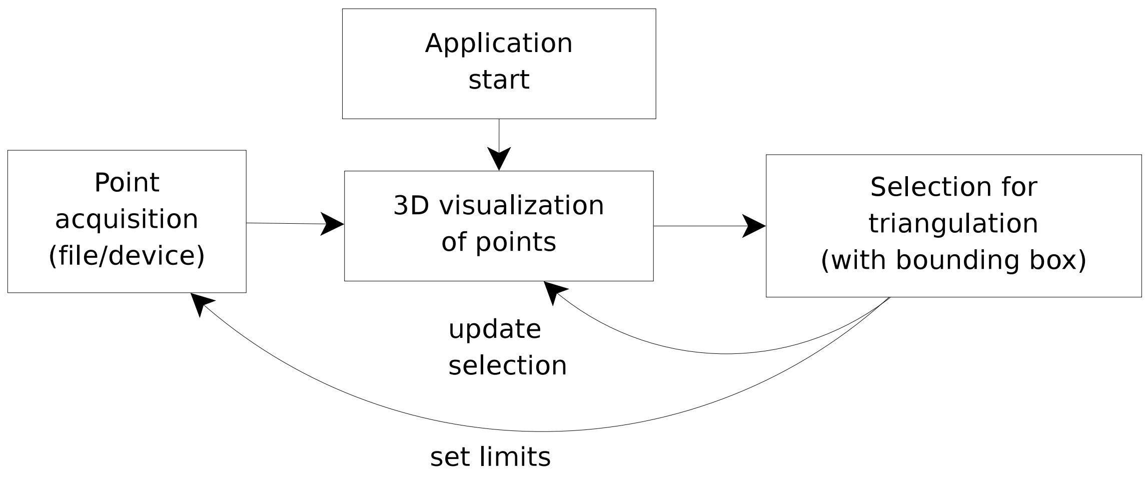

Application start
Point acquisition (file/device)
3D visualization of points
Selection for triangulation (with bounding box)
update selection
set limits

Figure 23. Diagram of point acquition process

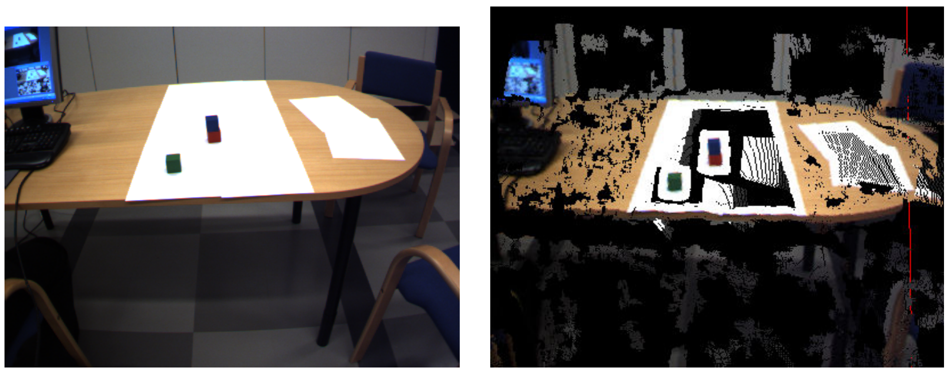
In Table 2 the sizes of used point clouds are presented. We have point clouds Stereo2, Stereo3, Desk, Fountain and Standford Bunny. Point clouds Stereo2, Stereo3 are the point clouds acquired by Bumblebee2 stereo camera. Fountain is acquired by a set of images from Strecha's dense multi-view stereo test images, using a single camera system built on GPU [6]. Desk is also generated by the same single camera system. Standford Bunny is a widely used test subject to test the reconstruction of different methods as it can be seen in multiple research papers [13, 15, 31, 32]. Stanford bunny and Fountain are the only point sets for which we have the correct reconstructions, other point clouds must be visually evaluated.

We need point clouds of different sizes because the used algorithms might work differently on them. Smaller sets are generated by taking 3D points from the big point clouds randomly. We continue picking until the desired amount of points is acquired. Images and corresponding point clouds are presented in Figures 24, 25, 26, 27 and 28. Noticeable is the difference between the point clouds acquired by the Triclops software and the ones acquired by other means. Stereo2 and Stereo3 introduce additional points (the cone shape). The cone shape was created because of the depth estimation errors in areas with only single color. The desired object for reconstruction can be found in the apex of the cone. This object must be selected and other points discarded. The same approach is required for Desk and Fountain because the whole scene might be difficult to convert into a 3D model. The selected areas are represented in Chapter 5.

Standford Bunny is the most accurate point cloud without any noticeable noise. Other stereo images, Fountain and Desk have some unestimated error in point coordinates. Points are also non-uniformly scattered on the surface. Acquired point clouds also have outliers that need to be filtered out in some manner.

Table 2. Point clouds used in our surface reconstruction experimens.

Name	*number of points*
Stereo2	95 048
Stereo3	95 048
Fountain	169 090
Desk	12 024
Standford Bunny	8065


(a) (b)
Figure 24. Stereo2 point cloud. (a) Original scene and (b) Point cloud

The point clouds are acquired from files created by softwares controlling the cameras. Each point cloud file is in text format where each line gives point coordinate (x, y, z) and point color (r, g, b), where $r, g, b \in [0, 255]$. This color information is used in further 3D reconstruction. Standford Bunny was originally in PLY format but was converted into a text file with uniform color white $(255, 255, 255)$.

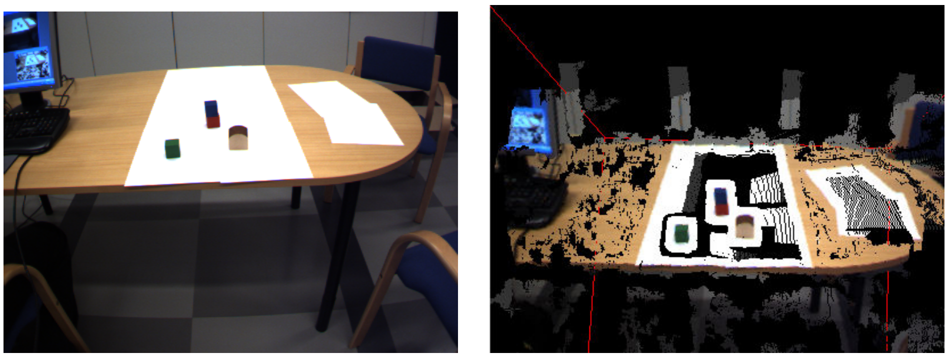

(a) (b)

Figure 25. Stereo3 point cloud. (a) Original scene and (b) Point cloud

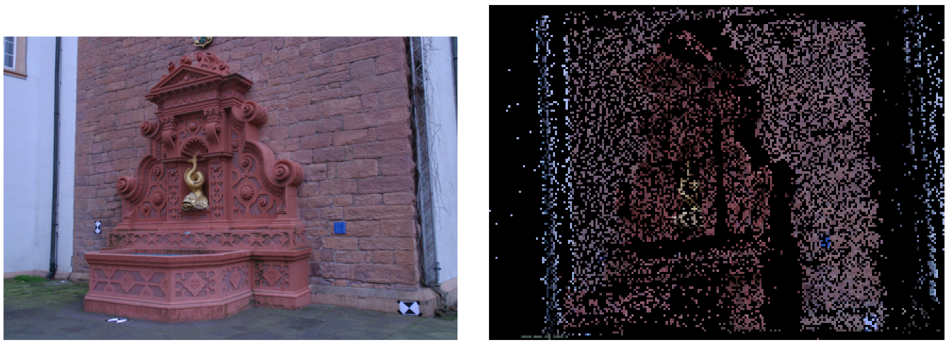

(a) (b)

Figure 26. Fountain point cloud. (a) Original scene and (b) Point cloud

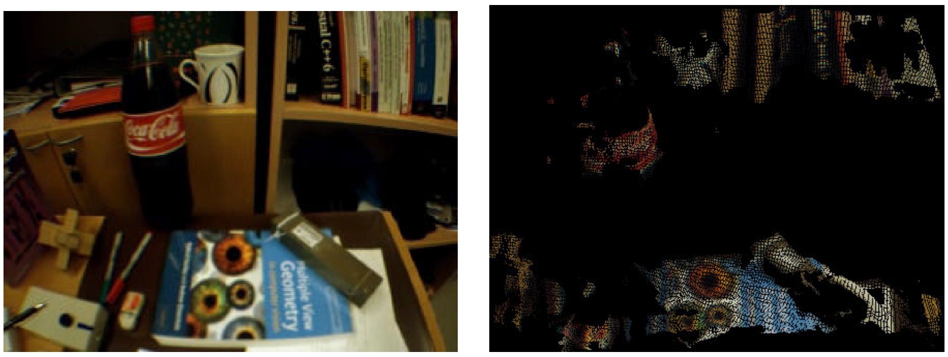

(a) (b)

Figure 27. Desk point cloud. (a) Original scene and (b) Point cloud

4.3 Data pre-processing

To handle noisy point clouds with some outliers, one must use some kind of algorithm to smoothen point cloud or remove anomalies by hand. Some outliers can be removed by algorithms but they may affect to the resulting model in a negative way. They may also remove some useful points, which are falsely identified as outliers. But on the other hand,

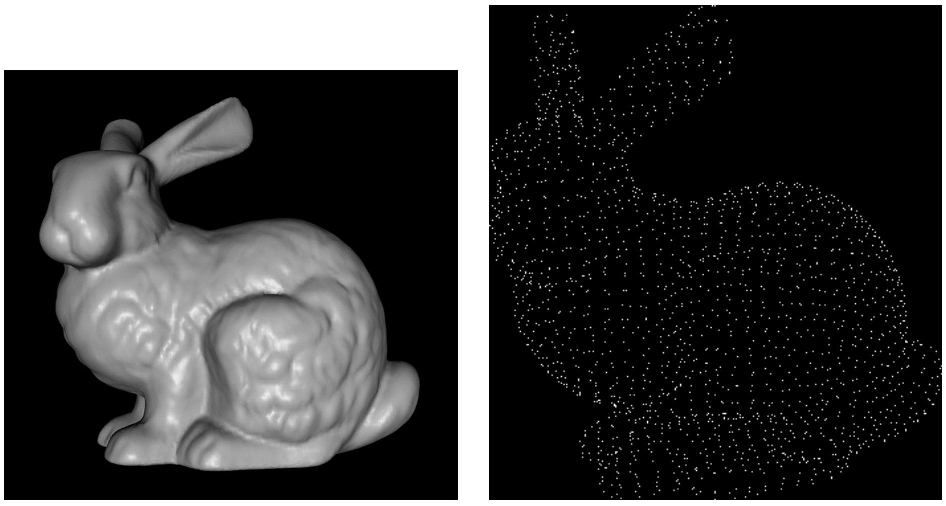

(a) (b)

Figure 28. Standford Bunny point cloud. (a) Original model and (b) Point cloud

to select all the desired points by hand would be very slow. In this thesis it was decided to use something in between.

The procedure goes as follows: first define the most undistorted and valid points as possible with a bounding box. The boundaries of a point set is fast to define with a bounding box because checking is linear. This, however, is not enough because the selected reconstruction algorithm does not deal well with points close to each other. The points near to each other usually create steep edges that are difficult to model by the triangulation method used in this thesis. To make a point cloud more simple, it was decided to apply a clustering algorithm. With the clustering algorithm one will acquire a point cloud that has well scattered points and describes the underlying model well enough.

As stated by Mark Pauly et al. [9], iterative and hierarchical clustering methods are the most efficient clustering methods for model simplification. These methods are presented in Figure 29. In iterative clustering, the clustering is started by selecting a random seed point p_0 and a cluster C_0 is built. After this, new points are added to the cluster only if they are closer than a maximum threshold (also called as a bound). Otherwise a new cluster is created. This results in uniformly distributed clusters. This approach can be extended by applying variational restriction for clusters [9]. This results in a curvature-adaptive method, where more smaller clusters are created in regions of high-curvature.

Hierarchical clustering splits a point set into a binary tree where each leaf of the binary tree is a resulting cluster [9]. The point set is split always if size of the point set is

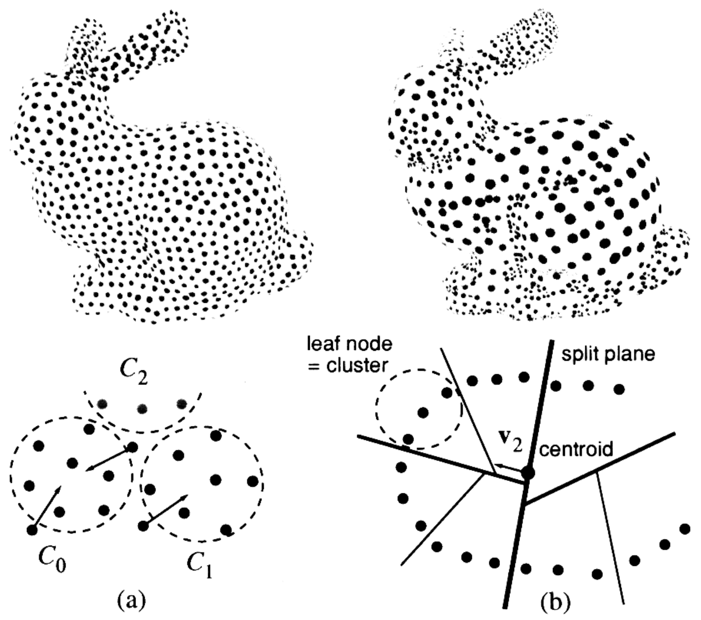


Figure 29. The clustering results. The bunny on the left is the result of incremental clustering and the bunny on the right is a result from adaptive hierarchical clustering [9].

larger than the user specified maximum cluster size or variation is larger than the user specified maximum threshold. The point set is split with Split plane, which is defined by the centroid of the point set and the eigenvector v_2 of the covariance matrix C. This means that the point set is always split along the direction of greatest variation. Splitting is continued until all clusters are within user specified maximum values.

In this thesis, clustering methods were applied and both are incremental, thus appropriate for our case. The results and the analysis of clustering are discussed in Chapter 5.

5 RESULTS

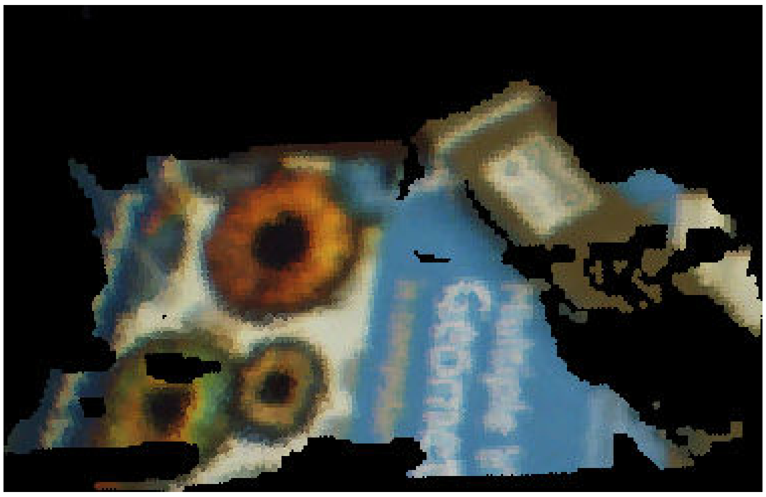
The point clouds presented in Chapter 4 are now processed with the algorithms presented in Chapter 3.3. The software applying these algorithms was presented in the previous chapter. We will now discuss how well the software performs with the different point clouds and how well the different procedures such as the clustering affect to the resulting surfaces.

The main categories in evaluation are the simplification with the clustering, the triagulation of clustered points and the NURBS surface reconstruction from the triangulated surface. In the end we also discuss about practical performance of the different parts of software.

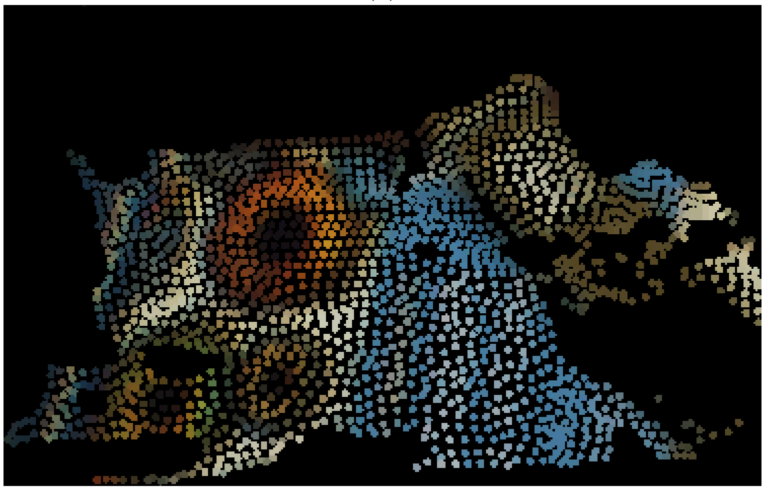
5.1 Selecting and clustering the point clouds

There are 5 different point clouds used in this thesis and most of them has over ten thousand points. Point clouds bigger than few thousand are impossible to process in few seconds with current hardware. Additionally, the point clouds are not only taken from many objects at once but also have much outliers. and that's why we will select certain points from each point cloud. In Figure 30 there are 3 different levels of iterative clustering done, each with different bounds. In Desk point cloud we decided to select points belonging to a certain object. We selected the book and the object on it as our target. The first image in Figure 30 had a bound set to zero and it now has as many clusters as original point cloud. Meaning that it has just one point in each cluster. In Figure 30 (b) there is a clustering result with the bound set to $6 * 10^{-5}$ and in Figure 30 (c) to $2 * 10^{-4}$. The bigger bound creates larger clusters with more points within. These results illustrate only the cluster centroids. The bound value reflects the distances between the clusters.

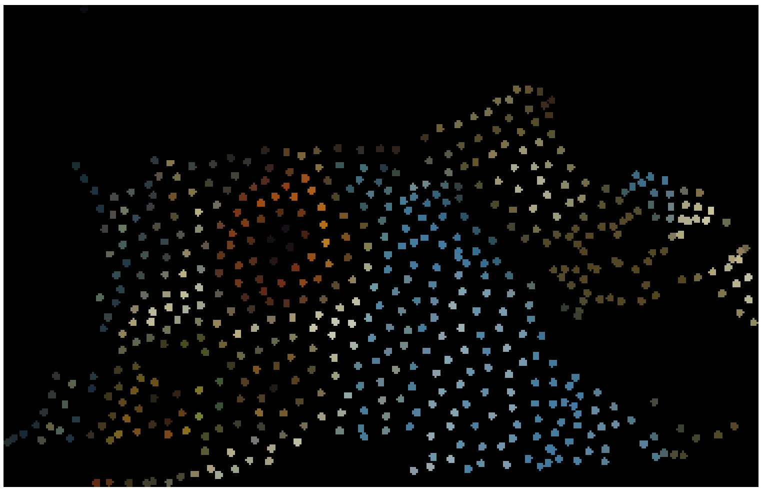
Figure 31 presents same kind of set of images for Stereo2 point cloud. In this point cloud we choose the block on the table as our target set of points. This object, however, is already small and therefore we selected only two different sets. This point cloud also contains a lot of outliers and some measurement error. The first image in Figure 31 shows clustering result with zero bound and the second one is with 10^{-5}. Stereo2 and Stereo3 are also really noisy point clouds as shown in Figure 32. Figure 32 (a) is taken from the side of the table. It seems that the points are wandering through the table due to error in the distance estimation. In Figure 32 (b) is taken from the top. The jittering on the edge



(a)



(b)



(c)

Figure 30. Clustered Desk point cloud. (a) 6708 clusters (equals the original point cloud), (b) 1872 clusters and (c) 582 clusters.

of the table shows the error. These point clouds cannot be turned to surfaces properly without careful selection of desired data points and filtering.

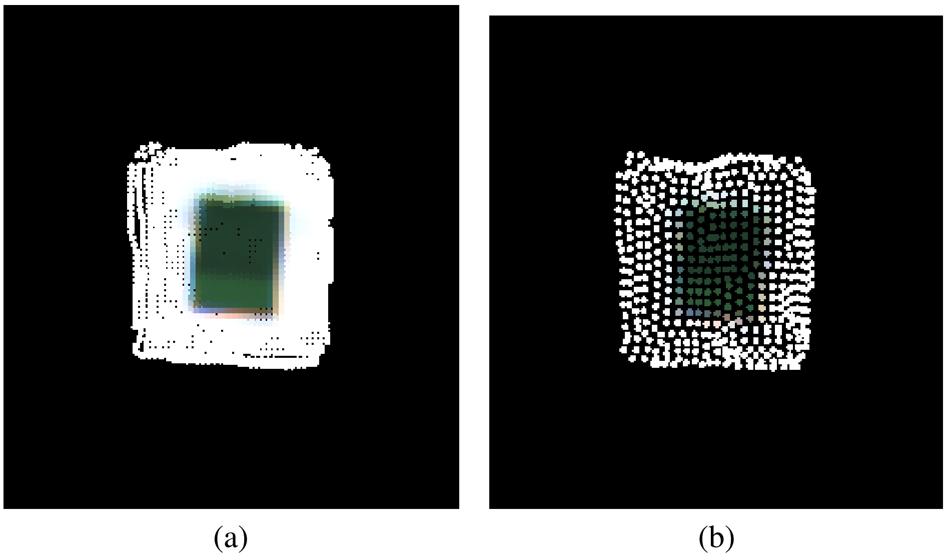


Figure 31. Clustered Stereo2 point cloud. The object is a block from the table (a) 1369 clusters (equals the original point cloud) and (b) 511 clusters.

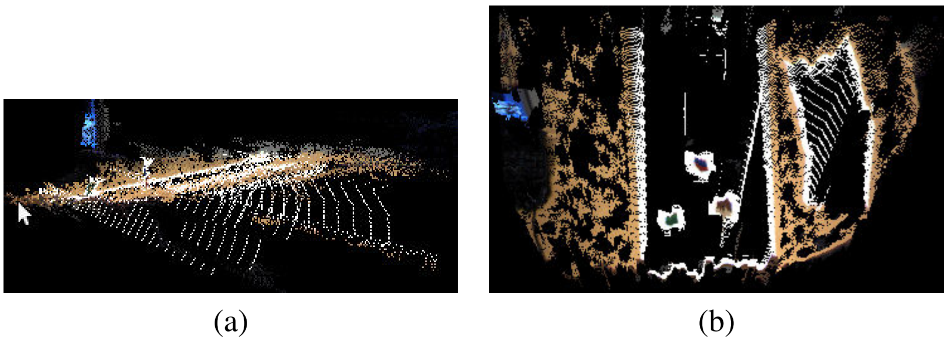


Figure 32. Clustered Stereo3 point cloud. (a) Visible outliers under the table and (b) visible measurement errors.

The third point cloud is presented in Figure 33. Fountain is the biggest point cloud which benefits greatly from the clustering. Fountain also has a lot of visible outliers which all can not be filtered with clustering. The most problematic outliers can be seen in Figure 34. There are dots scattered in the air and without any pruning, there'll be unwanted triangles after the triangulation step.

The points in the front of the fountain are not desired. The set of images are done with bounds set to 0, $6 * 10^{-5}$ and 10^{-4}. The problematic clusters were removed by removing clusters with only a few associated data points.

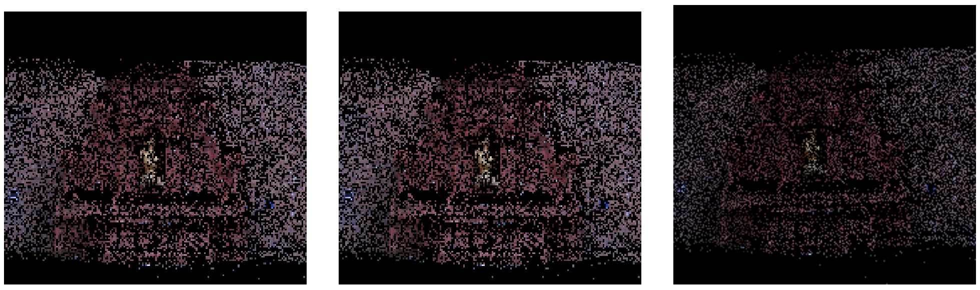

(a) (b) (c)

Figure 33. Clustered Fountain point cloud. (a) 27086 clusters, (b) 13100 clusters and (c) 8433 clusters.

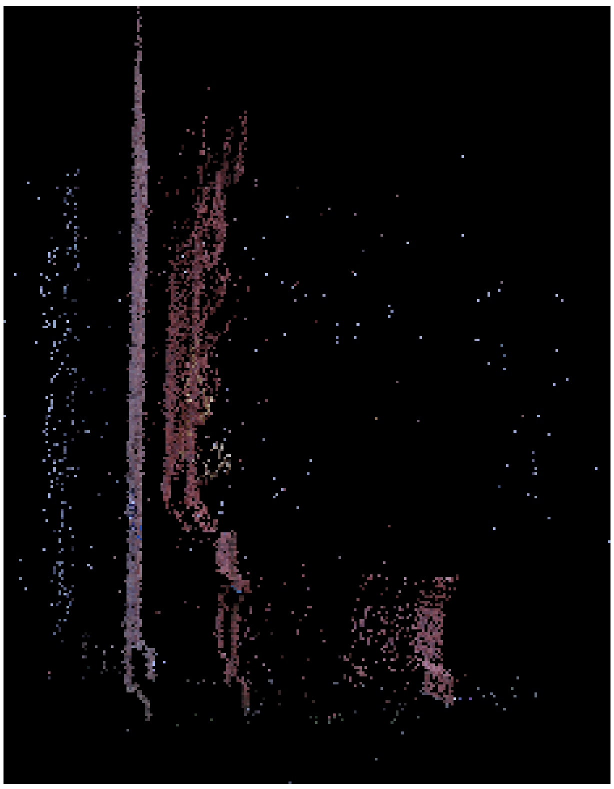


Figure 34. The problematic outliers in Fountain point cloud (from a side).

The last clustered point cloud is presented in Figure 35. Bunny was clustered with bounds set to 0, $3 * 10^{-5}$ and $6 * 10^{-5}$ and the points in this set does not have any outliers.

These clustered point clouds were now in a form that the triangulation algorithm can be applied in a reasonable time. The COCONE algorithm deals well with point clouds with

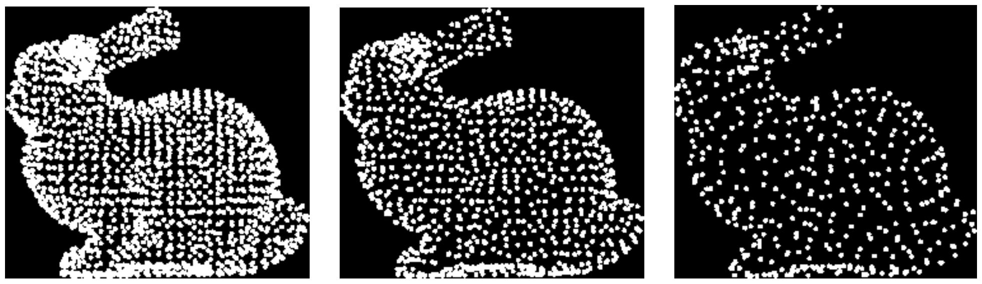


Figure 35. Clustered Standford Bunny point cloud. (a) 1890 clusters, (b) 1050 clusters and (c) 470 clusters.

less than 5 000 points. More points than this explodes computation time.

5.2 Triangulation

The results from triangulations are presented in this section. By default triangulation with Cocone algorithm is done with the default parameter values $\pi/8$ for Alpha and 1.2 for Rho. These settings were work in most cases, but sometimes needs to be adjusted for good results. In the following, we present results with the default parameter values and also results with more carefully selected values.

In the previous section, the clustering results were discussed. However, some clustering results seemed better than others depending on the amount of outliers. All possible combinations are not shown so, we present only results for Standford Bunny, Fountain, Desk and Stereo3.

It was decided to consider a recontruction good if it followed the surface as much as possible. Some missing triangles were accepted in triangulated model as long as the majority of the surface was continous. False triangles were not accepted as they do not represent the form of the model. We also calculated the total distance error from the surface point in point cloud to the generated surface triangle.

In Figure 36 we have the book from Desk point cloud. As shown in the second image in Figure 36, there are not much holes and the surface seems quite continous. Changing alpha value to a greater angle resulted more triangles and less holes, but there were also some false triangles. A clustering results with a bigger bound seemed to reduce error in point clouds, but bigger holes still remain and the error value increased. Naturally, the result is also blurred.

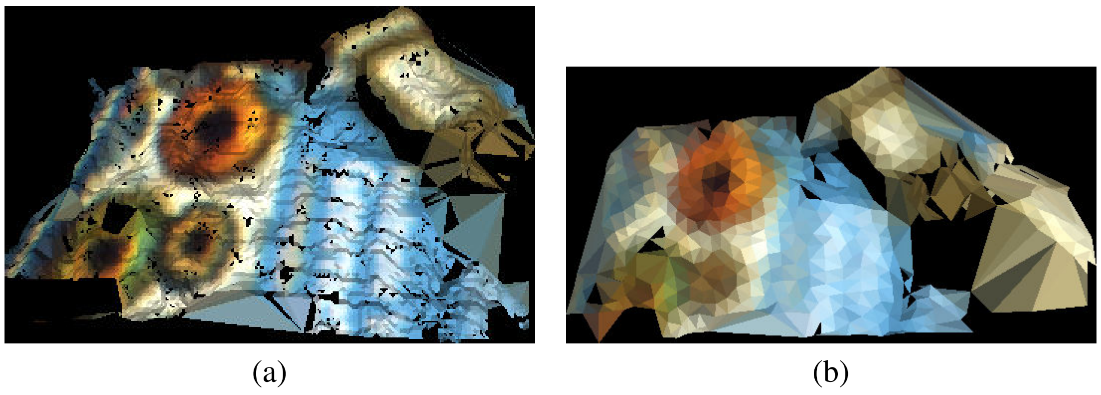


Figure 36. Triangulated book in Desk point cloud. Triangulation made with (a) the full point set, (b) another result with the smallest clustering result (582 clusters).

The triangulation of the selected Stereo3 point cloud is presented in Figure 39. We took the whole table eventhough we knew there was going to be outliers and holes. The results show that even with the clustering the noise cannot be reduced enough.

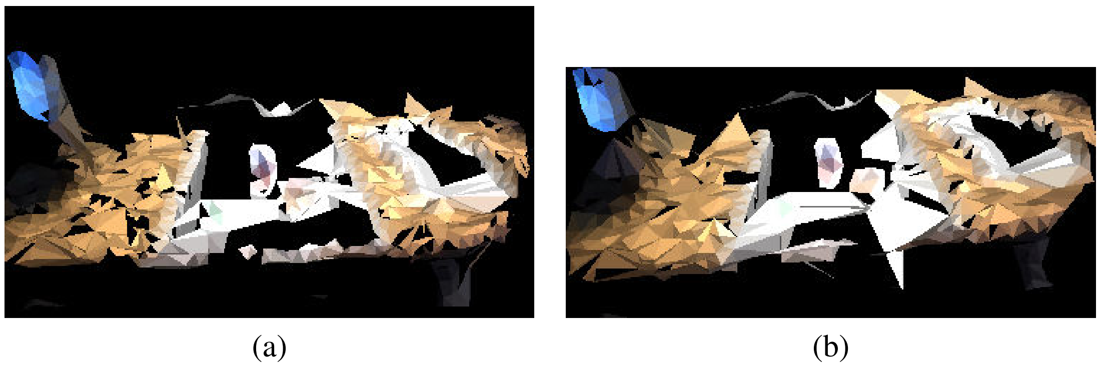


Figure 37. Triangulated Stereo3 point cloud. Triangulation made with (a) 3523 clusters, (b) another result with a smaller point cloud acquired through the clustering step (846 clusters).

The problematic areas of both Desk and Stereo3 are presented in Figure 38.

Standford Bunny is presented in Figure 39. Standford Bunny gives the best results when it comes to the correctness of the reconstruction. However, ears seem to be problematic.

When using the COCONE algorithm the most important aspect of a point cloud is to be as smooth as possible. When there is too much variation, the algorithm cannot decide which point resides on the surface and which does not. That leads to missing triangles. It

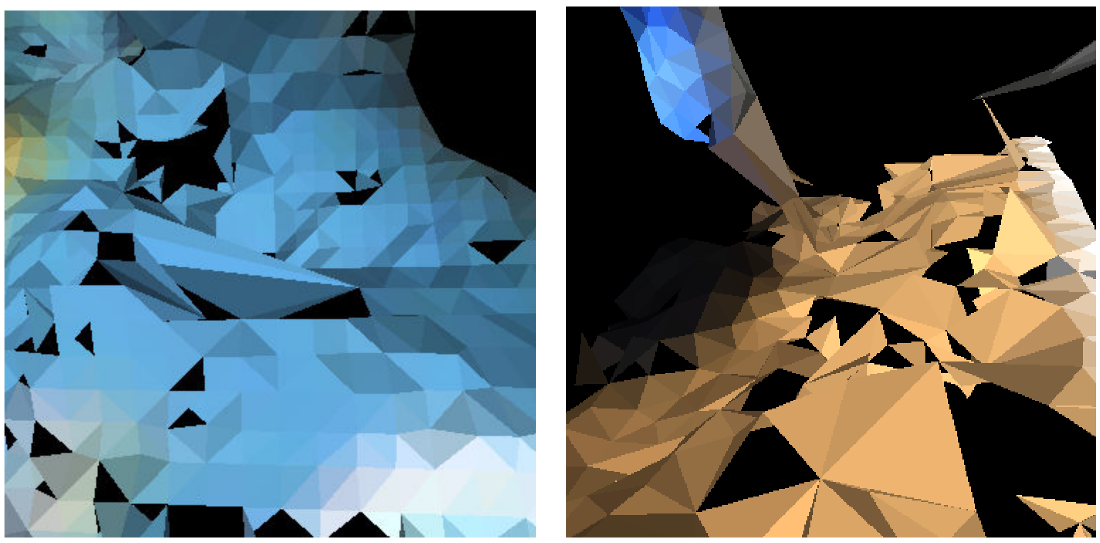

(a) (b)

Figure 38. Problematic triangulations. Visible holes in (a) Desk (b) Stereo2 point clouds.

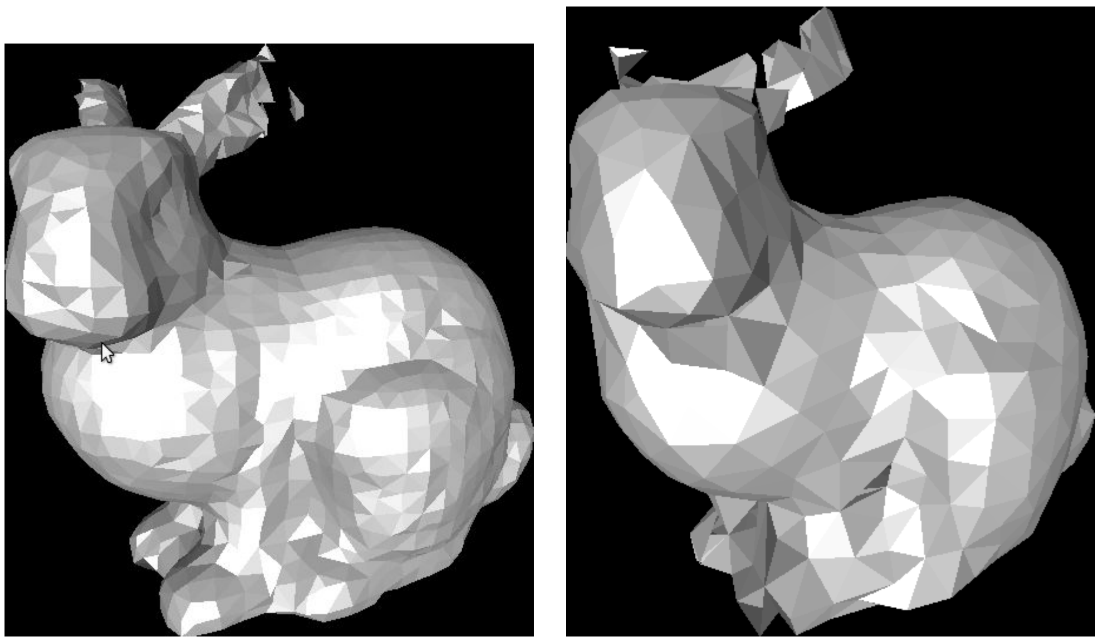

(a) (b)

Figure 39. Triangulated Standford Bunny point cloud. Triangulation made with (a) the full point set and (b) another result with a smaller point cloud acquired through clustering step.

seems, however, that the book from Desk point cloud was pretty good eventhough there was some noise.

5.3 Post-processing with NURBS surfaces

As stated in Chapter 3.2, a triangulated model is required before we can create a NURBS surface model. Before the NURBS surface approximation, the quadrilateral model must be created from the existing triangulated model. The used method works best if the triangulated model is as solid as possible without holes because there is possibility that the triangle can not find a neighboring triangle to merge with, thus leaving a hole to the qudrilateral model. It is worth noticing that quads might be anything with four vertices as the merged triangles can be in any alignment. An example of this procedure can be seen in Figure 40. Mostly quadrilateral domain creation works fine, but many single triangles (unable to create a quad) can be found, especially in Bunny model as it is a closed object and does not have any boundaries. The NURBS surface reconstruction is the only phase without colors.

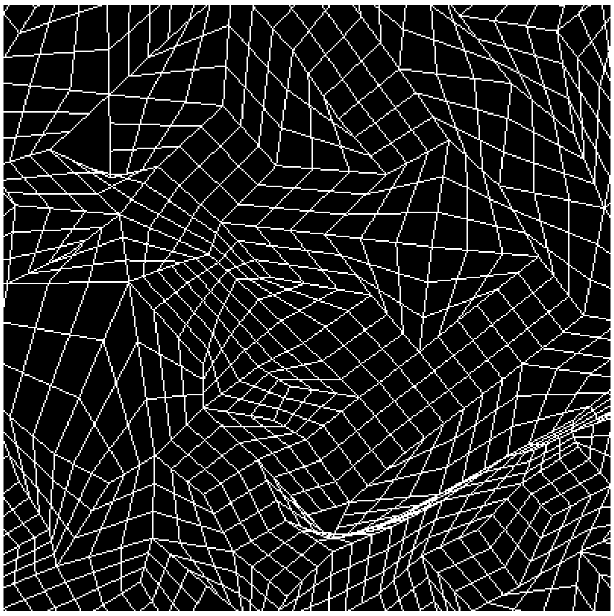


Figure 40. An example of the generated quadrilateral model for NURBS surface approximation.

The most solid triangular models were Desk and Standford Bunny. These two triangulated models were processed to a NURBS surface model. In Figures 42 and 41 the results for the Standford Bunny are presented.

Figure 41 presents a result with small amount of control points (4x4) but many grid points (15x15). The overall result is smooth over points. However, the grid structure has a few left-over triangles, since Bunny is a solid model and the breadth-first quad extraction

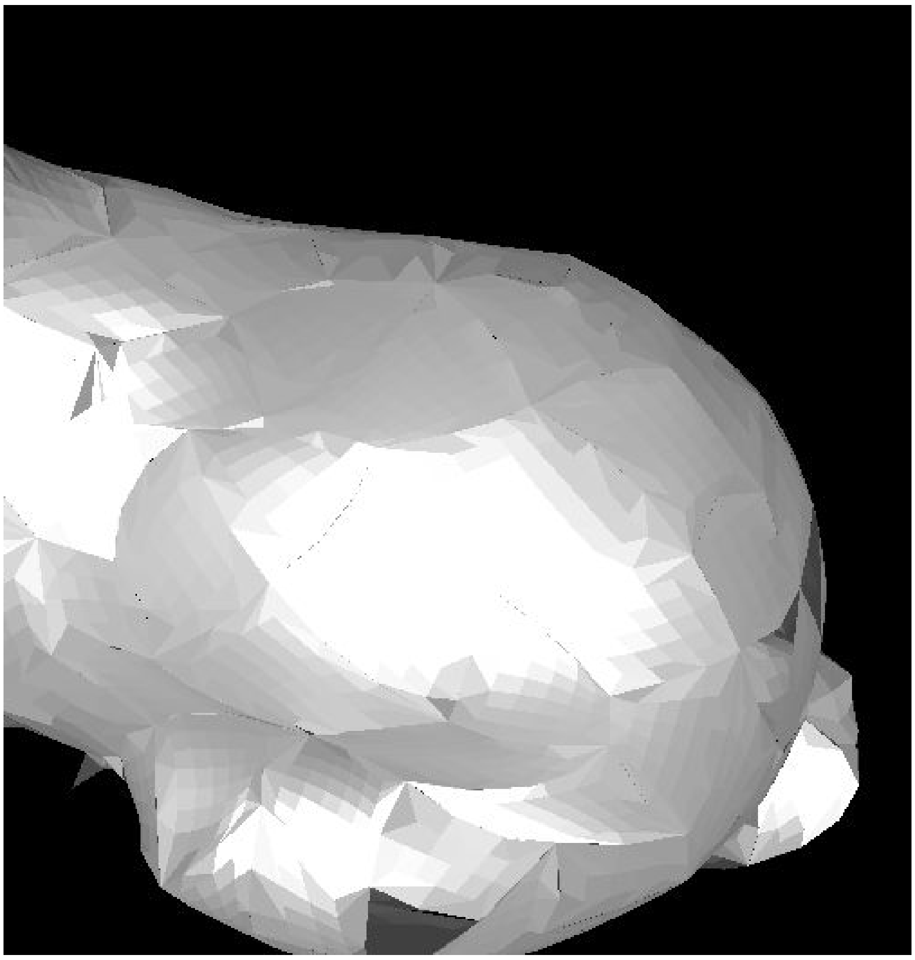


Figure 41. The back of Standford Bunny processed into a NURBS surface model.

algorithm will meet itself at some point. This meeting point may create stray triangles in the same manner as boundaries. The NURBS model creation worked best with a large grid size and with a lot of control points. The degrees for the NURBS surfaces should be at least 3 to gain good and smooth results.

The noise (black/gray areas) in Figure 42 are generated because of the difficulties to present badly formed grid. This problem is discussed in Chapter 6.

The triangular model of Desk as NURBS surface is presented in Figure 43. This triangular surface with boundaries and few holes was very succesfully turned into quadrilateral domain. Only two visible left-over triangles and other holes were in the original triangular model.

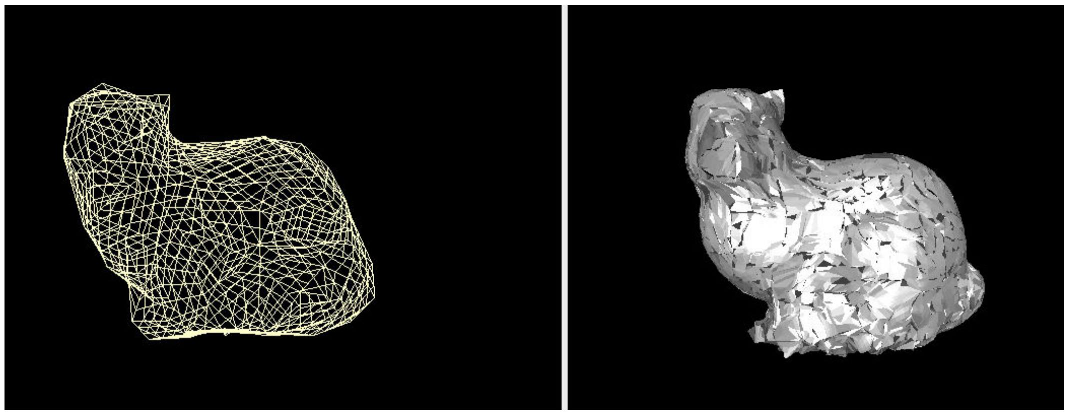


Figure 42. Standford Bunny processed into a NURBS surface model. On the left is the grid structure and on the right is the resulting NURBS model.

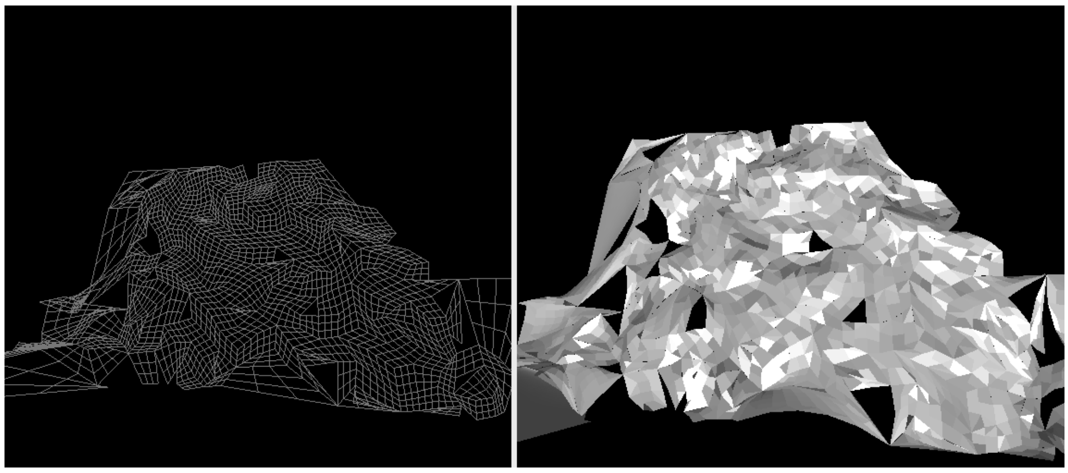


Figure 43. Triangular model made of Desk point cloud is processed into a NURBS surface model. On the left is the grid structure and on the right is the resulting NURBS model.

5.4 Practical performance

In this section we will discuss about the performance of the software. The software performance is estimated by the following criteria: speed of reconstruction algorithms, solidity of the model and the overall error of the model (if possible). The computer used for running the software was a low-end desktop computer with 1.6Ghz Intel processor with 512 megabytes of memory. Eventhough processing might be fast in some cases, the software still needs user interaction to be fully operational which is the slowest part of the program.

In Table 3 all essential parts of the software are presented evaluated. The speed is measured in seconds. The times tell the shortest time with the smallest set of points and the longest with the largest set of points. The NURBS surfaces are only generated from the smallest set of points. These times shows us the magnitude of software performance on a low-end computer. The overall complexity is also evaluated for each part. Clustering has the optimal and the worst case scenario presented.

The total time taken by the system to create a triangulated system only takes a few seconds with less than thousand points. To create the NURBS surface model, it requires a lot more calculation as it requires quadrilateral domain creation and the NURBS surface approximation.

Errors in models in Table 3 are not compatible between the models. The models with widely scattered points have bigger error because the distances between the points are longer and the scale can be anything. However, the error value can be used to evaluate how much the clustering affects to the point cloud with different parameters.

Table 3. Software speed (in seconds) with different point sets.

Phaze	*Desk*	*Bunny*	*Stereo3*	*Complexity*
Point set size	582 - 6 708	470 - 1 890	846 - 3 523	
Loading point cloud	0.45s	0.08s	1.66s	$O(n)$
Clustering	0.26s - 6.78s	0.01s-0.13s	0.05s - 8.68s	$O(n) - O(n^2)$
Triangulation	5s - 57s	1s - 2.46s	2.819s - 43s	$O(nlogn)$
NURBS model creation	13s	11.55s	15s	$O(n^2)$
Error calculation	[10s - 183s]	[2s-16s]	[5s - 157s]	$O(n^2)$
Error in model	0.0745	0.0308	0.00934	

Additionally, the overall solitude of the model is evaluated. Desk was a very noisy point cloud, however, we were able to get quite smooth results by clustering. In clustered version we had only few visible holes in the book. With many points the resulting surface had many missing triangles. Stereo3 point cloud with all possible points had too much missing triangles. Bunny were reconstructed very well, only a few missing triangles were found in the feet of the bunny. The ears of Bunny were problematic too.

If the original triangular model had missing triangles, the NURBS surface models had even bigger holes. However, with a small triangulated model and a large grid the resulting NURBS surfaces seemed accurate. Bunny point cloud is a good example of this. The NURBS surfaces were more difficult to evaluate because they lacked color which is quite important in this case.

6 DISCUSSION AND CONCLUSIONS

This thesis studied 3D surface modeling from point clouds. We studied the current state of geometrical surface reconstruction algorithms, stereo imaging, clustering and geometric models using NURBS surfaces. This thesis used many methods together to create triangular and parametric surface models from a point cloud. We used a triangular model to define the topology of the underlying model and further processed it to a NURBS surface model. We found that clustering smoothens and simplifies point clouds in such way that it is easily processed by the COCONE surface algorithm. The pre-processing by clustering also produces better NURBS surfaces in our post-processing step.

The results showed that too coarse clustering affects negatively to the triangulation, which estimates the surface topology for NURBS. The best results were gained by carefully selecting appropriate clustering and good triangulation parameters alpha and rho. Basically, too many triangles not belonging to the surface means too big alpha value. A lower alpha value produces less candidate triangles because it only accepts triangles that are parallel to the underlying surface.

Correspondingly, having too many clusters with only a few points per cluster leads to a very good surface estimation by the COCONE algorithm, but the NURBS surface is bad since, there are no points in the clusters to estimate the grid points for NURBS surfaces. When the triangulated model is sufficiently simple, then the NURBS model can estimate the original point cloud via grid points. The overall estimation suffers if the triangulated model is too simple. NURBS surfaces also suffered from discontinuity as the number of control point arose (over 15x15). The suggested reason for this is the number of grid points with too few cluster points. With only a few cluster points, the grid points are brought near to each other, resulting a grid which is difficult to approximate accurately. However, even with the discontinuity the whole model showed that the idea of approximating a NURBS grid with cluster points works.

Point clouds used in this thesis had a lot of outliers, errors and the points were unevenly distributed. Eventhough we selected only certain areas to be processed, they still contained outliers and other errors. The errors appeared as holes or missing triangles in the triagular model because angles between two adjacent triangles were too big. In the COCONE algorith too big angle between two adjacent triangles meant that the edge was sharp and all adjacent triangles needed to be removed. Basically, COCONE needs smooth point clouds with as little error as possible. These restrictions are one of the weaknesses of the COCONE algorithm. Additionally, the algorithm can not estimate a model globally, but only locally for each point at a time. Despite its weaknesses, the results showed that COCONE is quite fast and a simple surface reconstruction algorithm.

Most of the point clouds from stereo cameras seem to have a lot of error due to its computational principles. The clustering was applied to reduce this problem but it did not seem to help enough at some situations.

This thesis shows that it is possible to create NURBS models from point clouds by using a triangulation algorithm as an estimation of the surface topology. The surface reconstruction algorithm plays an important role because if the surface is not solid, we end up with an unwanted NURBS surfaces.

6.1 Future Work

As it can be seen in the results, many things can be improved in this system. The triangulated surface tend to have a lot of holes and missing triangles which also hinder the NURBS surface reconstruction. To further improve the geometric accuracy of the reconstruction one may use AMLS presented by Dey et al. [19]. It dramatically slows the reconstruction process but it also can remove the problem of missing triangles and holes. Since the COCONE algorithm uses boundary detection, the holes cannot be filled reasonably. We need to either change the algorithm or improve the current one.

It was also noticed that selecting only parts of a point cloud with a bounding box was surprisingly hard, as the desired regions were not in shape of a single cube. A better solution would be able to draw an arbitrary volume around the object. Additionally user could just remove or add more objects to the selection.

The solution in this thesis finds out a desired surface topology using the COCONE algorithm and further transfer it to the quadrilateral domain. Another way of finding out the surface is to use signed distance functions, tangent plane estimation and contour tracking. These methods were not studied in this thesis, but they could solve certain problems in the surface modeling.

One goal of this thesis was to have an incremental reconstruction. However, the COCONE algorithm does not provide an easy way to do this directly. Every time we add a point to a

Delaunay triangulation we need to check all triangles near that point and do pruning and manifold extraction again. Possibility of doing this step incrementally should be studied by replacing the pruning and the manifold extraction algorithms with more incremental versions.

The resulting NURBS model needs some improvement on continuaty. We suggest that the future adjustments require the grid points to be only moved in normal direction. This way we do not end up with too many grid points close together. Additionally a good solution would be to manipulate the boundary points of the each grid in such way we get the same tangent for neighboring grids. This would create even smoother model but it can also remove some details as NURBS surface doesn't follow all grid points.

The produced software can faster. To improve performance a better understanding and integration of CGAL library could provide faster results. The software also needs a better GUI as currently every step is done in a different window. One solution for this might be to create a master window which hosts all other windows. This approach would allow us to create multiple results from multiple different point clouds.

REFERENCES

[1] Nina Amenta, Sunghee Choi, Tamal K. Dey, and Naveen Leekha. A simple algorithm for homeomorphic surface reconstruction. pages 213–222, 2000.

[2] Franco P. Preparata and Michael I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[3] Christoph Strecha, Wolfgang von Hansen, Luc Van Gool, Pascal Fua, and Ulrich Thoennessen. On Benchmarking Camera Calibration and Multi-View Stereo for High Resolution Imagery. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[4] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[5] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[6] [www] real-time mosaicing and 3-d reconstruction. Available at `http://www.it.lut.fi/project/rtmosaic/`. Cited at 13.1.2009.

[7] Ruigang Yang and Marc Pollefeys. A versatile stereo implementation on commodity graphics hardware. *Real-Time Imaging*, 11(1):7–18, 2005.

[8] Pekka Paalanen, Ville Kyrki, and Joni-Kristian Kamarainen. Towards monocular on-line 3d reconstruction. In *ECCV 2008 Workshop: Vision in Action: Efficient strategies for cognitive agents in complex environments.*, Oct 2008.

[9] Mark Pauly, Markus Gross, and Leif P. Kobbelt. Efficient simplification of point-sampled surfaces. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 163–170, Washington, DC, USA, 2002. IEEE Computer Society.

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

[11] Les A. Piegl and Wayne Tiller. *The Nurbs Book*. Springer, 1997.

[12] Jean-Daniel Boissonnat and Frédéric Cazals. Smooth surface reconstruction via natural neighbour interpolation of distance functions. In *SCG '00: Proceedings of the sixteenth annual symposium on Computational geometry*, pages 223–232, New York, NY, USA, 2000. ACM.

[13] Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 415–421, New York, NY, USA, 1998. ACM.

[14] Michal Varnuska and Ivana Kolingerova. Improvements to surface reconstruction by the crust algorithm. In *SCCG '03: Proceedings of the 19th spring conference on Computer graphics*, pages 89–97, New York, NY, USA, 2003. ACM.

[15] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust. In *SMA '01: Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 249–266, New York, NY, USA, 2001. ACM.

[16] Tamal K. Dey and Samrat Goswami. Tight cocone: a water-tight surface reconstructor. In *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 127–134, New York, NY, USA, 2003. ACM.

[17] Tamal K. Dey, Joachim Giesen, and James Hudson. Delaunay based shape reconstruction from large data. In *PVG '01: Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics*, pages 19–27, Piscataway, NJ, USA, 2001. IEEE Press.

[18] Tamal K. Dey and Samrat Goswami. Provable surface reconstruction from noisy samples. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 330–339, New York, NY, USA, 2004. ACM.

[19] Tamal K. Dey and Jian Sun. An adaptive mls surface for reconstruction with guarantees. In *SGP '05: Proceedings of the third Eurographics symposium on Geometry processing*, page 43, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

[20] Kruth J.P. Ma W. Nurbs curve and surface fitting for reverse engineering. *The International Journal of Advanced Manufacturing Technology*, 14(12):918–927, Dec 1998.

[21] Weiyin Ma and J.-P. Kruth. Parameterization of randomly measured points for least squares fitting of b-spline curves and surfaces. *Computer-aided Design*, 27(9):663–675, 1995.

[22] I. Park, I. Yun, and S. Lee. Constructing nurbs surface model from scattered and unorganized range data. In *"Proc. 2nd International Conference on 3-D Digital Imaging and Modeling"*, Oct 1999.

[23] In Kyu Park and Sang Uk Lee. Geometric modeling from scattered 3-d range data. *The International Conference on Image Processing*, 2:712–715, Oct 1997.

[24] Tamal K. Dey and Joachim Giesen. Detecting undersampling in surface reconstruction. In *Symposium on Computational Geometry*, pages 257–263, 2001.

[25] [www] the nurbs++ library. Available at `http://libnurbs.sourceforge.net/`. Cited at 10.2.2009.

[26] [www] the computational geometry algorithms library. Available at `http://www.cgal.org`. Cited at 13.1.2009.

[27] [www] qt software. Available at `http://www.qtsoftware.com/`. Cited at 20.1.2009.

[28] [www] the stanford 3d scanning repository. Available at `http://www-graphics.stanford.edu/data/3Dscanrep/`. Cited at 18.2.2009.

[29] [www] point grey research inc. - bumblebee2. Available at `http://www.ptgrey.com/products/bumblebee2/index.asp`. Cited at 17.1.2009.

[30] [www] triclops sdk technical datasheet. Available at `http://www.ptgrey.com/products/triclopsSDK/triclops.pdf`. Cited at 17.1.2009.

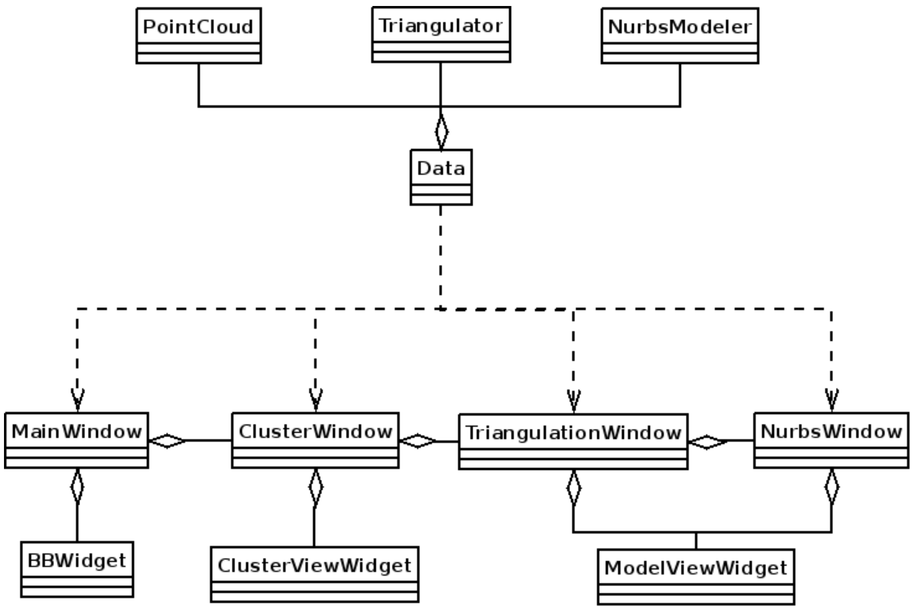
[31] Jianning Wang, Manuel M. Oliveira, and Arie E. Kaufman. Reconstructing manifold and non-manifold surfaces from point clouds. In *IEEE Visualization*, page 53, 2005.

[32] Yizhou Yu. Surface reconstruction from unorganized points using self-organizing neural networks. In *IEEE Visualization 99, Conference Proceedings*, pages 61–64, 1999.

BUMBLEBEE2 STEREO CAMERA

Specification	*Information*
Imaging Sensor	Sony 1/3" progressive scan CCD, ICX424 (648x488 max pixels), 7.4μm square pixels
Baseline	12cm
Focal lengths	3.8mm with 66°HFOV or 6mm with 43°HFOV
A/D Converter	12-bit analog-to-digital converter
White Balance	Automatic/manual
Frame rate	48 FPS
Interfaces	6-pin IEEE-1394a for camera control and video data transmission.
Voltage Requirements	8-32V via IEEE-1394 interface or GPIO connector
Power consumption	2.5W at 12V
Gain	Automatic
Shutter	Automatic/Manual, 0.01ms to 66.63ms at 15 FPS
Trigger Modes	DCAM v1.31 Trigger Modes 0, 1, 3, and 14
Signal to noise ratio	60db
Dimensions	157 x 36 x 47.4mm
Mass	342g
Camera Specification	IIDC 1394-based Digital Camera Specification v1.31
Lens mount	2 x M12 microlens mount
Emissions Compiance	Complies with CE rules and Part 15 Class A of FCC Rules
Operating Temp	Commercial grade electronics rated from 0 to 45 celcius
Storage Temp	-30 to 60 celcius

SOFTWARE UML DIAGRAM


PointCloud
Triangulator
NurbsModeler
Data
MainWindow
ClusterWindow
TriangulationWindow
NurbsWindow
BBWidget
ClusterViewWidget
ModelViewWidget