UNIVERSITY OF WEST BOHEMIA

Faculty of Applied Sciences

DOCTORAL THESIS

Plzeň, 2005

Michal Varnuška

University of West Bohemia

Faculty of Applied Sciences

DOCTORAL THESIS

in partial fulfillment of the requirement for the degree of
Doctor of Philosophy
in specialization

Computer Science and Engineering

Ing. Michal Varnuška

Surface Reconstruction of Geometrical Objects from Scattered Points

Supervisor: Doc. Dr. Ing. Ivana Kolingerová
Date of state doctoral exam: 30.6.2004
Date of thesis consignation: 20.5.2005

Plzeň, 2005

ZÁPADOČESKÁ
UNIVERZITA
V PLZNI

Fakulta aplikovaných věd

DISERTAČNÍ PRÁCE

k získání akademického titulu doktor
v oboru

Informatika a výpočetní technika

Ing. Michal Varnuška

Surface Reconstruction of Geometrical Objects from Scattered Points

Školitel: Doc. Dr. Ing. Ivana Kolingerová
Datum státní doktorské zkoušky: 30.6.2004
Datum odevzdání práce: 20.5.2005

V Plzni, 2005

Prohlášení

Předkládám tímto k posouzení a obhajobě disertační práci zpracovanou na závěr doktorského studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že tuto práci jsem vypracoval samostatně s použitím odborné literatury a dostupných pramenů uvedených v seznamu, jenž je součástí této práce.

V Plzni, 20.5.2005 Michal Varnuška

Surface reconstruction of geometrical objects from scattered points

Michal Varnuška

Abstract

The surface reconstruction is a common problem in the modern computer graphics and computational geometry. There are many applications which need to work with a piecewise linear approximation of existing real 3D objects. One of the methods for acquiring these models is the digitization of the real 3D object using many types of devices followed by the point cloud reconstruction.

This thesis deals with the surface reconstruction from the point cloud. For the reconstruction we use the CRUST algorithm which works on the principle of selecting surface triangles from Delaunay tetrahedronization using the information from dual Voronoi diagram.

This algorithm has nice properties, but as other reconstruction algorithm, it is not working properly for each kind of data. Our goal was to develop some steps to improve the quality of the whole reconstruction and which enlarge number of datasets that can be processed by the algorithm.

We have developed several improvements which can be divided into three groups. The first group of improvements deals with the points preprocessing and it prepares the data, if necessary, to be better reconstructed. The second group of improvements aims to the process of surface reconstruction. The last group orientates to the resulting triangle mesh and it tries to repair there possible errors. Most of our improvements can be used with any other reconstruction algorithms, too. We have also compared our results of surface reconstruction with other existing algorithms with positive and stimulating results.

This work was supported by

- the Ministry of Education of the Czech Republic – project MSM 235200005
- the project FRVŠ G1/1349 2004
- the project AKTION 36p9, the project KONTAKT 16-2003-04

Copies of this report are available on
http://www.kiv.zcu.cz/publications
http://herakles.zcu.cz/publications.php
or by surface mail on request sent to the following address:

University of West Bohemia
Department of Computer Science and Engineering
Univerzitní 8
306 14 Pilsen, Czech Republic

Surface reconstruction of geometrical objects from scattered points

Michal Varnuška

Abstrakt

Rekonstrukce povrchů z roztroušených bodů je stále hodně diskutované téma v oblasti počítačové grafiky a výpočetní geometrie, jelikož mnoho aplikací z těchto oblastí potřebuje pracovat s modely reálných objektů. Jedna z metod, jak získat modely reálných objektů, je jejich digitalizace různými zařízeními následovaná rekonstrukcí navzorkovaných bodů.

Tato práce se zabývá právě krokem rekonstrukce povrchů z navzorkovaných bodů. Pro vlastní rekonstrukci používáme CRUST algoritmus, který vybírá povrch z množiny trojúhelníků vytvořené Delaunayovou tetrahedronizací za pomoci Voronoiova diagramu.

Tento algoritmus má mnoho dobrých vlastností, nicméně, jako všechny ostatní algoritmy pro rekonstrukci, neumí zpracovat jakýkoliv typ dat. Naším úkolem proto bylo vyvinout takové kroky a vylepšení, které nám umožní rozšířit množinu typů zpracovávaných dat a dosáhnout tak jejich bezproblémové rekonstrukce.

Vyvinuli jsme několik vylepšení, které můžeme rozdělit do tří skupin. První skupina vylepšení se zabývá předzpracováním bodů před vlastní rekonstrukcí. Druhá skupina se pak zaměřuje na proces vlastní rekonstrukce a poslední skupina se orientuje na zpracování výsledné trojúhelníkové sítě a snaží se napravit chyby, které v průběhu rekonstrukce mohou vzniknout. Většina prezentovaných vylepšení je navíc použitelná i pro jiné přístupy v rekonstrukci povrchů. Následně provedené měření a porovnání s některými existujícími algoritmy nám ukázalo, že kroky vedoucí k úpravě algoritmu byly správné a v případě problematických dat dokáží účinnost rekonstrukce výrazně zvýšit.

Tato práce byla podporována následujícími projekty:
- projekt MSM 235200005
- projekt FRVŠ G1/1349 2004
- projekt AKTION 36p9 a project KONTAKT 16-2003-04

Kopie této práce je možné nalézt na:
http://www.kiv.zcu.cz/publications
http://herakles.zcu.cz/publications.php
nebo na adrese:

Západočeská univerzita
Katedra informatiky a výpočetní techniky
Univerzitní 8
306 14 Plzeň, Česká republika

Acknowledgments

At this place I would like to thank to all the people who have helped me with my PhD studies. At the first place the greatest thanks go to my supervisor doc. Dr. Ing. Ivana Kolingerová for big support, helpful discussions and especially wonderful patience, then I would like to express my gratitude to the head of the graphical section of our department, to prof. Ing. Václav Skala, CSc. for providing good conditions under which the work has been possible.

Great thanks go to all my colleagues in the room UK411 in our department of Computer Science and Engineering, thus to (alphabetically) Ing. Josef Kohout, Ing. Jindřich Parus, Ing. Jan Patera and Ing. Petr Vaněček, all engineers and PhD expectants, for fruitful discussions about many scientific problems and for collective programming in Q# language.

Thanks belong to other people from the Faculty of Applied Sciences, especially from the Department of Computer Science and Engineering and from the Department of Mathematics. Thanks also belong to the VRVis research group from the Technical University of Graz, Austria for providing real datasets and nice time in Graz and to prof. Tamal. K. Dey from the State University of Ohio, USA, for providing his surface reconstructors.

There are also many other people from different universities and institutions which helped me in my scientific research, e.g., RNDr. Arnošt Šarman, CSc. and his wife doc. RNDr. Jana Šarmanová, CSc. from the Technical University of Ostrava.

And the last but not least, my great thanks belong to my family for its support and understanding during my PhD studies.

Table of Contents

1. Introduction

Surface reconstruction problem is a theme which is very often mentioned in the area of computer graphics and computational geometry. Many algorithms based on various approaches have been developed and many newer algorithms are still under development showing that this area is under extensive investigation. The computational and visualization capabilities of the computers have dramatically increased during past years as well as the prices of scanning devices have decreased. Therefore a wide range of applications which need to work with a piecewise linear surface approximation is not restricted to the use of models with a low number of polygons but it can use the models which describe best the reality. The generation of such models which satisfy high requirements of the applications is required in many areas of the human activity, such as in computer graphics, computer games, CAD, geography, scientific visualizations and simulations, e-commerce, medicine, architecture or archeology.

In this thesis we would like to give a brief survey of the existing algorithms for the task of surface reconstruction and to present one possible approach which we have developed during the investigation of this area of computer graphics. Given only a set of points sampled without any other information from some real surface, we would like to create a triangle mesh which interpolates these points and which is as close as possible to the original surface. There is a lot of factors which can affect the success of the reconstruction - the points may not be uniformly sampled, they can contain undersampled areas, boundaries, places where two parts of surface appear very close, the points can be affected by a noise or outliers, we can have more points than our current hardware is able to process. All these problems should be solved by the surface reconstruction algorithm and it is not simple to cover all the problematic places. During the last twenty years the surface reconstruction problem has been addressed by many authors and still none of existing approaches is able to process any data.

Our approach to the surface reconstruction is built on the improvements to the already existing CRUST algorithm developed by Amenta et al. [Amenta98b]. Although this algorithm is nice working, it has big problems if the data does not satisfy the sampling criterion. Unfortunately, many of the datasets we have do not satisfy the required sampling criterion, therefore, we have developed some improvements - several preprocessing steps which prepare the data to be better reconstructed, then we have made some improvements to the CRUST and some postprocessing steps repairing the output surface. All the developed steps were designed after that we have implemented the CRUST and observed its behaviour. Many of the developed steps do not depend on the reconstruction algorithm but may be used in any other approach. After several tests and comparing with other surface reconstruction algorithms we can say that our approach is widely usable and brings nice results.

1.1. A thesis organization

This thesis can be divided into two parts – first four sections are theoretical background and the last eight sections cover our experience, work and contribution. In this chapter we introduce briefly the data acquirement together with sampling criteria. Next, Chapter 2 deals with the most often used terms and describes the Delaunay tetrahedronization, its dual Voronoi diagram, medial axis and briefly an introduction to topology. Chapter 3 presents the state of the art of the surface reconstruction algorithm while Chapter 4 describes the CRUST algorithm which is a base for our approach and a similar COCONE algorithm. The observed problems of the surface reconstruction using the CRUST algorithm are analysed in Chapter 5.

We have developed several steps to improve the reconstruction of problematic datasets, the overview of our proposed solution is presented in Chapter 6. Chapter 7 deals with the preprocessing steps and two approaches to points denoising together with one solution for large data reconstruction are shown. Chapter 8 presents step by step the surface reconstruction phase with our improvements. The postprocessing step dealing with the mesh improvement, such as overlapping triangles filtering, boundary triangles deleting and holes filling, is described in Chapter 9. We have tested a partly distributed version of our reconstruction approach which is presented in Chapter 10. The results and time of some datasets reconstruction together with the comparison with four existing algorithms is shown in Chapter 11 and Chapter 12 concludes the thesis.

1.2. Points acquisition

The way from the real object to the computer model is long but we can divide it roughly into two phases – the hardware phase and the software phase. In the first phase some hardware measure devices scan the real object and some information about the object, such as points on the surface, colour, curvature or normal vectors at the points, etc., are obtained. This information is then utilized in the second software phase of the process where the model of the real object is reconstructed. This thesis deals with this phase and one possible approach how to obtain the models from the scattered point data is presented.

The shape of real objects may be acquired by many types of techniques, with a wide range in the cost of acquisition hardware and in the accuracy and detail of the geometry obtained. On the high cost end, the object can be CAT (computerized axial tomography) scanned and the object surface obtained with an isosurface technique, on the low cost end we can use various techniques which can obtain the data from a set of pictures. Technical equipment for the 3D points scanning should fulfil the following conditions [Bernardini00]:

- low noise
- guaranteed high accuracy
- high speed
- low cost
- automatic operation
- no holes

It is not simple to satisfy all these conditions. The noise brings big problems for many of the developed algorithms, therefore, an aim is to minimize it. This is related to the requirement of high accuracy, when the device is not very accurate, we cannot expect the data without noise. Automatic work together with high speed and low cost of the scanning device is important, too, as the objects can be very large and for the user it is difficult and boring to operate with a slow scanner. The condition that the scanned surface should not contain holes is also difficult to keep. The scanned object usually sits or hangs somewhere, so there are places invisible for the scanner and they look as nonexistent boundaries in the model, or there can be some inner holes where the scanner cannot sample (scanners, such as CAT, working on principle of nondestructive ray infiltration to the object, are not affected by this condition).

Important properties of 3D scanners are scanning resolution and accuracy. The accuracy is a requirement of how close the measured value is to the true value on the surface. The absolute accuracy of any given measurement is unknown, but the precision is guaranteed by the manufacturer. The absolute value of the error increases with the distance between the scanner and the object. The resolution is the smallest distance between two points that the device measures, but this can be different from the accuracy. For example, devices which project stripes on the object may be able to find the depth at a particular point with a submillimetre accuracy, but because the stripes have some width, the system is able to scan data over the surface in a millimetre resolution.

The 3D scanners are divided into two big groups depending on the method of data acquirement: contact and contactless methods. The contactless methods use several kinds of sensors, the contact methods may use a CAT, a laser range scanner, a sonic scanner or just some set of pictures viewed from different angles. Next paragraph describes an example of a typical laser range scanner.

A lighting system produces a pattern of a light (Fig. 1.1) which is projected to the surface. The pattern may be a spot or a line, sometimes a detailed pattern formed by an ordinary light source passing through a mask or slide. A sensor, typically a CCD camera, samples the reflected light from the object surface. Software provided with the scanner computes an array of depth values, which can be converted to the 3D points using scanner coordinate system with a calibrated position and orientation of the light source and the receiver. The data quality of this type of scanner may be affected by the properties of the scanned surface, bad results are obtained on shiny surfaces, surfaces with low albedo or on surfaces which have subsurface scatterings. The 3D scanners augment sometimes the 3D points with additional information, e.g. a colour, a curvature, a normal vector or a set of sharp edges, which can substantially simplify the surface reconstruction.

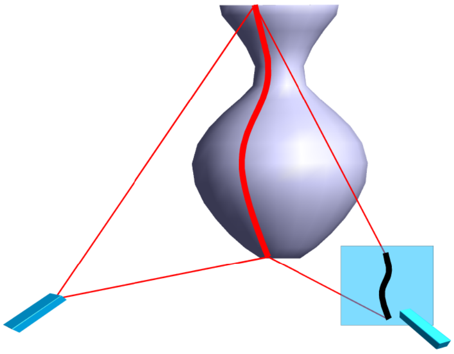


Fig. 1.1: Principle of E^3 scanning by laser scanner.

1.3. An input to the reconstruction algorithm

After having some 3D data, it is necessary to use some algorithm to obtain the original shape of the surface or something very close. The exact surface equal to the original object surface is in most cases impossible to be obtained because the sampling can never be so accurate and the resolution will be almost always bigger than the tiniest features (details are explained in the next section). So in the beginning of surface reconstruction process we have in our case a point cloud *P* sampled from an unknown object or objects *S*, whereas the distribution of points density is unknown, it can vary or it can contain noise.

- *input* : set *P* of 3D points *p* sampled from surface *S*:

$$\forall\, p \in P\,,\, P \subset S : p = [x\,,\, y\,,\, z]\,,\, x\,,\, y\,,\, z \in R \tag{1}$$

- *output* : surface *S′* interpolating or approximating the original surface *S*

The output surface should be close to the original surface and often is in the form of a triangle mesh, but other representation, such as surface patches, is sometimes used, too. The problem of reconstruction is not simple and many algorithms were developed dealing with it. But no algorithm can handle all kinds of data. The acquisition pipeline is illustrated in Fig. 1.2. The object is sampled in the scanning process and the points lying on the surface of object are obtained. These points are used as the input to the surface reconstruction algorithm whose output is the triangle mesh interpolating/approximating the surface.

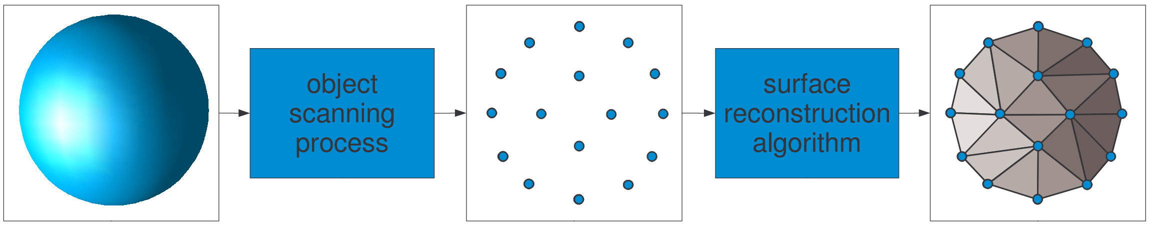


Fig. 1.2: 3D model acquisition pipeline. Some object (the sphere in this example) is scanned and the sampled data is used for the computer model reconstruction.

1.4. Sampling criteria

The triangle mesh obtained after the reconstruction from the set of input points will be just very close to the original surface of the real object because some information is lost during the process of digitization. For example, there is no way how to obtain an exact reconstruction for the surfaces with sharp edges because of the Nyquist criterion $f_{nyq} = 2f_{max}$, where f_{max} is the maximum frequency in a frequency spectrum of a function whose amplitude spectrum is finite. It means that the function can be exactly reconstructed if the sampling frequency is at least twice as large as the maximum frequency f_{max}. So for exact reconstruction of the edge it is necessary to have infinitely dense sampling.

The result of the reconstruction depends on the sampling density. There are two criteria specifying whether the sampling is sufficiently dense. One is based on the *local feature size* (*LFS*) and the other on the *sampling path.*

This sampling path criterion is based on the parameter ε, which denotes a radius of a sphere. We say the surface *S* is sampled with the sampling path ε if any sphere with the radius ε and centered on *S* contains at least one sampled point. Fig. 1.3 presents an example of ε sampling path. The surface containing both sharp edges and smooth parts is sampled according to the ε sampling path, which is visible on the circles with radius ε. It is shown that this kind of sampling produces uniformly sampled datasets without regards to the scanned object features so the flat parts of the surface are scanned with the same precision as the parts with small features, such as edges.

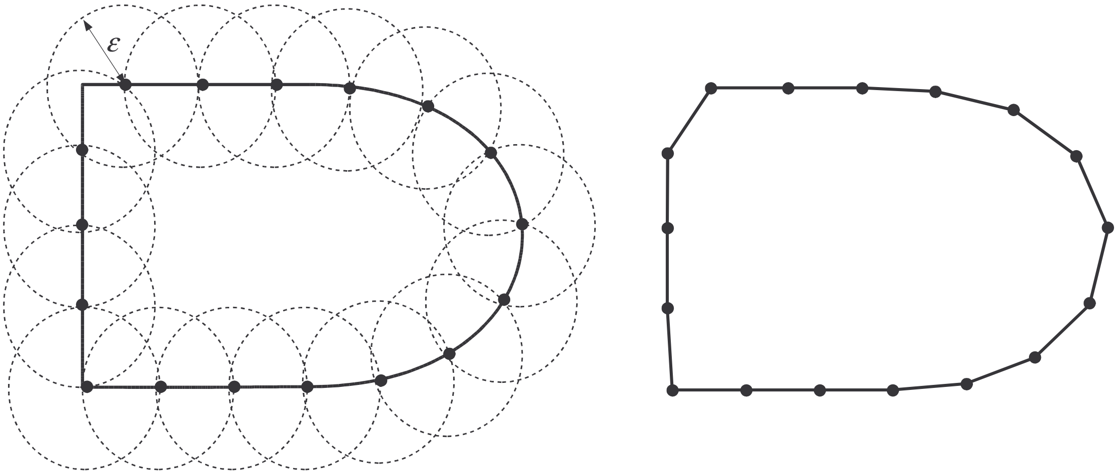


Fig. 1.3: The object and its sampling using the sampling path. The circles with the radius ε show the sampling path, the reconstruction is shown in the right part of the figure where a badly reconstructed corner appears.

Local feature size *LFS (s)* of the point $s \in S$ is a function that assigns to every point *s* a real value (*LFS* $(s \in S) : S \rightarrow R$) corresponding to the closest distance to the medial axis. Medial axis of *S* is defined as the closure of all points in E^3 which have more than one closest point on *S,* so the circles placed on the medial axis touch tangentially at least twice the surface *S*. Because there is no surface information in the sampled surface, it is impossible to compute the medial axis and get the correct *LFS*. Some of the Delaunay-based algorithms use an approximation of the medial axis, the poles (closer explained in Chapter 4).

Successful reconstruction depends on the ε-sampling [Amenta98b]. The point set $P \subset S$ is called ε-sampled, if every point $s \in S$ has a sample $p \in S$ within the distance of ε *LFS* (*s*). It is proved in [Amenta99] that for $\varepsilon < 0.06$ the reconstruction is homeomorphic to the surface *S*, however, in practice it is able to achieve a successful reconstruction even for $\varepsilon < 0.5$.

This sampling criterion works well and allows to scan parts of surface with small features more precisely (as the medial axis is close to the surface). In Fig. 1.4 the same part of surface as in Fig. 1.3 is shown and sampled according to the *LFS* criterion. For better clarity of the figure the variable ε is set to 1 (for successful reconstruction it should be less than 0.5), it means that, e.g., the closest point p_1 to the already sampled point *p* should lie in the maximum distance of $\varepsilon LFS\ (p) = LFS\ (p)$ from the point *p*. It is shown that flat and smooth parts of the surface are scanned with lower density, so the amount of points is less than using the previous criterion. On the other side, the places with sharp edges are sampled with higher density (ideally infinite exactly in the corner, practically the application has some limit and when the distance is under this limit, the sampling stops), so the details are better preserved.

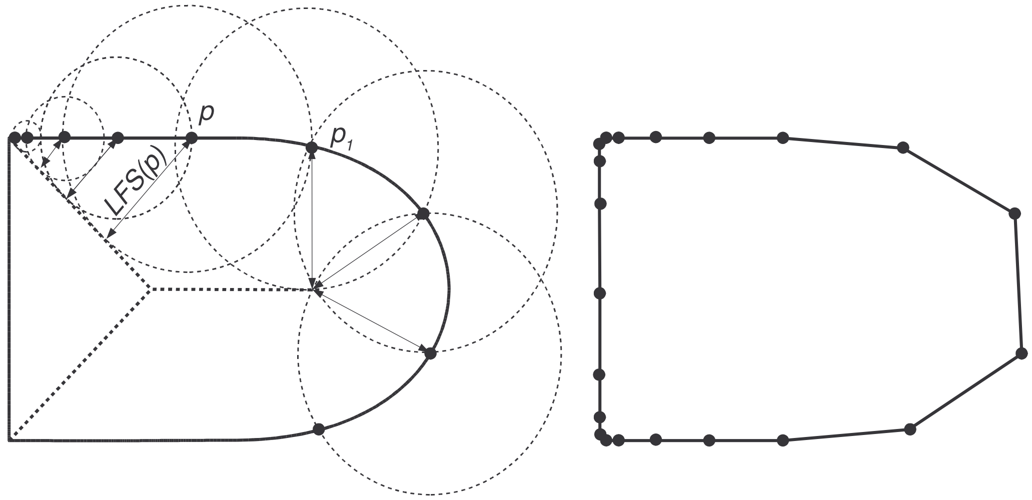


Fig. 1.4: The object and sampling with the local feature size criterion (only a part of the surface sampled is shown for better readability). The circles show the distance to the medial axis and where the closest point should lie. The arrows show the LFS of the current point. In the right part of the figure the reconstruction is presented, the parts with sharp corners are well reconstructed.

1.5. Brute force algorithm for surface reconstruction

Our task is to create the triangle mesh from the set of points. Denote N as the number of points. We can create $N(N-1)(N-2)/6$ different triangles (we have $N(N-1)/2$ different edges between N points, each edge can be connected with N-2 other points to create the triangles, therefore, we have $N(N-1)(N-2)/2$ possible triangles but each triangle is counted three times, so we divide the result by three). If we suppose that the surface is closed, its genus is 0 and it is orientable, then we can use the the Euler formula for computing the number of triangles in this surface. The Euler formula says for this kind of surface that

$$N-E+F=2 \tag{2}$$

where E is the number of edges and F is the number of faces, in our case the number of triangles, and N is the number of vertices. Any object that is homeomorphic to a sphere (see Chapter 2 for the topological terms) can be projected to the plane model and the projection does not change the number of triangles and edges. The projection can be done by stretching one of the object's triangle, the other faces will be projected into this triangle, see Fig. 1.5 for an example.

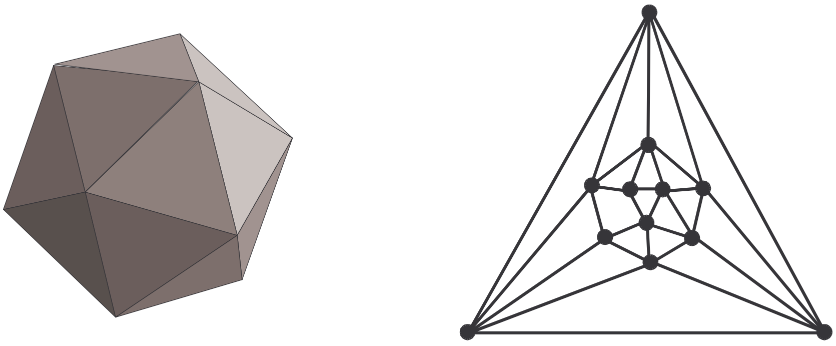


Fig. 1.5: The icosahedron and its planar model.

Then the number of edges is 3N-6 because this is the maximum number of edges in the planar triangulation (3N-3-E_{ch}, where E_{ch} is the number of edges in the convex hull, which is 3 in our case). Then the expected number of triangles is

$$N-3N+6+F=2 \rightarrow F=2N-4 \tag{3}$$

So, with a brute algorithm, we have to select from $N(N-1)(N-2)/6$ triangles the $2N-4$ surface triangles. The problem can be reformulated to: how many ways exist to select from the set of *n* triangles the set of *k* different triangles? This is a combinatoric problem, so there is

$$\binom{n}{k}=\frac{n!}{(n-k)!\,k!}=\frac{\frac{N(N-1)(N-2)}{6}!}{\left(\frac{N(N-1)(N-2)}{6}-2N+4\right)!\,(2N-4)!} \tag{4}$$

possible combinations, which is quite a big number. Fortunately, we can utilize the topological and geometrical properties of the surface, such as the mutually closest edges have to be connected by the edge or that the Delaunay tetrahedronization contains as a subgraph the surface, and it is possible to significantly reduce the set of possibilities.

If we restrict the set of possible surfaces by these assumptions, we can estimate that the reconstruction of surface from *N* points is a problem which is similar to the problem of the construction of Delaunay tetrahedronization. Therefore, the expected algorithm complexity would be O(*N* log*N*) because it is the expected time for the Delaunay tetrahedronization and the surface triangle selection from all faces in Delaunay tetrahedronization is a local problem around each point which can be done in a constant time per point.

2. Related terms

This chapter describes the most frequently used terms in this thesis and tries to explain them in more details for better clarity of the thesis. It starts with the definition of the useful topological terms, then it deals with the term of the medial axis. As we use in our approach a spatial subdivision based on the Delaunay triangulation/tetrahedronization, it will be explained next, followed by its dualization, by the Voronoi diagram.

2.1. Topology

As this thesis deals with the surface reconstruction, some space and metric of the space in which the reconstruction is processed need to defined. We assume the *Euclidean space* together with its metric. The general Euclidean space E^n is formed by the set of all ordered n-tuples (a_1, a_2, a_3, ..., a_{n-1}, a_n) where $a_i \in R$. In the Euclidean space the Euclidean metric is defined, the distance between two points x and y is:

$$L_2^n(x,y)=\sqrt{(x_1-y_1)^2+(x_2-y_2)^2+...+(x_{n-1}-y_{n-1})^2+(x_n-y_n)^2} \qquad (5)$$

Generally, the space of lower dimension E^{n-1} is a subset of E^n space, thus:

$$E^1 \subset E^2 \subset ... \subset E^{n-1} \subset E^n \qquad (6)$$

An *object* is defined as the set of points in a space. A *mapping* is a way how to associate points from one object to another object. A map $f: A \rightarrow B$ from A to B is a function f where for each point $a \in A$ is a unique $f(a) \in B$. A mapping is called an *injection* (see Fig. 2.1a), or the mapping *one-to-one*, if for each $f(x) = f(y) \Rightarrow x = y$, equivalently, $f(x) \neq f(y) \Rightarrow x \neq y$. If for each $b \in B$ $a \in A$ exists for which $b = f(a)$, then the mapping is called a *surjection* (see Fig. 2.1b), or the mapping *onto*. A mapping which satisfies the conditions of both mappings surjection and injection is called *bijection* (Fig. 2.1c), sometimes called *one-to-one correspondence*.

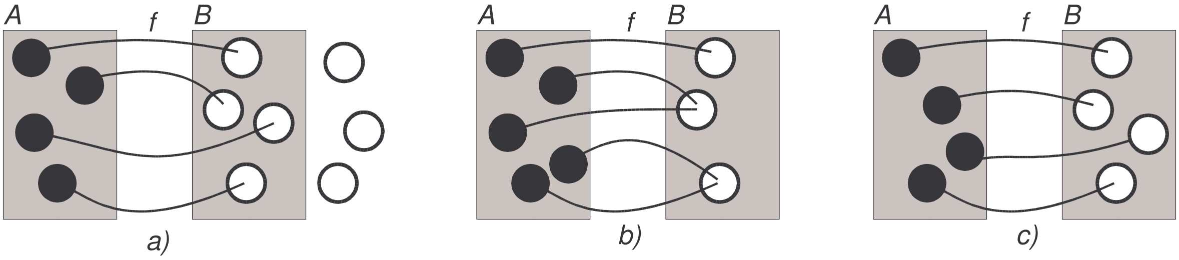


Fig. 2.1 The examples of mapping, a) an injection (one-to-one), b) a surjection (onto), c) a bijecton (ono-to-one and onto).

The mapping from one object X to the object Y is called *continuous*, when the consequence of points $x_i \in X$ converges to the point $x \in X$ and the consequence of points $y_i = f(x_i) \in Y$ converges to the points $y = f(x) \in Y$.

When we compare topologically two objects, we use the term *homeomorphism*. We say that two objects X and Y are *homeomorphic*, if there is a continuous bijection from X to Y and the inverse transformation is continuous, too. If two objects are homeomorphic, then they are *topologically equivalent*.

Three fundamental and magic words in the area of surface reconstruction are the terms *manifold, curve* and *surface*. But for closer definition we need to be more concrete and to define more terms. We start with the definition of intervals, there are four kinds of them, an *open interval* (in (7)), a *closed interval* (in (8)) and two *half-open* (or *half-closed*) intervals, *half-open from right* (in (9)) and *half-open from left* (in (10)).

$$(a, b) = \{x : a < x < b\} \tag{7}$$

$$[a, b] = \{x : a \leq x \leq b\} \tag{8}$$

$$[a, b) = \{x : a \leq x < b\} \tag{9}$$

$$(a, b] = \{x : a < x \leq b\} \tag{10}$$

An *open unit disk* is a set of all points (x_1, x$_2$) in a plane for which (11) states. This definition can be simply extended to a higher dimension than E^2 , then the term an *open unit ball* or an *open unit sphere* is used. Generally, for the E^n space it is defined as in (12). Similarly, a *closed unit sphere* (or ball, respectively) in E^n is the open unit sphere unified with its boundary, see (13).

$$x_1^2+x_2^2<1 \tag{11}$$

$$x_1^2+x_2^2+\ldots+x_{n-1}^2+x_n^2<1 \tag{12}$$

$$x_1^2+x_2^2+\ldots+x_{n-1}^2+x_n^2\leq 1 \tag{13}$$

We are almost done, the last term needed for the definition of a manifold is the term *neighbourhood*. Let *X* be an object, $p\in X$. A *basic neighbourhood* of the point *p* is a set of all points in *X* that lie strictly within the distance $\varepsilon\in R$ of *p*. By a neighbourhood we mean a subset of *X* which contains the basic neighbourhood. An example of the neighbourhood is presented in Fig. 2.2 where a sheet of glued paper is shown. The point p_1 has a neighbourhood homeomorphic to the planar open disk, the point p_2, which lies on the boundary of the paper, has a neighbourhood homeomorphic to the open half-disk. The point p_3 is problematic as its neighbourhood consists of three ”flaps” joined along a single edge.

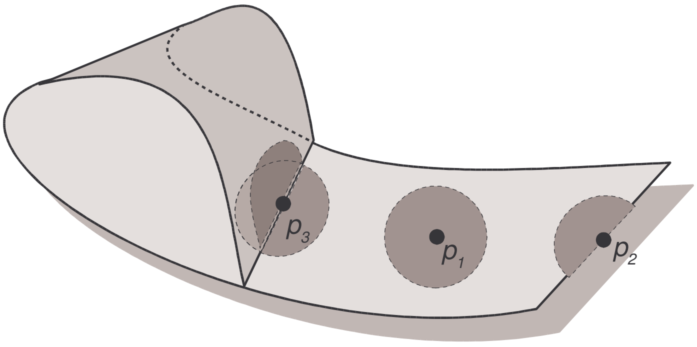


Fig. 2.2: Three kinds of neighbourhood.

Manifolds form a class of topological objects that includes *knots* (specific examples of one-dimensional manifolds) and *surfaces* (specific examples of two-dimensional manifolds). A set of points in a space is a *1-dimensional manifold* if each of its points has a neighbourhood homeomorphic to an open interval of the real line. Likewise, the set of points in a space forms a *2-dimensional manifold* if each its point has a neighbourhood homeomorphic to the open planar disk. For example, the E^2 space is a 2-manifold, a sphere or a torus are subsets of E^3 space but they 2-manifolds, too, such as the Klein bottle which is the subset of E^4 space.

We need to work with the objects with boundaries, too, so we extend a little the definition of a manifold to a *manifold with boundaries*. Such a manifold is the object whose neighbourhood is homeomorphic to an open planar disk while the boundary is homeomorphic to a half-open planar disk. In the next chapters when the term manifold will be used, we suppose that it is a 2-manifold.

The objects in the space have some properties, four fundamental of them will be presented now. The object *X* in the space is:

- *connected*, if *X* consists of just one component
- *bounded*, if there is an upper bound for the distances between pairs of *X* points
- *closed*, if *X* contains all points in the space that are limit of the consequence of *X* points
- *compact*, if it is bounded and closed

Thus, the *curve* is a *connected 1-manifold*, the surface is a *connected 2-manifold*.

2.2. Medial axis

The medial axis is a geometrical structure associated to any bounded open set in E^n. The medial axis of some object is the set of points which have at least two closest points on the boundary of the object, so the circle which lies on the medial axis touches the object at least twice. For the object with sharp edges or corners, the medial axis touches the object in such places. Fig. 2.3 shows two objects with their medial axes.

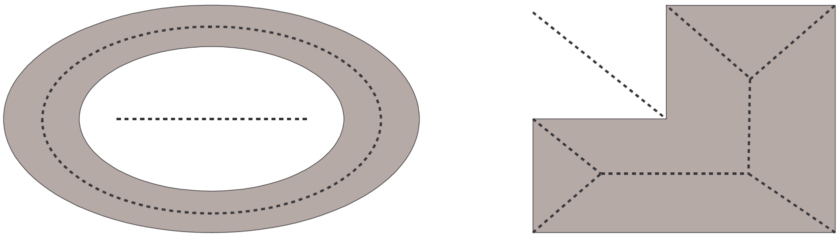


Fig. 2.3: The medial axes (dashed line) of two objects.

2.3. Delaunay tetrahedronization

A finite point set $P \subseteq E^3$ defines a special triangulation known as the *Delaunay tetrahedronization* (*Delaunay triangulation* in E^2). The triangulation has the name after the

Russian mathematician and geometer Boris Delaunay who first introduced the concept of this triangulation in his paper [Delaunay34]. Assuming a general position of points, this triangulation is unique and defines the space decomposition of the set P into tetrahedra, where all points from the set P lie in the convex hull.

The Delaunay tetrahedronization of P is the simplicial complex defined by the tetrahedra. It consists of elements F_k (k = {3, 2, 1, 0}): the tetrahedra F_3, the triangles F_2, the edges F_1 and the vertices F_0. The intersection of all open balls b around each tetrahedra and the set P must be zero set, $\forall b \cap P = \emptyset$. It has following properties:

- if there are not more than five points lying on the sphere, then this tetrahedronization is unique for each set P. Otherwise, there are more Delaunay tetrahedronizations differing only in these singular places,
- the boundary of tetrahedronization forms the convex hull,
- the Delaunay tetrahedronization minimizes the maximum radius of the simplex enclosing sphere,

The subsets of edges in the Delaunay triangulation belong to some graphs (the relation is shown in (14), some examples in Fig. 2.4), there are four best known, sometimes they are used in reconstruction approaches, too.

$$NNG \subseteq EMST \subseteq RNG \subseteq GBG \subseteq DT \tag{14}$$

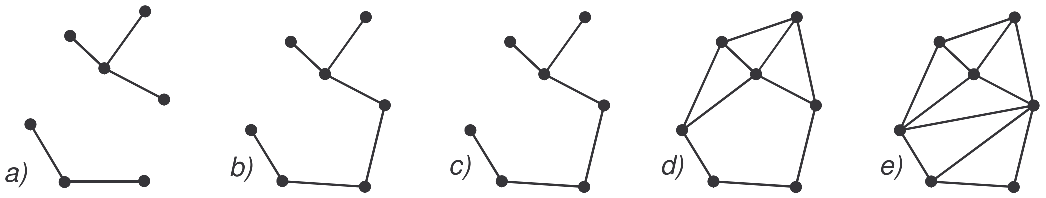


Fig. 2.4: The comparison of the subgraphs of Delaunay triangulation, a) NNG, b) EMST, c) RNG, d) GBG, e) DT.

The nearest neighbours graph (*NNG*) is the simplest one. Two points are connected by an edge in this graph if they are mutually nearest neighbours. The euclidean minimum spanning tree (*EMST*) is another subset of the Delaunay triangulation, a tree with the shortest sum of edge lenghts.

The relative neighbourhood graph (*RNG*) is another subgraph, two points are connected by an edge if the circle centred at any of these points with the radius of the points distance does not contain any other point. A little complicated is the Gabriel graph

(*GBG*), where two points are connected in the case that the circumscribed circle of the edge with the shortest radius does not contain any other point.

If we look to the real example of the subgraphs in Fig. 2.5, we can see why there are sometimes used for the reconstruction. The grey edges present the Delaunay triangulation of the points printed in white, the black edges than show the subgraphs of the triangulation. The reconstructed surface contain all edges of the *NNG* and many edges of the *RNG* and *GBG*.

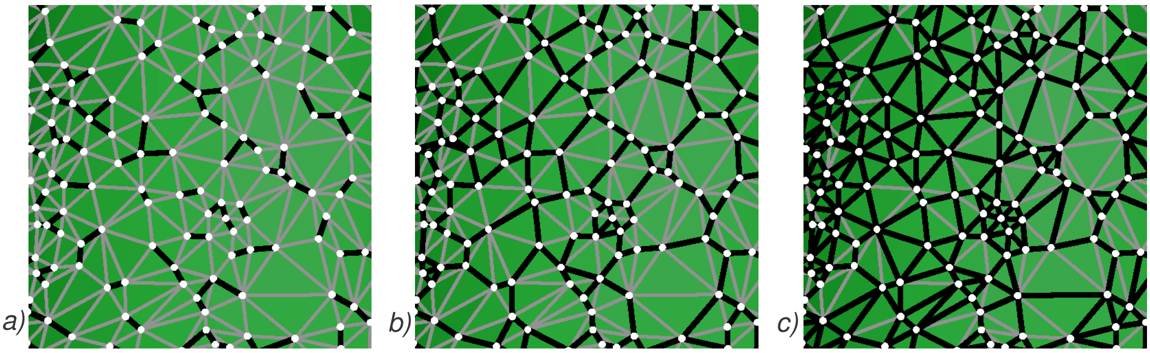


Fig. 2.5: The comparison of some Delaunay triangulation subgraphs, a) the nearest neighbour graph, b) the relative neighbourhood graph, c) the Gabriel graph.

The Delaunay tetrahedronization is often used in space decomposition algorithms and in surface reconstruction algorithms. Due to the behavior and properties of the triangulation it can be proved that the reconstruction of the point set *P* sampled from the surface *S* belongs to the subgraph of the *DT*(*P*).

2.4. Voronoi diagram

The concept of *Voronoi diagrams* was first introduced by G. Voronoi in [Voronoi07]. Voronoi diagram is a space partitioning which decomposes the space into the convex polyhedral cells called a *Voronoi cells* (this term is used in this thesis), *Dirichlet regions* or *Thiessen polytopes*. For a point $p \in P$ the *Voronoi cell* $V(p)$ is the set of points $x \in R^3$ for which the euclidean distance between *x* and *p* is less or equal to the distance between *x* and any other point of *P*:

$$V(p)=V_p=\{\forall q \in P, x \in E^3 : |p-x| \leq |q-x|\} \tag{15}$$

Each cell forms a convex polyhedron and the union of all cells (one for each point of the set *P*) is the Voronoi diagram of the set *P*. The face of the Voronoi cell is a geometric place for which it holds that the distance of two given points to the face is equidistant. Similarly, the edge of the Voronoi cell is the place where three given points have the same distance and the vertices are the places where more than three points have an equal distance. In contrast to the Delaunay tetrahedronization which is closed in the convex hull, the cells lying on the convex hull are open.

An important property of the Voronoi diagram and Delaunay tetrahedronization is that they are mutually dual, see Fig. 2.6 for a planar example. The vertices of the Voronoi diagram represent the centers of circumspheres of the Delaunay tetrahedra. When there is an edge in the Voronoi diagram, then there exists a face (triangle) in the Delaunay tetrahedronization.

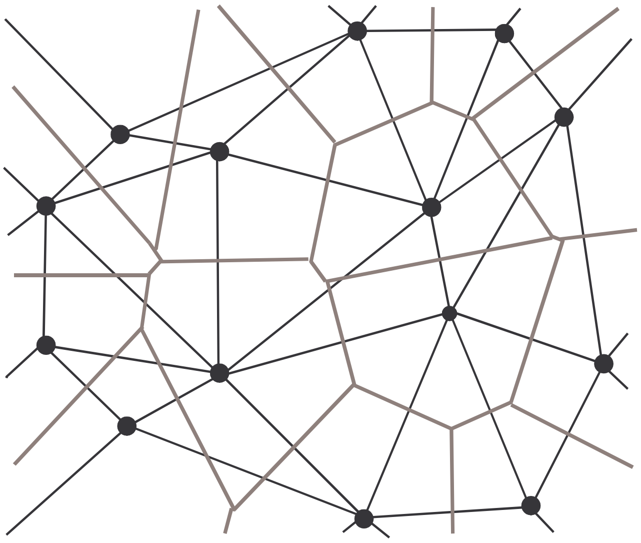


Fig. 2.6: A planar example of Voronoi diagram in gray and Delaunay triangulation in black.

3. State of the art

Many methods for solving the problem of surface reconstruction were developed during recent years. The development started more than twenty years ago by the curve reconstruction, the E^2 version of our problem. Many methods exist with strong theoretical background, e.g. [Amenta98a, Dey99, Dey00, Atalli97].

The E^3 problem has been addressed by many researchers in computer graphics and computer vision. We can divide the developed algorithms using many criteria but two of them are the most used. The former criterion takes into acount the output surface from the reconstruction. Thus the reconstructed surface can

- approximate
- interpolate

the input dataset. The algorithms which approximate the surface are more robust to the errors in the input data, such as noise or incorrect sampling, but the surface is just an estimation of the original because the surface does not go through the sampled points. This effect is shown in Chapter 11 where a comparison of several reconstruction algorithms is presented, sometimes the surface is far away from the sampled points. By contrast, the interpolation algorithms require more accurate sampling and are more susceptible to the sampling errors, but the output surface passes through the input points. Thus the user can choose what to prefer, whether he wants the surface which is as close as possible to the real object but with possible errors or less precise surface but with low number of errors.

The latter criterion takes into acount the ways how the surface is created. The algorithms can be divided into following groups (the division is not strict, many algorithms use a combination of more approaches):

- warping
- incremental surface construction
- distance function methods
- spatial subdivision
  - volumetric methods
  - surface methods
- other methods

3.1. Warping

Warping works on the basic idea that we deform some starting surface to the surface that forms the object. We can imagine the process of warping on some object, which is hidden in a big ball filled with the air. When we start to deflate the air from the ball, the size of the ball decreases and at the end of the deflating process there will be just an empty ball copying the surface of the object. Geometrically, let us have as the starting surface some triangle mesh around the sample points. For all vertices of the triangle mesh we find their correspondency with the sampled points and we move them to these positions. The consequence is that the starting triangle mesh deforms to the mesh which is close to the original surface, this is used in the Miller's et al. approach [Miller91].

The idea of warping is relatively old and basic methods are, e.g., Muraki “blobby model” for 2.5 mesh approximation [Muraki91] or deformable superquadrics by Terzopoulos and Metaxas [Terzopoulos91a, Terzopoulos88].

Other approach, physically oriented, was introduced by Szeliski and Tonnesen [Szeliski92]. The use oriented particles connected by springs, whereas every particle has some parameters whose values are updated during the modelling simulation. Every sample point has a particle with corresponding parameters and the surface is created by an interaction modelling between particles (attraction x repulsion).

Some methods use the neural networks for the reconstruction. One of these methods was presented by Baader and Hirzinger in [Baader93, Baader94]. Their approach uses the Kohonen neural network (single-layered neural network with forward propagation and learning without a teacher) for a reconstruction of 2.5D datasets, where each input point is represented by a neuron. More robust approach based on the same kind of the neural network and which is not limited to 2.5 meshes was presented by Yu in [Yu00].

3.2. Incremental surface reconstruction

Boissonat [Boissonat84] presented another approach how to obtain the model from a point cloud. It begins on the shortest edge from all edges between points and incrementally appends points to create a final triangle mesh.

Mencl and Müller [Mencl95, Mencl98b] developed a similar algorithm. The algorithm creates an extended minimum spanning tree and extends it to surface description graph by connecting the points under specific rules. Then it identifies and extracts typical features of the surface, such as rings or edges, and in the final step it extracts the surface using these properties.

3.3. Distance function methods

Other algorithms (sometimes called volumetric methods) are based on a distance function. This function describes the shortest distance from the point to the surface. For closed surfaces, the value of the function is negative or positive depending on whether the point is inside or outside the object. This function is computed for each point using the tangent plane. The plane can be estimated from *k* nearest neighbours (points) by the least square approximation.

Hoppe et al. [Hoppe94, Hoppe92] gave an algorithm, where the surface is represented by the zero set of a signed distance function. The algorithm estimates the normal vectors in the first step using *k* nearest neighbours. Then it creates the Riemannian graph and extracts EMST from this graph. The EMST is then used for the normal vectors orientation, because all the vectors must be oriented out or into the object volume. Finally, the distance function is computed for each point and the surface, which has the zero distance function value, is extracted by the Marching cubes algorithm [Lorensen87].

The next approach presented by Bittar et al. uses the space voxelization of the input dataset [Bittar95]. The voxels are processed from outside and the process stopped on voxels containing points. The distance function is then propagated through the voxels inside the volume and the voxels, where the function gains the maximum, are taken as medial axis. The medial axis helps the algorithm to reconstruct the surface using the distribution of spheres on it.

Curless and Levoy [Curless96] gave a really effective algorithm which represents the signed distance function on a voxel grid and is able to reconstruct eventual holes by a post-processing step. Their algorithm is designed for very large data and was used for statues reconstruction in the Michelangelo project [Michelangelo00].

Roth and Wibowoo show simple algorithm based on the propagation of the distance function values through the voxel grid [Roth97]. This algorithm estimates the normal vectors in points and computes the distance function using the distance to the nearest points.

3.4. Spatial subdivision

The basic property of the methods based on spatial subdivision is that the boundary hull (convex hull, box around points, etc.) of the point input set is divided into independent areas. A typical example is the division by a regular grid, adaptive by an octree or an irregular tetrahedronization.

The methods based on spatial subdivision can be divided into two subgroups. The former are surface based and these methods work on the principle that they look for the simplice intersected by the surface. The latter is volume based and these methods exploit the volume of the surface created by the spatial subdivision for surface extraction.

One of the first algorithm which uses the Delaunay tetrahedronization was the algorithm developed by Boissonat [Boissonat84]. It removes recursivelly the tetrahedra from the whole tetrahedronization till all the points lye on the surface. The volume of the surface is then composed by the rest of tetrahedra. This algorithm was extended in Isselhard's algorithm [Isselhard97] which extends its capabilities by special function for tetrahedra evaluation.

Algorri and Schmitt [Algorri96] gave an effective algorithm in which the space is subdivided by a regular grid (into voxels). In the next steps those voxels are chosen which contain points from input set and the surface is extracted.

Veltkamp has introduced a new general geometric structure called γ-graph [Veltkamp92]. Initially, the γ-graph coincides with the convex hull of the data points and is constricted until the boundary of the γ-graph is a closed surface – the tetrahedra which have boundary faces are deleted.

Edelsbrunner [Edelsbrunner92] developed the program for uniform sample set surface reconstruction using the algorithm called α-shape. The decomposition of the input set is achieved by the Delaunay tetrahedronization. Next step is deleting the simplices whose circumsphere radius is bigger than the radius of a so called α-ball (the sphere with a radius α, which is the input parameter of the method) and, finally, extraction is done. The problem of the approach is that it is sensitive to sampling density changes, so the next version of the algorithm, weighted α-shape by Edelsbrunner and Mücke [Edelsbrunner94], was developed to bypass this limitation.

Bernardini and Bajaj [Bernardini97] developed an algorithm which gets the surface subcomplex of the Delaunay tetrahedronization. This algorithm extends the idea of α-shapes and it use the binary search on the parameter α to find this subcomplex. Smaller concave features not captured by the α–shape are found using heuristic. The surface is then used to define a signed distance function and a C^1 piecewise polynomial function is then adaptively fitted to the signed distance field.

A paper by Bernardini et al. [Bernardini99] describes a ball pivoting algorithm to interpolate a set of points not based on the Delaunay sculpturing, but extending the surface (Delaunay triangles initially) like in the surface growing methods. A ball of fixed radius (approximately the distance between two sampled points) is placed to three points which form the initial triangle. The edges are put to the queue and the ball pivots through all edges in the queue to obtain new surface triangles. For places of undersampling it is possible to restart the algorithm with a bigger ball radius. The advantage of the algorithm is that it is very fast and it can handle millions of points.

Amenta et al. introduced a concept of CRUST, it has two versions, two-pass [Amenta98b] and one-pass [Amenta99, Amenta00]. The two-pass version creates the subcomplex of Delaunay tetrahedronization $S \cup P$, where P is the point cloud and S is the set of poles taken from the Voronoi diagram. The surface triangles are formed just with the triangles whose vertices belong to the input set P. The one-pass version can choose the surface triangles from the first tetrahedronization using a little different approach than in the two-pass version. Dey et al. extended the ideas of Amenta and gave an effective COCONE algorithm. The basic idea is presented in [Amenta00]. Other papers presented by Dey et al. introduced the way how to handle large data [Dey01b], which is the common problem of Delaunay based algorithms, and what to do with boundaries [Dey01c], undersampling and oversampling [Dey01a]. These ideas are based on the observation that the places with point density changes can be detected using shape of the Voronoi cells in these places. The authors gave an algorithm for a watertight surface reconstruction, Amenta et al. the PowerCRUST based on medial axis transformation [Amenta01] and Dey et al. the TightCOCONE based on tetrahedra removal [Dey03].

Boissonat and Cazals presented an algorithm [Boissonat00] which uses the natural neighbour interpolation (an extension of Voronoi diagram) and which is able to run in any dimension. The surface is defined as the zero set of pseudodistance function and the surface can be meshed to satisfy an user error-bound.

Cohen-Steiner and Da showed the algorithm [Cohen03] which recursively appends triangles from Delaunay tetrahedronization to the already reconstructed surface δS. The surface is extended on the boundary edges of δS by one of non-processed triangles on the edge, the triangle is chosen according its evaluation.

As the Delaunay tetrahedronizaton contains other subcomplexes, they can be used for the task of surface reconstruction, too. For example, the Gabriel graph (*GBG*) is used in some algorithms because many edges of *GBG* are equal to surface edges of the reconstructed surface. This approach is used by Boyer and Petitjean in theirs algorithm [Boyer00] where the regular interpolant set, which is created using the geometrical properties of the *GBG*, was introduced. The Spiralling edge algorithm by Crossno and Angel [Crossno99] supposes the normal vectors at points and it creates the *GBG* locally using *k* nearest neighbours. This graph is used for the triangle fan computation around the points, the fans are then glued together. Giesen and John presented an algorithm [Giesen02] which computes the distance function from Voronoi diagram. They observed that the critical points of the distance function correspond with the edges of *GBG*. In the next step the function values are transformed to a dynamical system and flow complex, which is then reduced to the surface.

3.5. Other methods

Some methods use approaches known from other areas of computer graphics. For example, the approach presented by Gopi and Krishnan uses a projection based technique for the reconstruction [Gopi00]. Another useful approach uses the radial base functions. These functions are circularly symmetric and centered around points and they may be used to interpolate the surface, which can be meshed by an implicit polygonizer. This approaches are used in papers, e.g., by Morse et al. [Morse01], Carr et al. [Carr01] and many others. The functional surface description was presented in other papers, too, the paper by Gross et al. [Gross96] shows the possibility how to reconstruct the surface using the wavelet transformation followed by tresholding. Then the inverse transformation is applied and the surface is approximated using quadtree, which can control the level of detail of the output surface. Pauly and Gross presented the paper which uses the Fourier transformation for the surface reconstruction [Pauly01]. The big advantage of functional surface description is the possibility of a functional analysis utilization, such as low pass filtering or resampling.

The goal of this chapter was to give the brief summary of the existing algorithms. It is not possible to mention all of them, many other methods were developed and many other, which are not primary used for the task of surface reconstruction, may be adapted. We recommend papers e.g. by Mencl and Müller [Mencl98a], Fabio [Fabio03], Bernardini and Rushmeier [Bernardini00], Kobbelt et al. [Kobbelt00] or our technical report [Varnuska04] for other summary of the reconstruction algorithms.

4. CRUST algorithm

The task of surface reconstruction can be solved in two ways. The former is to develop a new algorithm, but this way is more complicated than the latter, to use some existing algorithm and to improve it. The reason is simple, many algorithms based on various geometrical properties have been developed during last twenty years and almost no new areas remain which can be used for development of some newer approach. The quality of developed approaches speaks moreover for the latter way, many of them produce surfaces of good quality for the points satisfying some properties and the extension of these algorithms is more simple then the development of newer algorithm.

We have chosen the CRUST algorithm to solve the surface reconstruction problem. The method is not built on heuristic and the success of the reconstruction is guaranteed by a theoretical background. It is a relatively new algorithm and its principle is relatively simple for understanding. It belongs to the group of methods built on spatial subdivision, which is done using Delaunay tetrahedronization. The usage of the *DT* was the second reason why to use this algorithm because our graphical group has a fast and efficient code for the *DT* computing developed by Ivana Kolingerová and Josef Kohout.

The algorithm is built on the observation that the surface triangles are contained in the *DT* and they form its subgraph. To select the surface triangles we can utilize the geometrical properties of the dual Voronoi diagram.

The E^2 version of the algorithm was first introduced in [Amenta98a] and it is derived from the Attali's normalized mesh algorithm [Attali97]. The name CRUST means the set of edges selected from the triangulation forming the curve interpolating the points set. The E^3 version [Amenta98b] is built on the same principle. The disadvantage of these algorithms is a double use of the *DT*, where the second pass uses about three times more points than the initial size of the input set. It limits the speed of the reconstruction and mainly the maximum size of the input point set. This limitation was overcome by the one-pass version in [Amenta00] which is able to select the surface triangles from the first *DT*.

The basic concept of the algorithm is simple. The first step is the computation of the Delaunay triangulation (or tetrahedronization in E^3). By the dualization, the Voronoi diagram of the point set is obtained. Information from the Voronoi diagram is used for the surface triangles selection from Delaunay triangulation. We call this set a primary surface and it has not to be a manifold, so an extraction manifold step is necessary.

The CRUST uses the sampling criterion built on the local feature size. That is the reason why the algorithm has a problem with parts of the surface containing sharp edges, outliers, boundaries or noise because the medial axis touches or is very close to the surface in these places and the LFS is bigger than the computed theoretical limit. On the other side, due to the local feature size criterion it has no problem with the places of varying points density and the surface can be sampled according to its features.

4.1. The poles

The geometrical properties of the Voronoi diagram can be utilized using the concept of poles. They are formed by the vertices of the Voronoi diagram (at least three points from the input set have an equidistant distance to the Voronoi vertices). The positive pole *p+* is the farthest Voronoi vertex (*VV*) of the Voronoi cell of some point *p*, the negative pole *p-* is the farthest *VV* on the "other side" of the plane containing the point *p* with the normal vector (*p+, p*), so the dot product of the vectors (*p-, p*) and (*p+, p*) is negative.

For a successfully sampled and smooth surface it holds that all Voronoi cells are thin and long, so the poles lay on or near the medial axis and vectors to the poles approximate the normal vectors. The algorithm versions differ especially in the meaning what poles are. The two-pass algorithm takes the poles as an approximation of the medial axis while the one-pass version takes the vectors from the point to the poles as an approximation of the normal vectors.

The example of the poles in E^2 is presented in Fig. 4.1a). The black points are the points from some input set and the Voronoi diagram (*VD*) of the given set is shown. The Voronoi cell related to the point *p* is highlighted with its Voronoi vertices painted as white circles. The farthest Voronoi vertex from the point *p* is marked as *p+* and it presents the positive pole, the arrow to this point is the estimated normal vector of the curve. The dashed line is perpendicular to the normal vector and the farthest Voronoi vertex on the "other side" is marked as the negative pole *p-*. Similar relations hold for the E^3 example in Fig. 4.1b), where the Voronoi cell for the point *p* is painted.

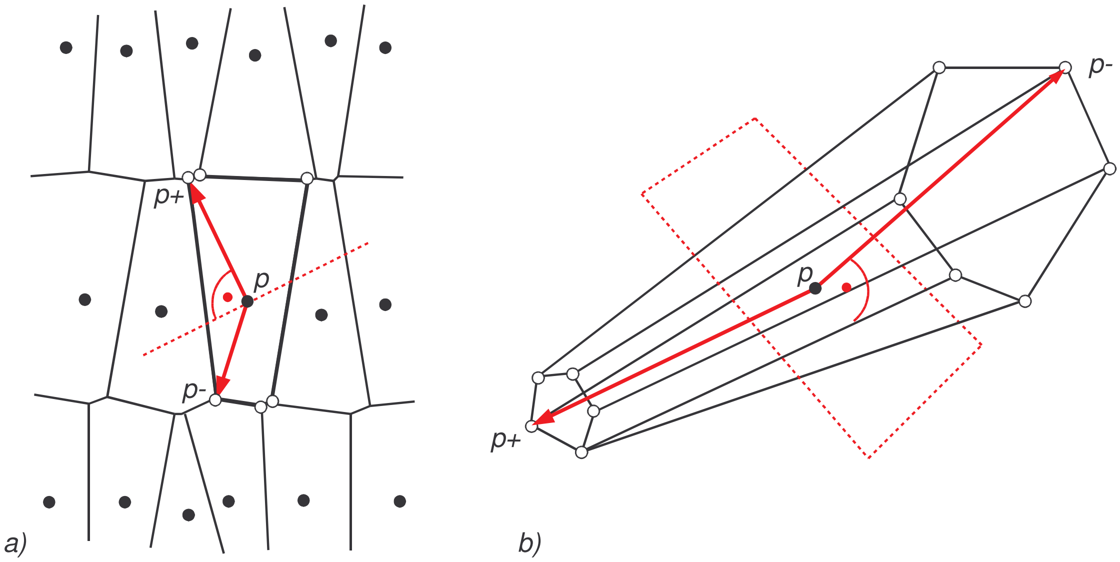


Fig. 4.1: The point p, the Voronoi cell and Voronoi vertices around it in E^2 and E^3. The vertex p+ is the positive pole, because it is the farthest vertex from the point p, the vertex p- is the negative pole, it is the farthest vertex on the "other side".

4.2. E^2 two-pass CRUST

The two-pass version of the algorithm comes out of the observation that the medial axis of the curve separates the opposite sides of the curve. Fig. 4.2 presents an example of the E^2 CRUST algorithm. Fig. 4.2a) shows the input point cloud sampled from some curve together with the Delaunay triangulation. Let us denote the set of Voronoi vertices as *V* (centers of circumcircles of the Delaunay triangles, see Fig. 4.2b). We can take *VD* as an approximation of the medial axis of the curve, thus *VV* lie on or near the medial axis. The *VV* of the input points approximates the medial axis.

Next step is a union of the input points *P* and the set of Voronoi vertices *V*, so that $U = P \cup V$. When we compute now the second *DT* (*U*), we get a triangulation (see Fig. 4.2c), where each triangle contains points from the set *V* and *P*. Just those edges, whose vertices belong to the input points set *P,* form the reconstructed curve (highlighted edges in Fig. 4.2c).

The approximation of the medial axis formed by the Voronoi vertices included to the second *DT* separates the curve edges from other edges. We obtain a curve approximation of *P*. The set of these edges, called crust, is a subset of both *DT*. This step is called Voronoi filtering and for sufficiently dense sampling it works perfectly.

In the box above the figure Fig. 4.2c) a problematic place after the reconstruction is also shown. The reconstructed curve in this place does not form a manifold because two points here have more than two coincident edges. The consequence of this is that the curve/surface extraction step has to follow.

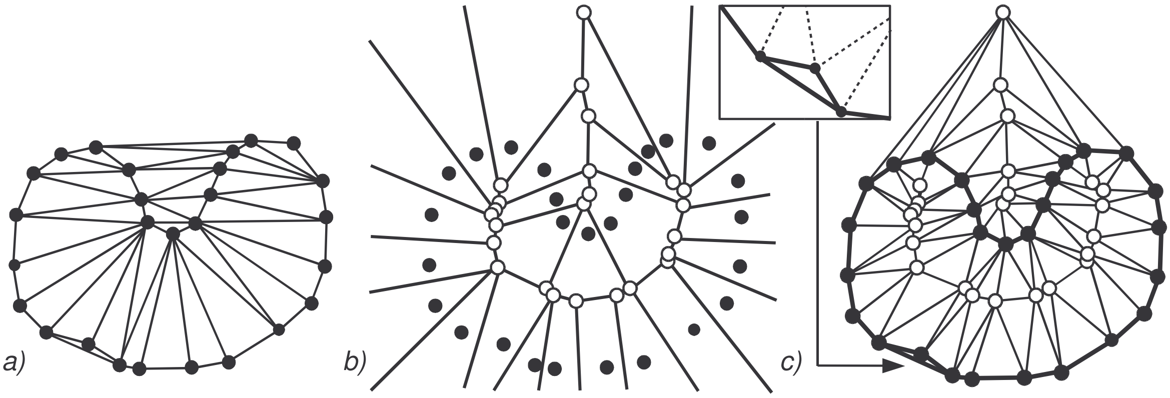


Fig. 4.2: An example of the two-pass curve reconstruction, a) the input point cloud with the Delaunay triangulation, b) the Voronoi diagram with highlighted Voronoi vertices. When we unify the points and Voronoi vertices and triangulate the set, we get the triangulation c) where curve edges (highlighted) incident only with the original points, the box above shows the detail of the problematic area.

4.3. E^3 two-pass CRUST

One of the advantages of the CRUST algorithm is that it can be simply extended to E^3. The problem is that the set of Voronoi vertices is huge and the tetrahedronization of the $P \cup V$ is almost impossible due to the big amount of data. For example, the dataset containing 10k points has around 64k of *VV*, the dataset of 35k points has 246k points, so the growth of *VV* amount is big. The second problem is that not all *VV* lie near or on the medial axis, on the other hand, we can still find many points for which this condition holds (see Fig. 4.3).

When we look at the following figure (Fig. 4.4), we can see two parts of surface (S_1 and S_2). The black point *p* represents the point for which the Voronoi cell is drawn, the other dots, painted in white, are other points sampled from the surface. It is shown that not all of Voronoi vertices lie on the medial axis although it holds for many of them.

The cell related to the point *p* is almost perpendicular to the surface and thin. The normal vector of the surface in the point *p* can be estimated using the positive pole *p*+. The observation that many Voronoi vertices lie on the medial axis leads to the following two-pass algorithm improvement. We compute the union of the input points, positive and negative poles $U = P \cup V+ \cup V-$ (instead of all *VV*), where *V*+ and *V*- are the sets of positive and negatives poles, respectively, and then we compute the second Delaunay tetrahedronization. Only those triangles, whose vertices are from the input point set *P*, belong to the surface triangles, to the set of crust.

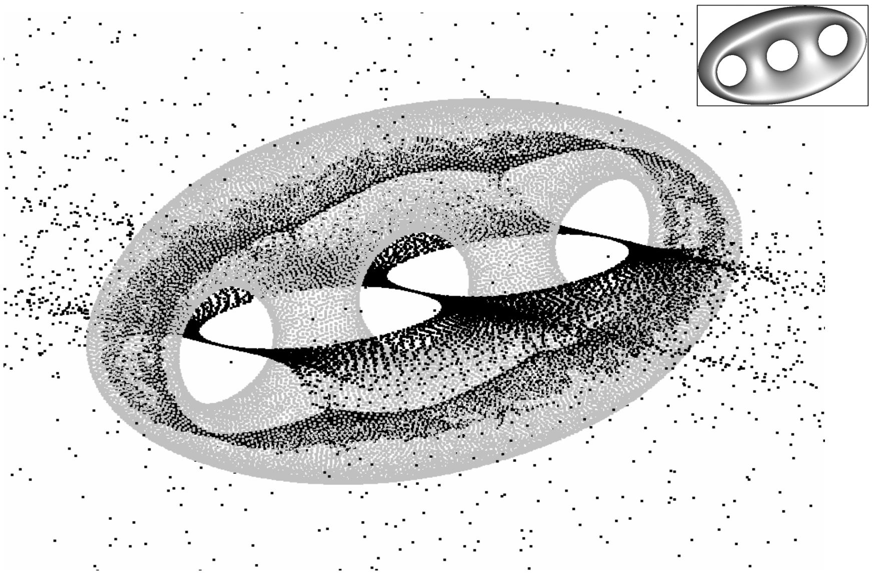


Fig. 4.3: The grey points of some object and the poles computed from the Voronoi diagram (positive and negative poles) in black. Many of them lie on or near the medial axis but some of them are far away..

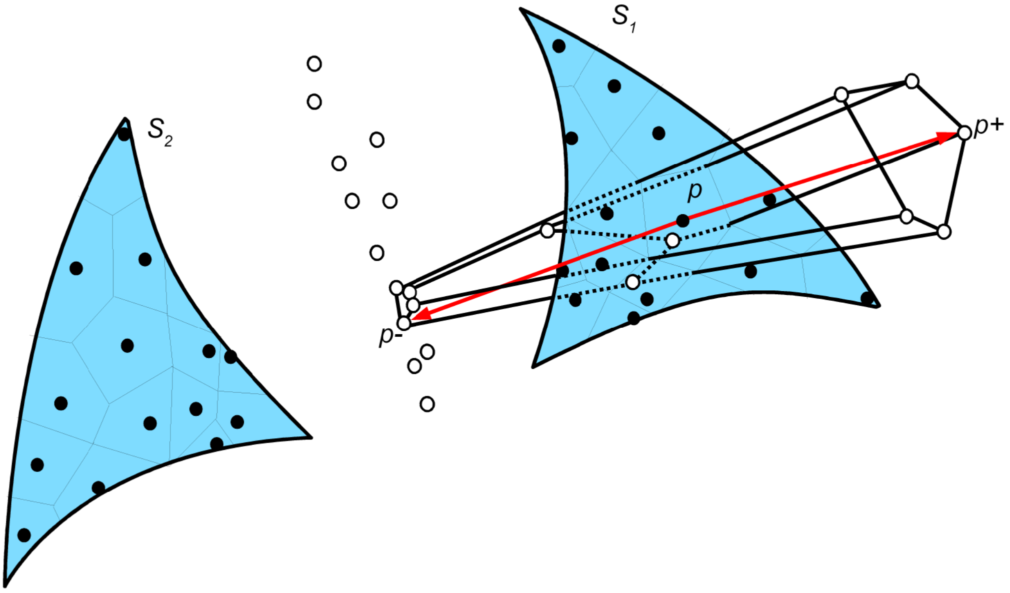


Fig. 4.4: The surface with two parts S_1 and S_2 with the Voronoi cell around the point p and the medial axis created by other Voronoi vertices (white points). The Voronoi vertices p+ and p- are positive and negative poles of the Voronoi cell.

This improvement enables to use the two-pass algorithm in E^3 (due to a smaller amount of *VV*), we need approximately (points lying on the convex hull have only one pole) three times more points for the second tetrahedronization than for the first tetrahedronization. A binding with the theory is provided by the following theorem

[Amenta98b] (let us remind that the closest sampled point of some point p must lie in ε *LFS* (p) distance to p):

> *Let us denote P a sample of the smooth surface S, where $\varepsilon < 0.06$. Then the crust consists of the triangle set which forms the mesh topologically equivalent to S. Every point from crust lies within the distance 5ε *d(p) of some point p on S, where d(p) is the distance from p to the medial axis.*

The authors of the original algorithm tried to use a little different approach for the reconstruction. Instead of using the negative poles, the second farthest Voronoi vertices are used and better results were obtained for some data. But it is not theoretically correct, so the result of the Amenta's testing was that better results can be only a good luck. During the testing of the algorithm we also tried this approach. Our impression was the same as Amenta's, the reconstruction was correct and sometimes better for some data but other data were reconstructed incorrectly. The main reason of the incorrect reconstruction was that more triangles were marked as surface and the step of the manifold extraction failed due to a big amount of overlapping triangles.

4.4. ϵ^3 one-pass version

Even if we limit the use of Voronoi vertices only to poles, the second Delaunay tetrahedronization consumes much time and memory. It is proved that the surface is contained in the Delaunay tetrahedronization as a subgraph, so there has to be some way how to obtain the reconstruction without the second *DT*.

When we look closer to the shape of the Voronoi cell, we can see something similar for all cells. For a sufficiently sampled smooth surface the cells have to be long, thin and perpendicular to the original surface so this assumption allows to approximate the normal vectors using the positive poles. A typical example is in Fig. 4.5a) where the Voronoi cell of some point in the reconstructed surface is shown. Fig. 4.5b) presents an example of a Voronoi cell in the place where two surfaces are close together.

The cell in this place is not so much long and thin but the shape is still good enough for the normal vectors computation. We tried to use the same, uniformly sampled, model as in the previous figure to simulate the influence of points removal on the shape of the Voronoi cell. In Fig. 4.6a) the model reconstruction and Voronoi cell of point p is shown. Then we removed all neighbours of the point p (white points around the point p) and made the reconstruction of such a dataset, the result is shown in Fig. 4.6b). Even if the data was locally undersampled, the reconstruction was correct. The reason is again in the shape of the Voronoi cell whose shape was not too much changed after points removal.

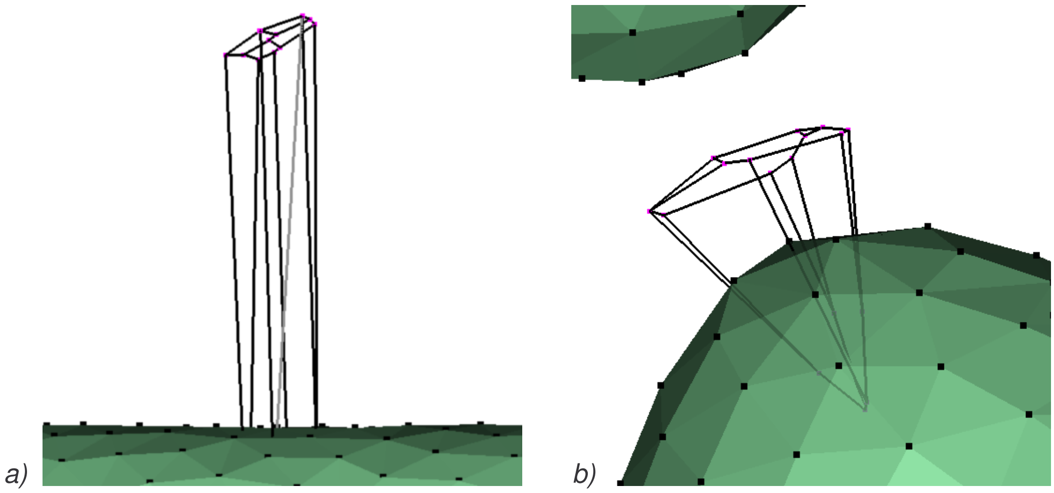


Fig. 4.5: Examples of the Voronoi cells on the reconstructed surface, a) the Voronoi cell of some point in a flat part of some surface (the gray line presents the normal vector estimated using the positive pole), b) the Voronoi cell of some point in the place where two surfaces are close (the normal vector is not visible, it directs down to the sharp edge of the Voronoi cell).

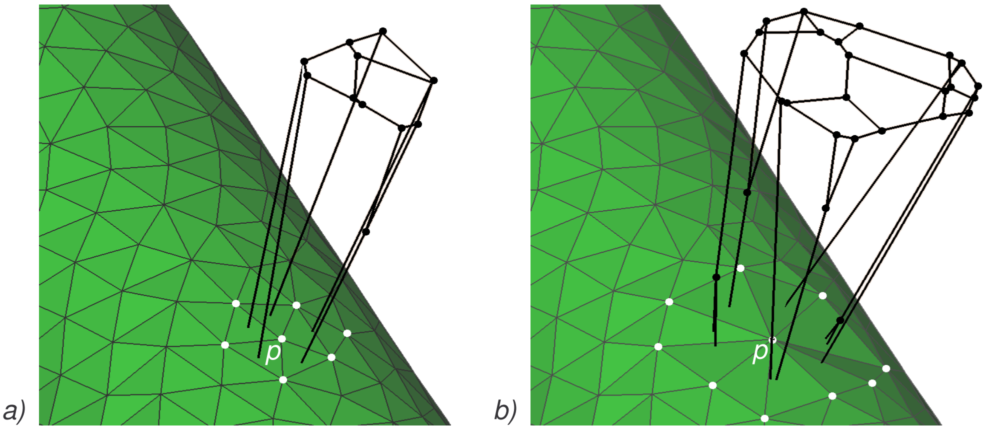


Fig. 4.6: The visualization of the Voronoi cell of the point p, a) the original dataset, white points will be removed, b) the change of the Voronoi cell shape after removing the points, white points are the vertices of the triangles coincident with the point p.

Amenta introduced the algorithm which is based on this observation in [Amenta99]. She defined for the surface triangles following three conditions. Let T be a set of surface triangles, then:

- the set T consists of triangles from Delaunay tetrahedronization whose dual Voronoi edges *intersect* the surface S,
- each triangle in T is *small*, it means, that the radius of triangle circumcircle is much smaller than the distance of triangle vertices to the medial axis,
- each triangle is *flat*, so the normal vector of the triangle makes a small angle with the normal vectors in the triangle vertices estimated using poles.

Under the assumption that the surface S is smooth and sampling sufficiently dense, the first condition assures that T is a piecewise linear manifold homeomorphic to P. The second and third conditions say that any piecewise linear manifold M extracted from T which contains all the points from the input set and for which every adjacent pair of triangles meets at an obtuse angle must be homeomorphic to S.

After the tetrahedronization we can compute the triangle set T as follows. We have the set of Delaunay tetrahedra. For each point p in the point cloud P we can simply find all incident tetrahedra and compute the centers of its circumscribed spheres. These centers form the dual Voronoi vertices of the Voronoi cell around the point p. We mark the farthest *VV* as the positive pole $p+$ and calculate the normal vector estimation n of the surface at the point p as the vector from the point p to the pole $p+$ ($n = p+ - p$).

For each tetrahedron t_1 we compute the center w_1 of the circumscribed sphere (see Fig. 4.7). Then we take all tetrahedron faces (triangles) f one after another and compute the center w_2 of the circumscribed tetrahedron sphere of the opposite tetrahedron (with the shared face f). The edge e from the center w_1 to the center w_2 is the dual Voronoi edge to the triangle f.

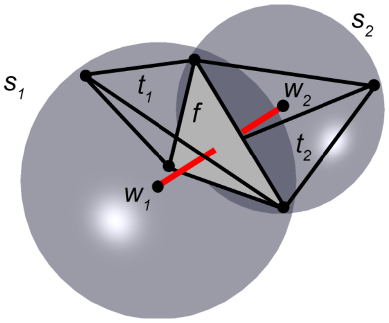


Fig. 4.7: The tetrahedron t_1 has a circumscribed sphere s_1 with the center w_1. The tetrahedron t_2 has the circumscribed sphere s_2 with the center w_2 and t_2 shares the face f (filled light gray) with the tetrahedron t_1. The edge from w_1 to w_2 (bold line) is the dual representation of the triangle f in the Voronoi diagram.

Whether the triangle f belongs to the set of surface triangles T depends on this criterion (Fig. 4.8): for the triangles on the surface, their dual Voronoi edge e has to pass through the surface *S*. Let us denote the vector e_1 as the vector from the point *p* to one of the vertices of the edge *e*, the vector e_2 as the vector from *p* to the other vertex of the edge *e*. The angles $\alpha = \sphericalangle(e_1, n)$ and $\beta = \sphericalangle(e_2, n)$. When the interval $\langle \alpha, \beta \rangle$ intersects the interval $\langle \pi/2 - \theta, \pi/2 + \theta \rangle$ and this condition holds for each vertex *p* of the triangle *f*, then the triangle is on the surface and thus in the set *T*. The parameter θ is the input parameter of the method. When we set the parameter θ to zero then the edge has really to pass through the surface. But due to noise and other sampling mistakes we cannot be so accurate and the parameter θ is set to 22.5 degrees. The theory says that this value is the best for the reconstruction.

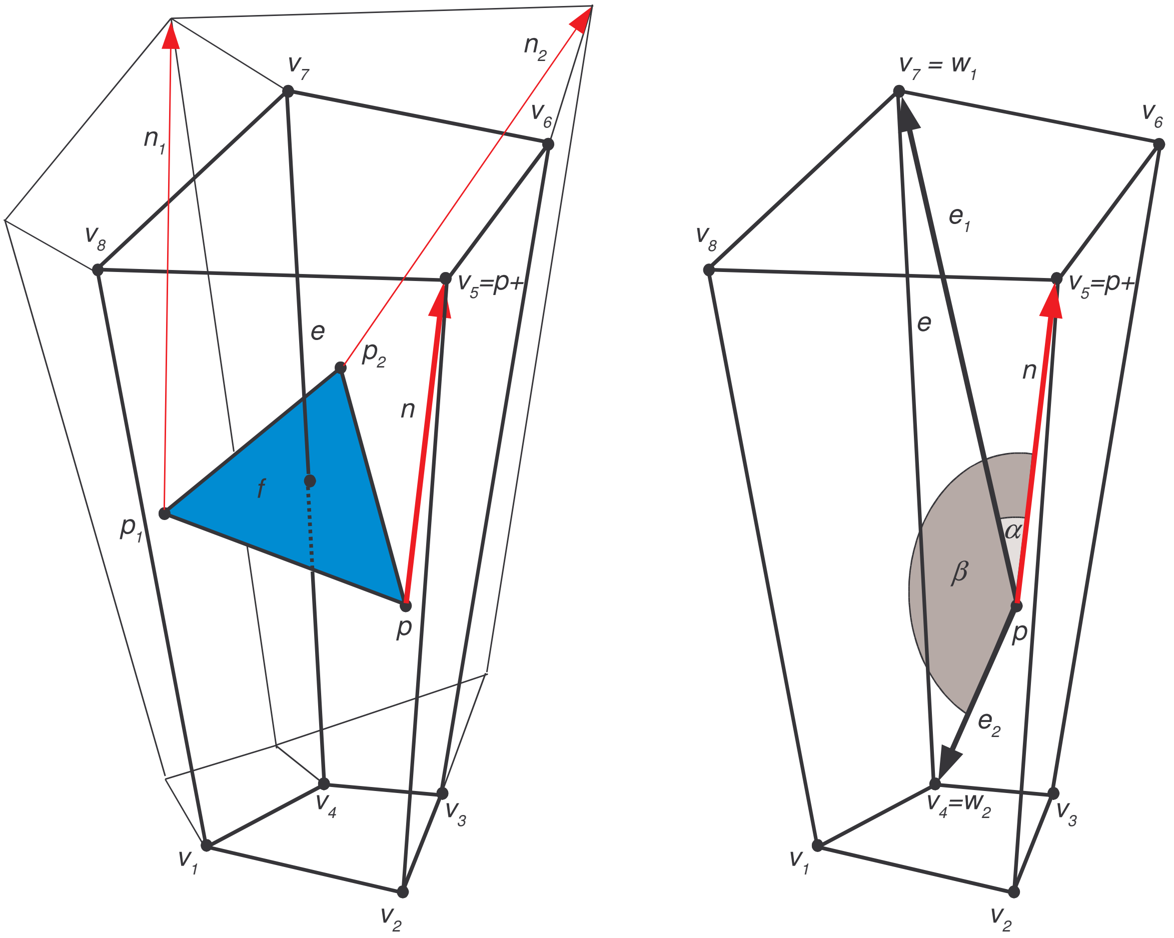


Fig. 4.8: a) The triangle f (shaded) and three Voronoi cells coincidenting with each triangle vertex. The edge e is the dual representation of the triangle in Voronoi diagram. The arrows represent the surface normal vectors n, n_1 and n_2 in the triangle vertices p, p_1 and p_2 estimated using poles. One Voronoi cell of the triangle vertex p is highlighted, its Voronoi vertices are v_1-v_8. The figure b) shows this cell with the computation of the angles α and β. The vectors e_1 and e_2 are the vectors from the point p to the ending points of the edge e. The angle α is the angle between the normal vector n and the vector e_1, while the angle β is between the vectors n and e_2. If the interval <α , β> intersects the interval <π/2 – θ, π/2 + θ> and it holds for other two Voronoi cells too, than the dual triangle to the edge e lies on the surface.

When we get using the above described calculation the set *T* of the primary surface, we use the set *T* as an input of the manifold extraction step. This step is necessary because although many triangles from *T* lie on the surface, they can overlap or create other unwanted configurations. This step is described in its own chapter.

4.5. COCONE algorithm

The COCONE algorithm presented by Dey et al. is very similar to the CRUST algorithm, even the theoretical background was developed by both authors together [Amenta00]. But Dey et al. went more into details and they presented many improvements to the original algorithm. The algorithm has three stages. The first step is called "*candidate triangle extraction step*" and using cocones the set of all candidate triangles is extracted from the Delaunay tetrahedronization. The cocone is a geometrical object derived from the Voronoi cells around each point (the details are presented, e.g., in [Dey01b, Varnuska04]) and it is used for the surface triangles selection, this step is similar to the Voronoi filtering step in the CRUST algorithm.

The candidate set of triangles is already close to a manifold for a sufficiently densely sampled surface but it does not form it. The second step, called "*pruning*", walks outside or inside the triangle mesh. It deletes triangles incident to sharp edges (an edge *e* is called sharp if there are two consecutive triangles incident to *e* such that the angle between them is more than $3\pi/2$) in a cascaded manner. In next improvements of the algorithm, where the boundary or undersampling is detected, we have to be careful in this algorithm step and remove only the triangles whose vertices are in smooth areas (marked by a special flag). The extraction is done in the third step called "*walk*".

The COCONE algorithm was extended in other Dey's papers. The first extension allows to detect boundaries, places of undersampling or oversampling on the surface [Dey01a, Dey01c]. Undersampling happens when the surface has some features such as high curvatures and sampling is not dense enough to capture them. It cannot be avoided when the surface is not smooth, as an infinite dense sampling would be necessary for sharp edges or corners. The boundary can be understood as a special case of undersampling, the scanning process was stopped in the places of surface boundaries and the consequence is the same – missing points. Due to this we cannot exactly decide if some detected part of surface is undersampled and the hole should be retriangulated or if it is a boundary and should not be retriangulated. The counterpart of undersampling is oversampling, which causes difficulties, too, particularly in the postprocessing steps. A surface is sometimes sampled with unnecessarily high density and the surface reconstructed from this sampled points contains large number of triangles in flat parts.

The test of these surface features is done using the shape of the Voronoi cells. For each cell its width and height is computed and using simple heuristic tests it can be quickly detected whether the point lies on the flat or boundary part of the surface.

Another extension tries to bypass the main difficulty of the algorithms based on Delaunay tetrahedronization, the huge memory requirements. For example, our implementation of *DT* is able to process about 250K points on a system with 1GB of memory. The superCOCONE algorithm [Dey01b] is able to reconstruct large datasets using octree space subdivision and applying COCONE to each leaf. As there would be a problem when connecting the parts of the particularly reconstructed surface, the leaves are a little extended before the reconstruction, so close points to the computed octree cell of the neighbouring cells are computed, too.

Watertight reconstruction is sometimes required by the reconstructor, it requires the knowledge that the surface is closed and all possible holes has to be filled by the reconstructor. The extension tightCOCONE able to make a watertight reconstruction was presented in [Dey03] and it is based on the tetrahedra removal.

The last extension of the algorithm presented in [Dey04] deals with the reconstruction of the noisy data. It has two stages - two Delaunay tetrahedronization computations. After the first computation the noisy points are removed, this test is based on the comparison of the radii of tetrahedra circumscribed spheres. In the places of noise the points are close together in all directions and in these places small spheres occur. The second tetrahedronization is then processed on the modified set of points, our tests imply that around half of points is used. After the second tetrahedronization the COCONE algorithm is launched.

5. Problems and analysis

This chapter is more practical than the previous ones and it presents the results and conclusions of our experiments with existing methods. During the development of the CRUST algorithm implementation and its improvements we have tried to identify the reconstruction problems of this algorithm and the causes of such problems. The reconstruction of many datasets were done and the tests confirm the theoretical conclusions known from the papers of Nina Amenta such as that the algorithm works well for sufficiently sampled data of the closed smooth objects without the requirement to be uniformly sampled. In this chapter we will try to analyse the sampled points properties and to describe the behaviour of the algorithm. The properties of the points can be divided into the following groups (will be presented in their own sections in this chapter):

- uniformly sampled points
- different sampling in various directions
- nonuniformly sampled points
- points affected by a noise, outliers

The properties of the sampled surface have also big influence to the results the reconstruction. The surface can have the following properties:

- smooth surface
- surface with sharp edges
- surface with close parts
- surface with boundaries

These properties will be closer explained in the section 5.6 of this chapter. As the run of the algorithm is also strongly affected by the numerical stability, we discuss the stability in its own section.

5.1. Numerical stability

The problem of numerical stability does not belong to the previously described properties but it is important for the success of the reconstruction. We have observed that the code for the computation of Delaunay tetrahedronization is the most critical part as it is susceptible for numerical errors. After many tests we use in the code for the *DT* computation the Shewchuck's library for the numerically stable geometric predicates [Maur03, Schewchuck96].

The reconstruction using the unstable version of *DT* leads to a little surprising observation, the uniformly sampled data of smooth objects was worse reconstructed than the nonuniformly sampled datasets with noise, because the noise made the tetrahedra not so thin and flat and the computation was more numerically stable. Fig. 5.1 presents two examples of the same point cloud reconstructions, the cactus at Fig. 5.1a) was reconstructed using the *DT* with the normal FPU arithmetic, the second cactus at Fig. 5.1b) was reconstructed using the *DT* with numerically stable geometric predicates. The difference is clear, the first figure contains a lot of bad triangles marked as the surface due to bad positions of Voronoi vertices.

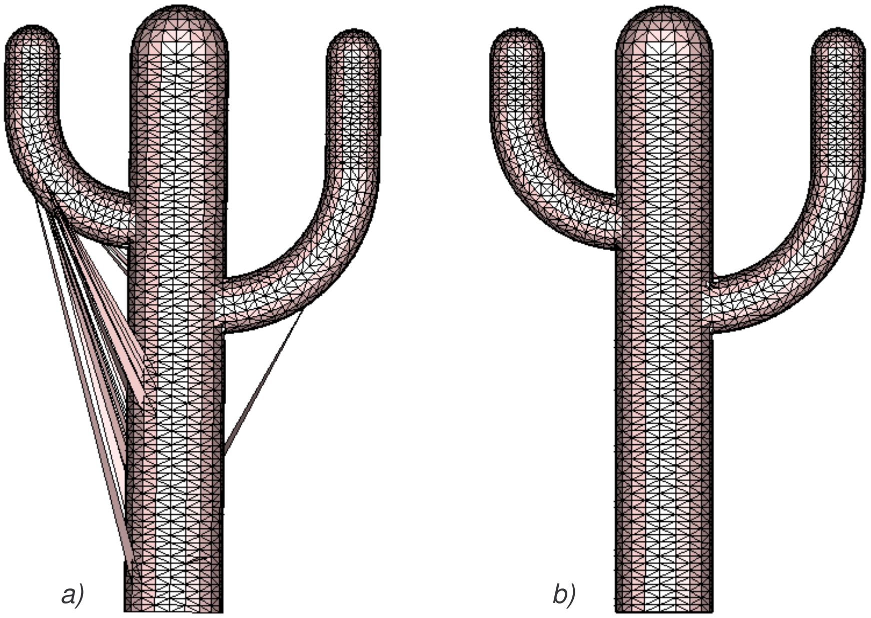


Fig. 5.1: a) An example of the reconstruction without using the numerically stable Delaunay tetrahedronization, b) the reconstruction with numerically stable Delaunay tetrahedronization.

Although the above described problem concerns the implementation and not the algorithm, it is important. The reconstruction problems arise due to sampling properties, whether the surface is undersampled or oversampled, whether it contains boundaries or is closed, whether it is sampled uniformly or not or only in some directions. The surface properties are important, too, the sharp edges make problems in many algorithms. Other question is the noise, the data with three dimensional noise (influencing the point in all spatial directions) is a problem.

5.2. Uniform sampling

Described sampling criteria or surface features influence one another, we cannot simply say that some place of the object is undersampled or that it is just the local boundary. When we stay in the oversampled region, from this point of view other regions seem to be undersampled. If the data is nonuniformly sampled than it looks like undersampled or oversampled.

Generally, the CRUST algorithm works perfectly for uniformly sampled closed smooth objects (if we suppose that all computations are exact). The Voronoi cells of such a point cloud fulfil the criteria given by the theory and the set of primary surface triangles is close to the manifold (only with some overlapping triangles of very flat tetrahedra). Fig. 5.2 shows the reconstructed object with the detail of the triangle mesh after reconstruction. Fig. 5.2c) is the ε-sampling of the surface, the more grey the surface is, the higher is the ε-sampling (in all figures), thus the ratio between the nearest neighbour and the distance to the medial axis is higher, but it does not exceed the value of 0.4 (theoretical guaranties still hold).

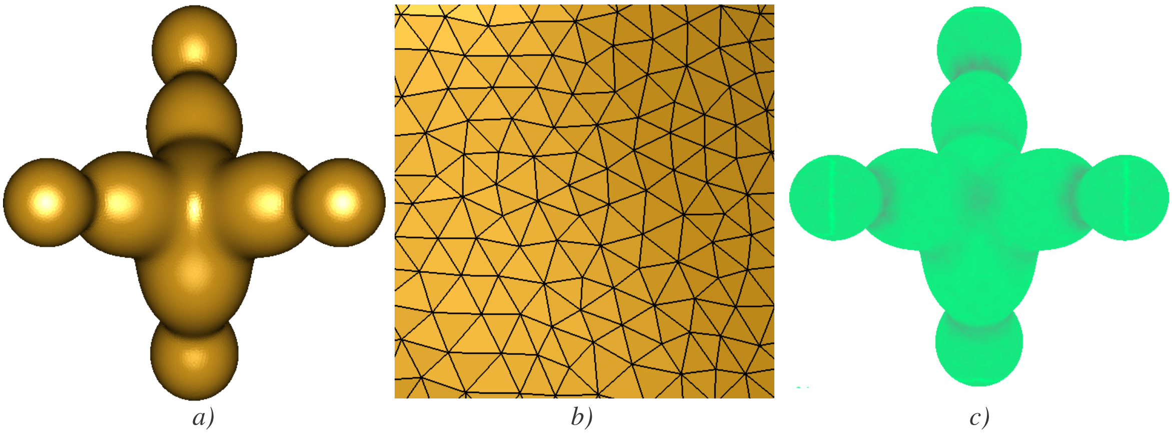


Fig. 5.2: a) An example of the reconstruction of a uniformly scanned object, b) the detail of the reconstructed surface with a triangle mesh, c) the ε-sampling of the surface, all places passed the LFS sampling criterion.

5.3. Different sampling in different directions

Very problematic datasets are the ones, which are sampled in one direction more densely than in the other, see an example in Fig. 5.3. It is a part of a uniformly sampled object (cylinder), which is in Fig. 5.3a) sampled more densely in one direction than in others. The reconstruction fails, because the plane formed only by the points in the direction of plane zy has higher probability to be a part of the surface than the original surface. Fig. 5.3b) presents the same object but sampled in both directions uniformly. We lost some details but the reconstruction is correct.

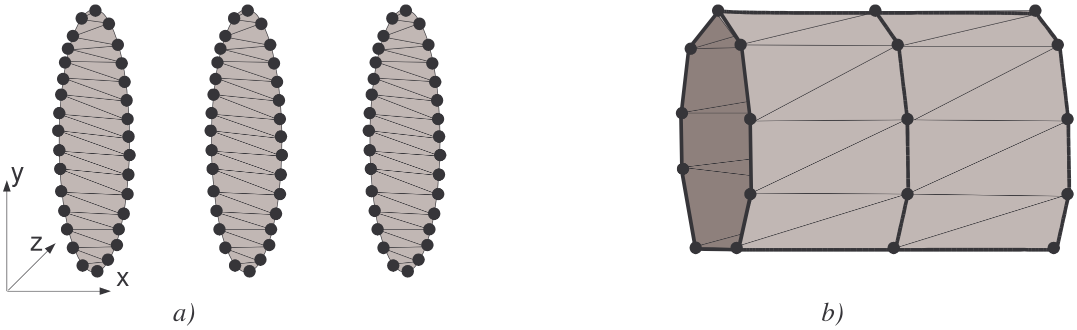


Fig. 5.3: a) An example of the object sampled in one direction more precisely than in others, b) uniformly sampled object (in both directions almost the same resolution).

Fig. 5.4a) shows a real object, which is sampled more densely in one direction in the part of neck and tail. The distribution of the points of this problematic parts is visible in Fig. 5.4b) and Fig. 5.4 d). The reconstruction fails in these parts because the Voronoi cells here have bad direction even though the shapes are thin and long (Fig. 5.4c). The estimated normal vectors direct parallel with the surface. Fig. 5.4e) shows that this distribution of points cannot be detected using the ε-sampling because the shapes of the cells are good, only their directions are bad.

5.4. Nonuniform sampling

When the data is nonuniformly sampled, or uniformly but with some 2D noise, then the reconstruction success depends on the local points configuration, some reconstructions work perfectly and other not. Although CRUST is not very sensitive to the changes in sampling, the Voronoi cells can have bad shapes, the normal vector estimation is not very good and holes or bad triangle configurations appear. An example of a nonuniformly sampled object is in Fig. 5.5a), Fig. 5.5b) shows the detail of the reconstruction. In Fig. 5.5c) the overlapping triangles arise from the local undersampling equally as in Fig. 5.5d) in the region of the ears. Fig. 5.5e) is the ε-sampling, the dark gray parts are the parts with bad ε-sampling and in these parts the reconstruction is not guaranteed.

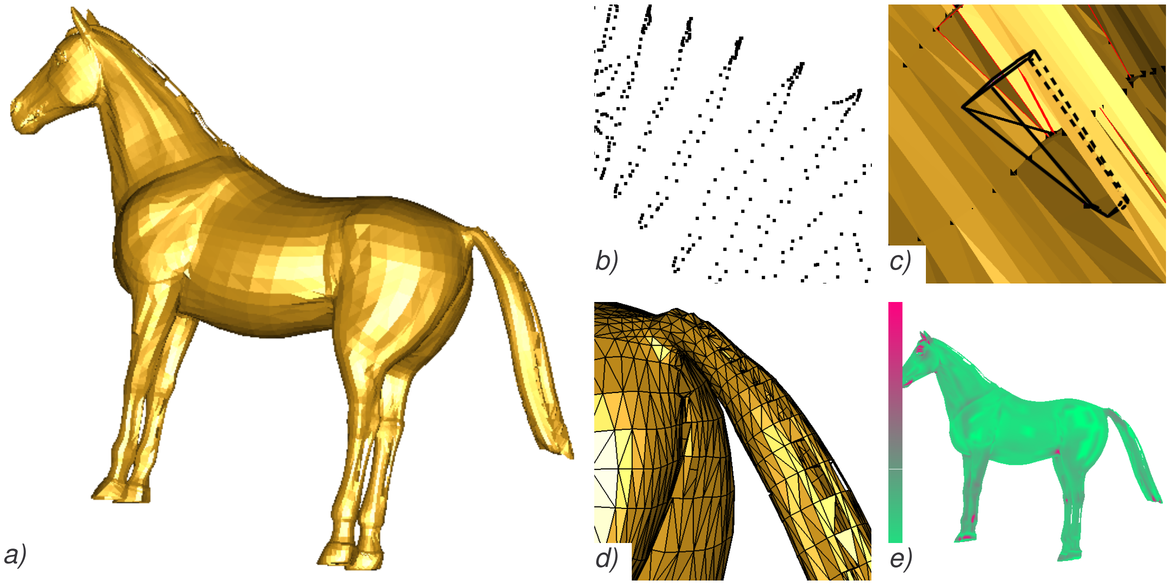


Fig. 5.4: a) An example of the uniformly scanned object, which is scanned in some regions more densely in one direction, b) the distribution of points in the neck part, c) the Voronoi cell of some point in the neck part (the estimated normal is not orthogonal to the surface), d) the tail part of the object with the triangle mesh, e) the ε-sampling, darker places have bad LFS.

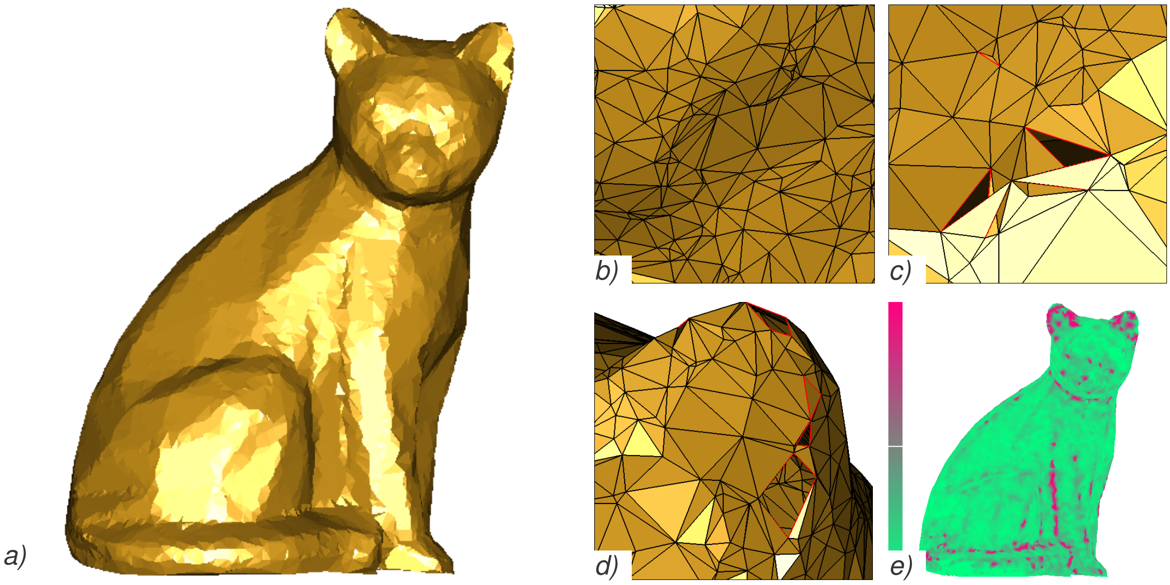


Fig. 5.5: a) An example of a nonuniformly scanned object, b) a detail of the reconstructed triangle mesh, c) the detail of the place with bad triangle configuration, d) the region with a sharp edge and with holes due to a local undersampling (cat's ears), e) the ε-sampling, darker regions have worse ε-sampling.

5.5. Noisy datasets

The algorithms completely fail on the data which contains 3D noise (the noise which is distributed in all directions, the 2D noise is distributed only on the surface). An example of this data is presented in Fig. 5.6a,b). It is a sampled terrain with a lot of noise, the reconstruction failed and the triangle mesh (Fig. 5.6d) is nearly unusable. In Fig. 5.6b), the reason is shown, the Voronoi cell shapes of many points do not fulfill the algorithms requirements. Fig. 5.6e) shows the ε-sampling, well reconstructed parts of the surface have ε bellow 0.4 (bright parts in Fig. 5.6e).

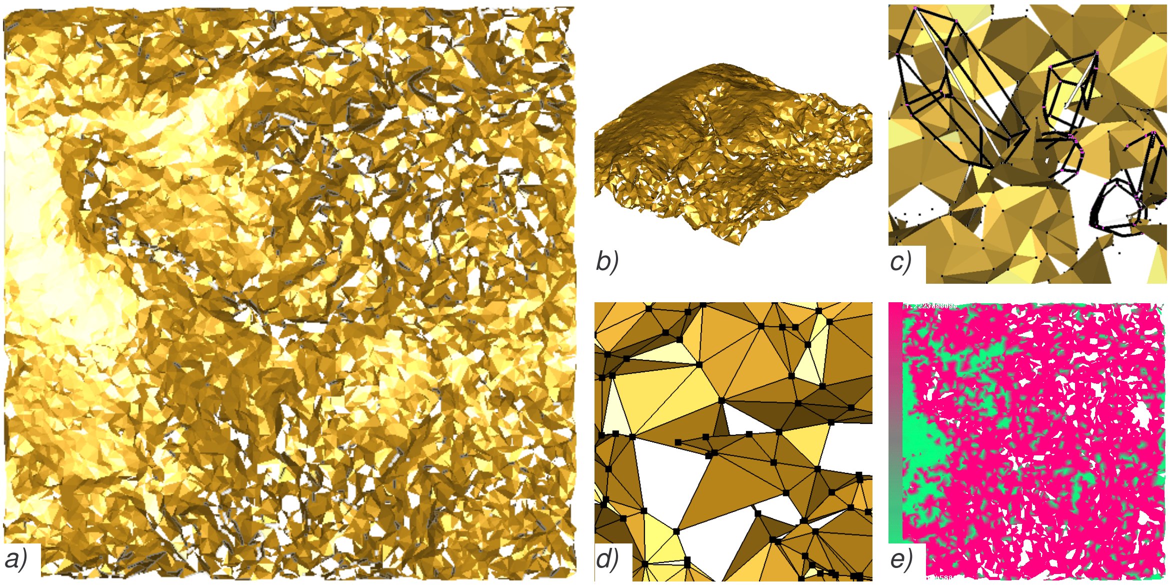


Fig. 5.6: a) An example of the surface with noise, b) another view of the surface, c) the Voronoi cells of some points in the surface, d) the detail of the reconstructed triangle mesh,e) the ε-sampling of the surface.

5.6. Surface features

The described problems depend on the surface sampling and whether there were some errors in the scanning process. The surface can have also some features, such as edges, boundaries or outliers. As mentioned above, the boundary is detected as the local undersampling and, due to it, in the reconstructed surface some triangles arise with incorrect positions.

Edges are problematic, the ε-sampling criterion in these places does not work because the medial axis touches the surface here. It depends again on the local point configuration if we get a correct reconstruction. Fig. 5.7 shows an example of the edge in the surface. The reconstruction is mostly correct even though some triangles are missing and the shape of the edge is not kept. Fig. 5.7a) is the detail of the edge while Fig. 5.7b) presents the larger neighbourhood of the edge with the Voronoi cells. It is noticeable that the Voronoi cells of the points near the edge (four cells at the bottom of the figure) have good shapes. But some of the cells of the points on the edge (the one at the top part of the figure) are very “fat” and the estimated normal is incorrect. Generally, even if the edges are well reconstructed, they need often some remeshing postprocessing steps which improve their quality.

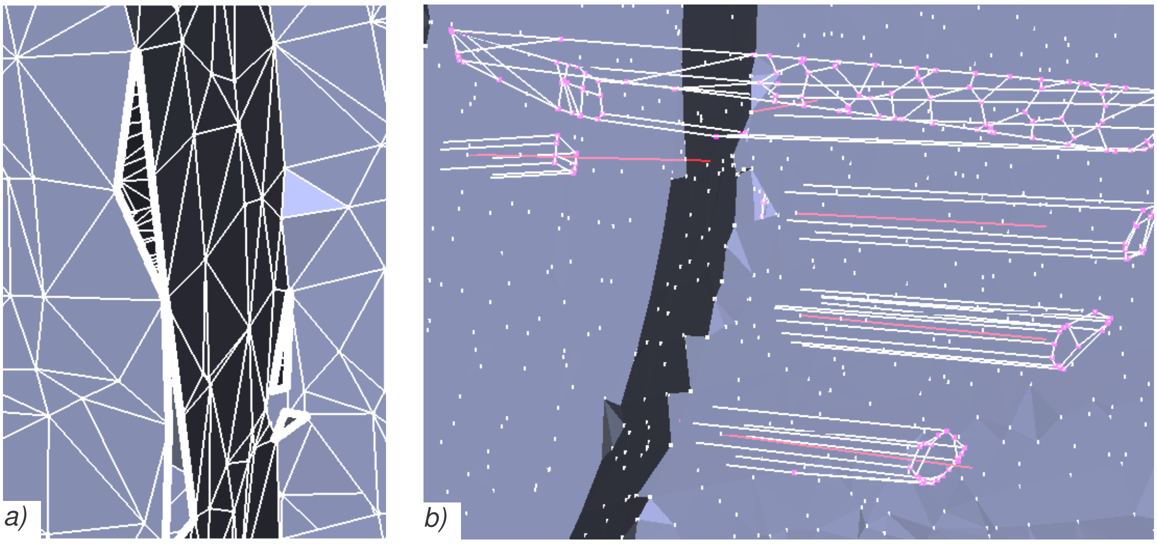


Fig. 5.7: The reconstruction of a sharp edge, a) a detail, b) the Voronoi cells near the sharp edge have required properties but the cells at the points on the edge have bad shape (not thin in one direction).

Outliers are defined as the point or a group of points lying far away from the reconstructed object. They can occur due to noise in the scanning process or they can be a part of some small object which could not be correctly sampled. When the group is big then mostly no problems arise and such small objects are reconstructed as other objects. In the other case some unwanted triangles connect the outliers with the other points making the reconstruction more difficult and the results not very good.

Another problem arises in the case when two parts of surface are mutually very close and the sampling here is not dense enough. Then these places are not correctly reconstructed due to triangles connecting different parts of surface and the last word has to be given by the user saying which part of the surface is related to other parts. The reconstruction of the surface with boundaries has the same consequence. As the surface is very often connected with the other side, the user has to decide if it is correct or not.

Above presented problems show the necessity of the algorithm improvements. Although it is very difficult to implement the algorithm which is able to handle all kinds of data, we have tried to develop some improvements of the original CRUST algorithm.

6. Overview of the proposed solution

Our implementation of the CRUST algorithm was done in the programming tool Borland Delphi in Object Pascal under the operating system Windows XP. The tests of the object reconstruction ran on the CPU AMD XP+ 1500 with 1GB of memory. We aim in this thesis only to the implementation of the one-pass algorithm due to the amount of data limitation in the two-pass algorithm although many presented improvements (mainly the preprocessing steps, the steps of manifold extraction and the triangle mesh improvements) can be used in any other reconstruction algorithms, too. The result of our two-pass algorithm implementation can be found in [Varnuska02]. The work of the original one-pass algorithm can be divided into the following stages (Fig. 6.1):

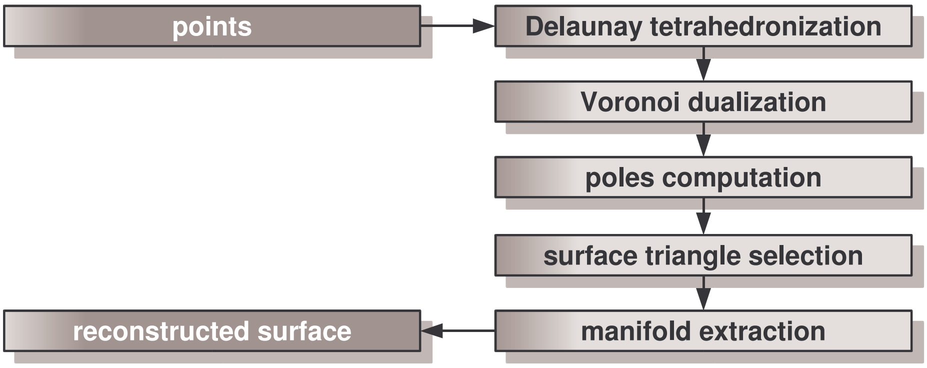


Fig. 6.1: The scheme of the original CRUST algorithm.

The first step is the Delaunay tetrahedronization and its dualization to the Voronoi diagram. Then the poles are computed for each point from the Voronoi diagram and the vectors from points to their related positive poles are taken as the normal vectors. In the next step the whole tetrahedronization is processed and the faces of each tetrahedron are tested whether they belong to the surface or not. The manifold extraction step is the last part of the algorithm and the manifold is created from the set of triangles which passed the previous test.

First we will survey the improvements which we have made to the original algorithm, They consists of several steps divided into three groups:

- the points preprocessing steps
- the surface reconstruction steps
- the triangle mesh filtering steps

The points preprocessing steps modify the input set of points to be better reconstructed by the CRUST algorithm, see Fig. 6.2 for the scheme.

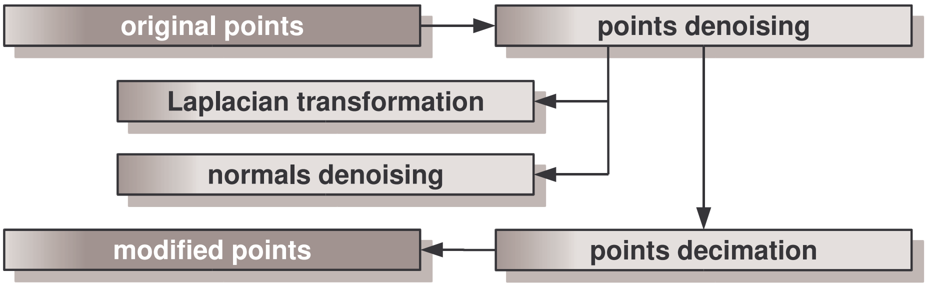


Fig. 6.2: The points preprocessing steps.

The points denoising is a necessary step in the case of data affected by noise. We have tried some approaches for denoising and developed two usable versions, faster, but less successful step based on Laplacian transformation of points and more successful but a little slower version based on our own approach which we call normal denoising. If we have a very large dataset which cannot be reconstructed due to technical limits (memory requirements), we use a very simple approach based on iterative shrinking of the dataset by the reduction of two mutually closest points, we call this step the points decimation. The sequence of these steps is not necessary as shown in Fig. 6.2 and the steps can be swapped.

The output of these steps is the modified set of points. After this we are able to run the modified version of the CRUST reconstruction algorithm. The modified sequence of steps of the CRUST is shown in Fig. 6.3. First the Delaunay tetrahedronization of the points is computed and the Voronoi diagram from the tetrahedra mesh is obtained. Next, the poles and normals vectors for each point are computed from the Voronoi diagram. To make the computation of normal vectors more precise, we use the step called average normal computing [Varnuska03]. The poles and normals are then used for selection of the surface triangles in the next step. In this step each face of the tetrahedra is tested by the Voronoi filtering test whether it could be a surface triangle or not. This step is the only one from the original algorithm which was left unmodified. As some tetrahedra can be very flat, all faces of these tetrahedra can be selected and it brings problems in the following surface extraction step. To avoid such situations, we use a tetrahedra prefiltering step [Varnuska04b] where some faces of these tetrahedra are eliminated. The output from the triangle selection step is the set of triangles which probably lie in the surface and we call this set a primary surface. As the selected triangles do not form a manifold, e.g., some edges may be shared by more than two triangles, the manifold extraction step has to follow [Varnuska03].

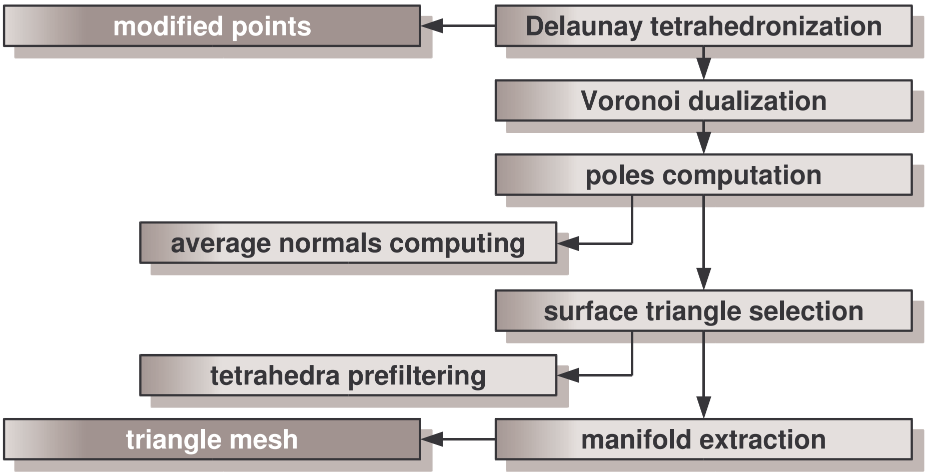


Fig. 6.3: The surface reconstruction steps.

The triangle mesh may not be perfect after the manifold extraction step. We call it a manifold extraction to use the same name convention as in the papers about the CRUST, although this name is not exact as the triangle mesh has not to form the manifold in the case of badly sampled data. To repair the mesh we have developed three steps which are shown in Fig. 6.4.

All these steps work on local triangles configurations. The first step, called mesh filtering, tests the triangle fan around each point and it deletes wrong overlapping triangles. The second step was developed for handling objects with boundaries where unwanted large triangles over the boundaries appear. These triangles are found and deleted so the that

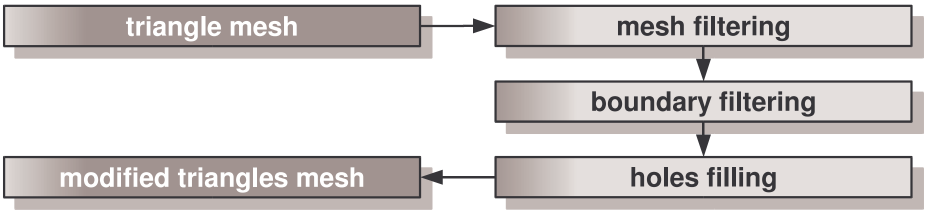


Fig. 6.4: The triangle mesh filtering steps.

boundaries are well reconstructed. As the surface may contain holes and the previously presented filtering steps may also produce the holes, they are are retriangulated in the last step of our approach [Varnuska05a].

In next chapters, all previously presented steps will be explained in detail together with the results. Unfortunately, we have not found any usable approach how to measure the correctness of the reconstruction except the visual comparison. Some methods, such as quadratic error measure, exist but they are used for the comparison of surface approximation techniques. We use an interpolation technique where the surface exactly fits the input points so the quality of the results depends more on the quality of data sampling process.

7. Preprocessing steps

In this chapter we aim to the preprocessing steps of our surface reconstruction approach. We will describe here the steps needed for the points denoising and one possible approach how to handle large datasets.

7.1. 3D-grid

As we have only the points as the input, we need some structures which will help us with point location. All approaches in this chapter are locally based and for each point we need to find its k nearest neighbours. The simplest linear search is really time consuming and not usable even for small datasets, its algorithmic complexity is $O(N^2)$. Therefore we use a 3D-grid structure for speeding up the search, then the algorithmic complexity decreases to $O(cN)$ where c is a constant depending on the uniformity of points distribution but it is much smaller than N.

The 3D-grid is created according to the size of the data. Firstly, the minmax box of the data is built and the longest side of the minmax box is divided by $\sqrt[3]{N}/2$. The gained value represents the size of the cell.

To speed up the search we do not use any pointers and memory allocations, everything is done in simple arrays created in the grid preparation step. The grid is a 3-dimensional array of integers, where the integer value is the index pointing to one of the points related to the cell. The other points in the same cells are accessible by the linked list connected to this point. In the case that the cell has no assigned point, the value is -1. The structure of one item in the table of points (which is just an array of items) consists of three subitems, the real x, y, z coordinates of the point, and two integer indices "*next*" and "*previous*" which refer to other points in the same cell. These indices are used for the bidirectional linked list. The positions *X, Y, Z* of all points in the grid are computed in the grid preprocessing step. The algorithmic complexity of this step is *O* (*N*) and thus the preprocessing step is very fast.

The process of *k* nearest neighbours search of some point *q* works as follows. First, we look for the neighbours present in the same cell as the point *q*, the distances to the point *q* are computed and only points with *k* nearest distances are returned. In the case that there is less points than *k*, we search for other points in all directions in neighbouring cells. After we have found *k* neighbours we have to extend our search a bit to more neighbouring cells because there could be closer points. Fig. 7.1 shows the case when this could happen. We are looking for the nearest neighbours for the point *q*. The point p_1 was found as the *k*th nearest neighbour, but we have not searched in the white cells yet where some closer point should be (e.g., the point p_2). Therefore we have to extend the search to these cells, too.

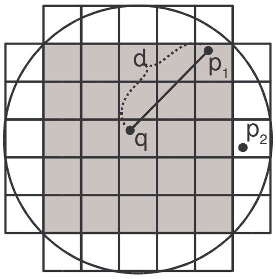


Fig. 7.1: An example of the k nearest neighbours search.

A typical dependency of the speed on the grid size for the computation of the "bunny" dataset is shown in Fig. 7.2. The value *d* means the denominator in the computation of the cell size $\sqrt[3]{N}/d$. It can be seen that *d*=2 is for this dataset the best value, for other datasets the value can be different but this value is a compromise.

7.2. Points denoising

The problem of the developed algorithms is the necessity to have "well" sampled data. In the case that the input dataset does not satisfy the sampling conditions of the algorithm, the reconstructed surface is bad and unusable. In Chapter 5 we have presented some problems which may occur when reconstructing the surface from badly sampled data. It was said that the datasets affected by noise are the worst data, these datasets are almost impossible to be reconstructed with any current existing interpolation algorithms. The only approaches which can obtain a usable surface from such datasets are the approaches which approximate the surface.

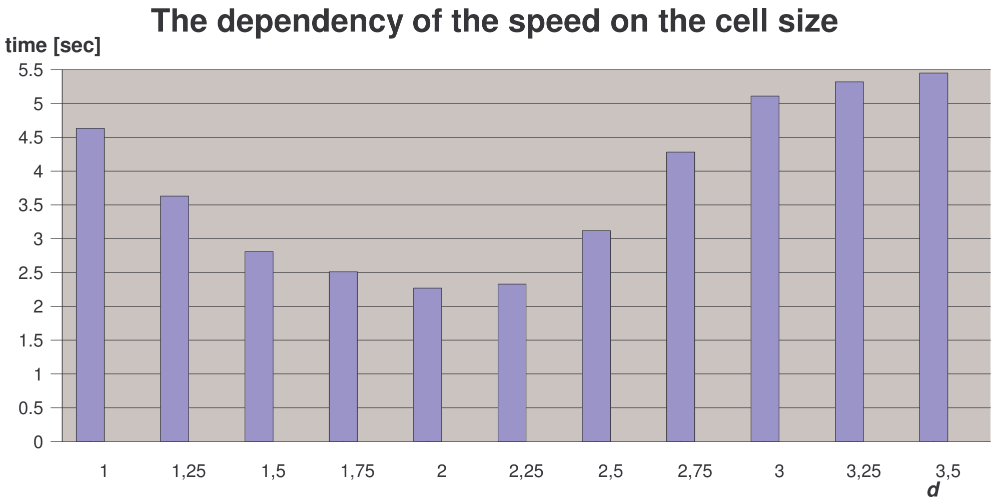


Fig. 7.2: The graph of the dependency of the speed on the cell size.

There are not many papers dealing with the surface reconstruction of badly sampled datasets. Kolluri et al. presented the Eigencrust algorithm [Kolluri04] able to reconstruct data affected by noise and outliers, it is built on the CRUST algorithm. Dey et al. gave robustCOCONE [Dey04] which creates a watertight reconstruction from noisy data. It needs to compute two Delaunay tetrahedronizations and is limited to closed surfaces. Other way how to denoise the data is low pass filtering in the function domain [Pauly01], but the surface is only an approximation. These approaches are rather complicated to implement and they cannot be used in combination with already implemented surface reconstructor. That is why we have tried to develop other steps able to denoise the data and to repair the reconstruction, to be combined with any other reconstruction approaches.

We have also tried to reconstruct some of noisy datasets by a free surface reconstructors, an example of the result of one reconstruction is presented in Fig. 7.3. Fig. 7.3a) shows the result of the INRIA surface reconstructor which can be found in [INRIA_WEB], Fig. 7.3b) then shows the work of the COCONE reconstructor from [Dey_WEB] and the last Fig. 7.3c) presents the reconstruction by our implementation of the CRUST algorithm without points denoising. Even without enlargement it is clear that the reconstructions failed and the results are unusable.

We have developed two possible solutions how to remove the noise from the data. Both solutions employ the fact that the CRUST algorithm is not very sensitive to the changes in sampling density and it has no problem with such surface reconstructions. One solution is based on the Laplacian transformation of points and the second one on normal vectors and tangent planes estimation (which represent the surface).

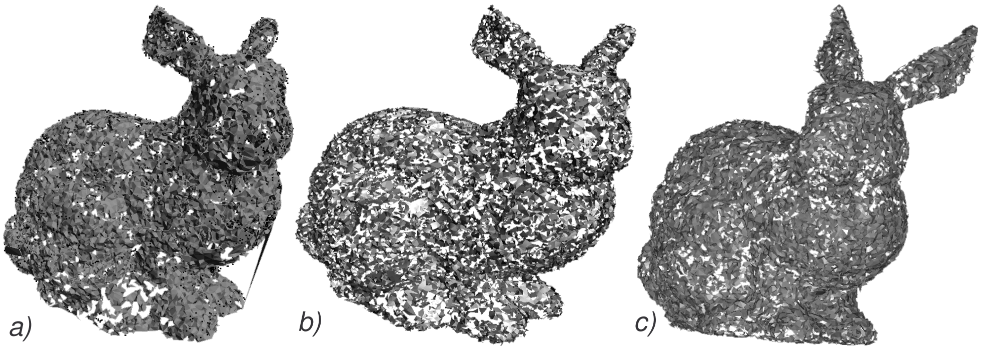


Fig. 7.3: The unsuccessful surface reconstruction of a noisy dataset, a) by the INRIA surface reconstructor, b) by the COCONE, c) by our CRUST implementation without points denoising.

7.2.1. Laplace denoising

The simplest way how to denoise the data is derived from the denoising of the triangle mesh made by the Laplacian smoothing. Unfortunatelly, we cannot use robust approaches known in the surface smoothing (such as in [Freitag97]) for the points denoising, because at the process of reconstruction we do not know the surface and thus the connectivity yet. So, the simplest way how to get off the noise is to use the Laplacian transformation which does not need to know the connectivity. For each point *p* of the dataset we have to find *k* nearest neighbours p_i and compute the centre *p'* of the neighbours where the point *p* has to be placed. The computation is easy:

$$p' = \frac{p + \sum_{i=1}^{k} p_i}{k+1} \tag{16}$$

One of two input parameters to the Laplacian smoothing is the number of neighbours *k*. As we can use the Laplacian smoothing repeatedly, the other parameter specifies a number of iterations, where one iteration means the Laplacian smoothing of the whole point set *P*.

When increasing the number of neighbours, the results become better, but when we use too many neighbours, the reconstruction is without any small features and becomes oversmoothed. Using a small number of neighbours does not bring a positive effect either, because the denoising has not so strong influence in this case and the data remains noisy. Thus the best number of neighbours is about twenty, this number came from our experiments. Fig. 7.4 shows the reconstruction of the "cactus" dataset for a different number of neighbours.

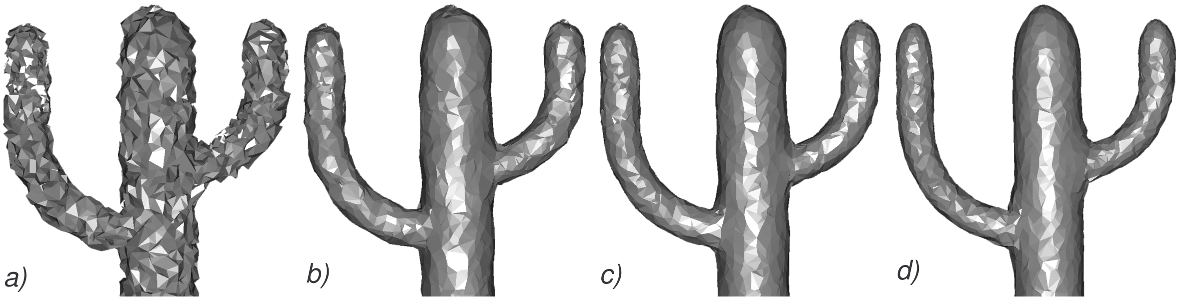


Fig. 7.4: The reconstruction of the "cactus" dataset, a) without denoising, b) 10 neighbours, c) 20 neighbours, d) 40 neighbours.

Changing number of iterations brought one surprising effect which is depicted in Fig. 7.5 (black points are the original points moved in the direction of black line to the white points). The first iteration smooths the surface, but more iterations start to create clusters of points, which are moving closer together when the number of iterations increases. Therefore it has no purpose to use more iterations than one. The reason for this is probably a tendency of the Laplacian smoothing to shrink the data sets and because there are no connections between points as in surface denoising.

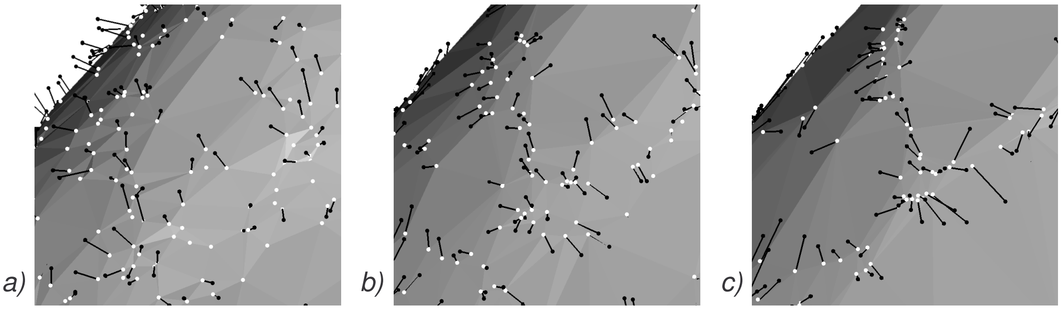


Fig. 7.5: The clustering effect of more iterations, a) after one iteration, b) after two iterations, c) after three iterations

One of the most important reasons why the reconstruction from the noisy datasets fails is that the sampling does not fulfil the *LFS* sampling criterion. Fig. 7.6a) shows the reconstruction of very noisy dataset with the visualization of the *LFS* in Fig. 7.6b). The darker areas are the places which do not satisfy the criterion and as there is a lot of such a places, the result is unusable. In Fig. 7.6c) there is the reconstruction of the same dataset but with Laplacian denoising, its *LFS* is presented in Fig. 7.6d). It is clear that the *LFS* is much better than in Fig. 7.6b) and the result of the reconstruction is good (the dark places are the places where the LFS does not fulfil the sampling criterion).

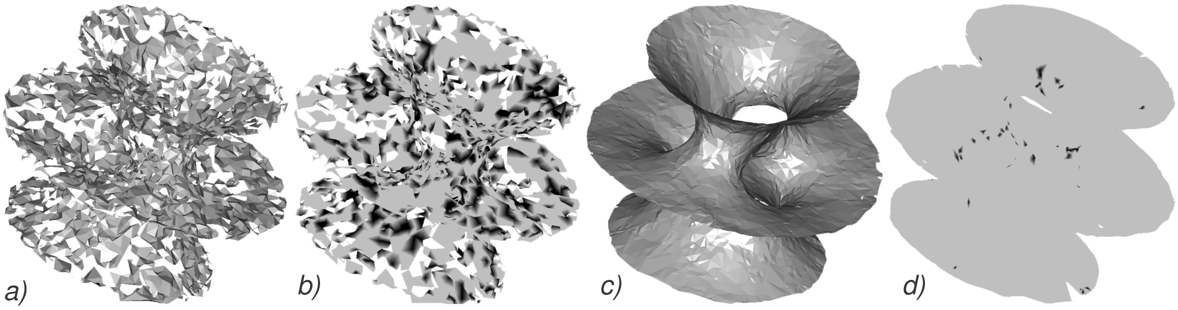


Fig. 7.6:The surface reconstruction of a noisy dataset, a) badly reconstructed surface without denoising, b) the LFS of the surface from a), c) the reconstruction after Laplacian denoising, d) the LFS of the surface from c).

7.2.2. Normal denoising

The second approach we have tried is more complicated but more powerful. It is built on the estimation where the surface probably lies for each point and on the normal vectors estimation. Then the points are moved to the specific location on the estimated surface and the denoised points are reconstructed. After the reconstruction and the mesh improvements (as presented in the following chapters), the points can be moved back to their original positions. We get than the surface which interpolates exactly the input points, otherwise we get the smooth surface but where the points do not represent the original dataset.

The normal vectors and tangent planes estimation for each point is performed by a principal component analysis (a similar approach as in [Hoppe92]).

For each point p its k nearest neighbours p_i are found and the centroid c is computed (the same calculation as for p' in (16)), this centroid is used to translate the points to the origin of the local coordinate system.

We want to find the tangent plane in which the centroid lies, then the normal vector can be computed using the principal component analysis. We have to form the covariance matrix C (shown in (17), p_x, p_y, p_z are the coordinates of the point p, c_x, c_y, c_z are the coordinates of the centroid) from the k nearest neighbours, a symmetric positively semi-definite matrix of 3x3 size [Lukasova85].

$$C=\begin{Vmatrix} \sum_{i=1}^{k}(p_{ix}-c_x)^2 & \sum_{i=1}^{k}(p_{ix}-c_x)(p_{iy}-c_y) & \sum_{i=1}^{k}(p_{ix}-c_x)(p_{iz}-c_z) \\ \sum_{i=1}^{k}(p_{ix}-c_x)(p_{iy}-c_y) & \sum_{i=1}^{k}(p_{iy}-c_y)^2 & \sum_{i=1}^{k}(p_{iy}-c_y)(p_{iz}-c_z) \\ \sum_{i=1}^{k}(p_{ix}-c_x)(p_{iz}-c_z) & \sum_{i=1}^{k}(p_{iy}-c_y)(p_{iz}-c_z) & \sum_{i=1}^{k}(p_{iz}-c_z)^2 \end{Vmatrix} \tag{17}$$

This matrix is used for the eigenvalues and eigenvectors computation. The computed eigenvectors show the directions of three fundamental variances of the data (we can imagine them as the main axes of the ellipsoid best approximating the points). The eigenvectors related to two biggest eigenvalues have the direction of the biggest data variance, transferred back to geometry, they form the tangent plane which is the best approximation of the used *k* points. The eigenvector related to the smallest eigenvalue is the required normal vector, it is the direction of the minimum variance of the data.

In the next step the points are moved in the direction of the expected surface estimated by the computed tangent plane, see Fig. 7.7. As we have an estimation of the normal vector *n* and the tangent plane *t* with the centroid *c* for each point *p*, we can move the point *p* in the direction of the estimated normal *n* to the tangent plane. The same computation is done for all points from the input dataset.

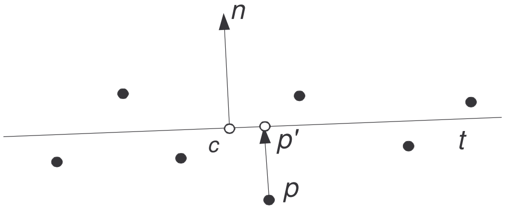


Fig. 7.7: The process of denoising, k nearest neighbours for the point p are used for the centroid c, tangent plane t and normal vector n computation, the point p is transformed to p' lying in the tangent plane in the direction of n.

The biggest problem of the Laplacian transformation of points, which was described in previous section, is that we cannot use the process repeatedly because the points start to create clusters which do not allow a correct reconstruction. This effect, not observed in the Laplacian smoothing of meshes, arises from the fact that the points do not have any connections to its neighbours as in the mesh and when we repeat denoising, the points move close together in clusters and its *k* nearest neighbours are chosen from this cluster which is then smaller and smaller. In our approach, this restriction is no more true because each point is moved just to the estimated surface in the direction of the normal vector and we can use the process repeatedly to get better denoising.

The last phase of smoothing, the points translation, is used in the case we want to use original non-smoothed points. The points are moved after a complete reconstruction to their original positions. This translation may cause intersections or overlaps of triangles but due to translations of points to their original position in the direction of normal vectors, there is low probability of such an error. We did not detect any error of this kind in our experiments. Fig. 7.8 shows the results of such operations on a critical part of some surface where this problem may occur. At the end, some robust mesh smoothing can be used.

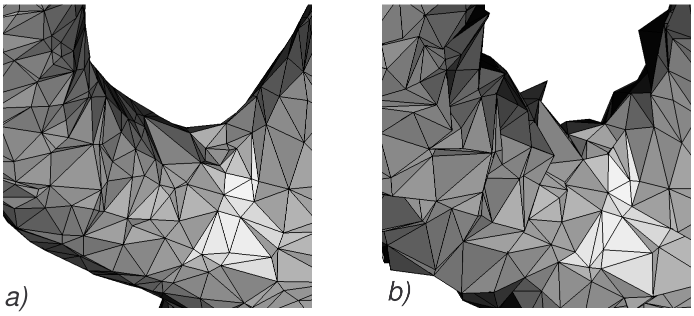


Fig. 7.8:The reconstruction of a noisy dataset with normal denoising, a) without points translation, b) with points translation to their original positions.

The number of iterations has a big influence to the process of denoising. One iteration is the complete process of denoising. Thus it is necessary to create the 3D-grid because the position of points changed in the previous iteration, to find k nearest neighbours, to compute the normal vectors and tangent planes and to move the points. For less noisy data one iteration suffices and when the data contains more noise, we should use more iterations. But we observe from our experiments that two iterations are enough even for very noisy data. Fig. 7.9 shows the reconstruction of a noisy dataset with increasing number of iterations. If we compare Fig. 7.9c) and Fig. 7.9d), there is not so much difference between the moved points and the reconstruction is correct, therefore two iterations is enough for correct surface reconstruction.

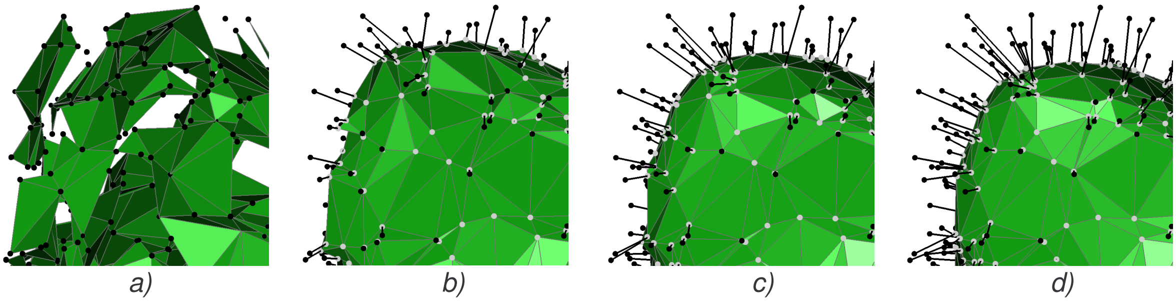


Fig. 7.9: The reconstruction of noisy data, white points are moved denoised points, black points present the original positions of points, a) the reconstruction without denoising, b) after one iteration, c) after two iterations, d) after three iterations.

The number of neighbours has also big influence to the success, Figure Fig. 7.10 presents this. We have changed the number k from 5 to 30 and compared the results. We observed that small value of k up to about 10, depending on the model, does not bring a usable effect because the points remain noisy and the reconstruction fails. The reconstruction using bigger values than about 25 successes but other problem arises, the points are too much smoothed and the places, where two parts of surface are mutually close, can wrongly connect. Thus the best number of *k* is about 20.

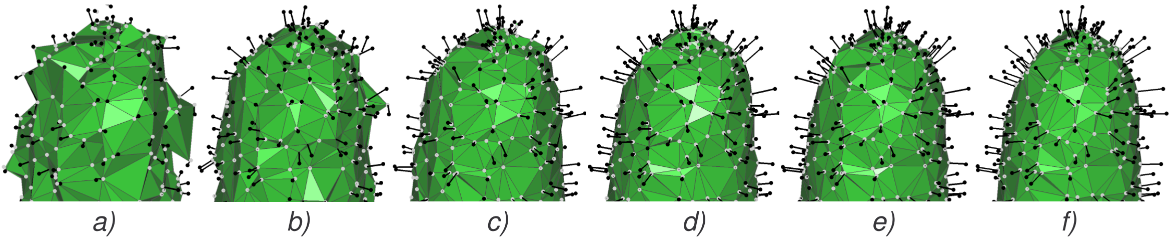


Fig. 7.10: The reconstruction of noisy data, white points are denoised points, black points show the original position, a) k = 5, b) k = 10, c) k = 15, d) k = 20, e) k = 25, f) k = 30.

We have found that the usage of this approach is not only for points denoising. The CRUST algorithm, and many others, too, is designed for the reconstruction of smooth surfaces without sharp edges. In the case of sharp edges (recall section 5.6) the correct reconstruction is not guaranteed because the estimation of normal vectors in these places fail due to bad shapes of Voronoi cells (in the CRUST algorithm). Then the surface may contain holes and badly connected and overlapping triangles. But we can artificially smooth the data, reconstruct the surface and move the points back to their original positions. In Fig. 7.11 a surface with sharp edges is presented, Fig. 7.11a) shows the reconstruction after denoising without the points translation while Fig. 7.11b) presents the reconstruction with points translation.

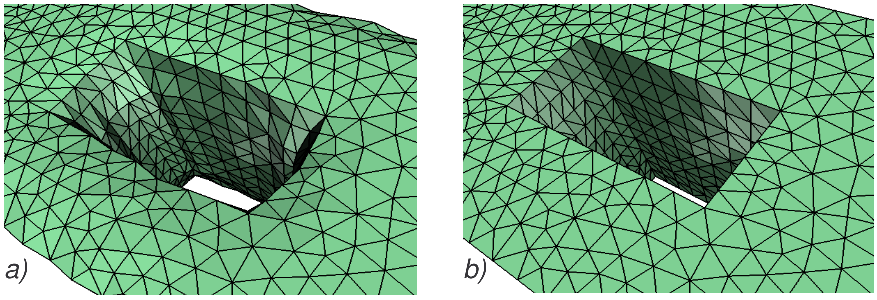


Fig. 7.11: The use of denoising in the reconstruction of sharp edges, a) the reconstruction after denoising without points translation, b) with points translations.

As we have not found many papers dealing with the reconstruction of noisy datasets, we can compare our results only with one approach, the new Dey's robustCOCONE algorithm [Dey04] especially designed for noisy datasets. The quantitative comparison of the methods is difficult so we compared the methods visually, looking for the surface correctness, our results were similar or better than the robustCOCONE algorithm. The algorithm in the provided implementation has one disadvantage (the author is currently working on a newer version without this restriction), it is designed for watertight reconstructions, so the datasets with boundaries may not be correctly reconstructed. The output from the algorithm is a triangle mesh containing less points than in the input because the algorithm tries to remove noisy points in the first tetrahedronization and in the second it reconstructs the rest of points. The watertight models we had were correctly reconstructed, but on some places, mainly where two parts of surface were mutually close, some incorrect reconstruction appeared. Our approach reconstructed this places better than robustCOCONE.

An example of the reconstruction of such a dataset is shown in Fig. 7.12a), b) and c). The dataset has 10K points, robustCOCONE algorithm created an output mesh with 4.5K points, an incorrectly reconstructed part is visible in Fig. 7.12f), where the details of the surface is presented, as well as in Fig. 7.12d) and e). The surface is badly connected with the other part of surface due to the limitation of the algorithm to watertight reconstructions.

Fig. 7.12g), h) and i) show reconstructions of the “bunny” dataset, the surface was well reconstructed by both approaches, while Fig. 7.12j), k) and l) show the reconstruction of a part of some terrain, the robustCOCONE approach had problems on the edges and corners of the surface due to its limitation to the closed surfaces, large triangles appeared there. All outputs in Fig. 7.12 are created using $k = 20$ neighbours and two iterations.

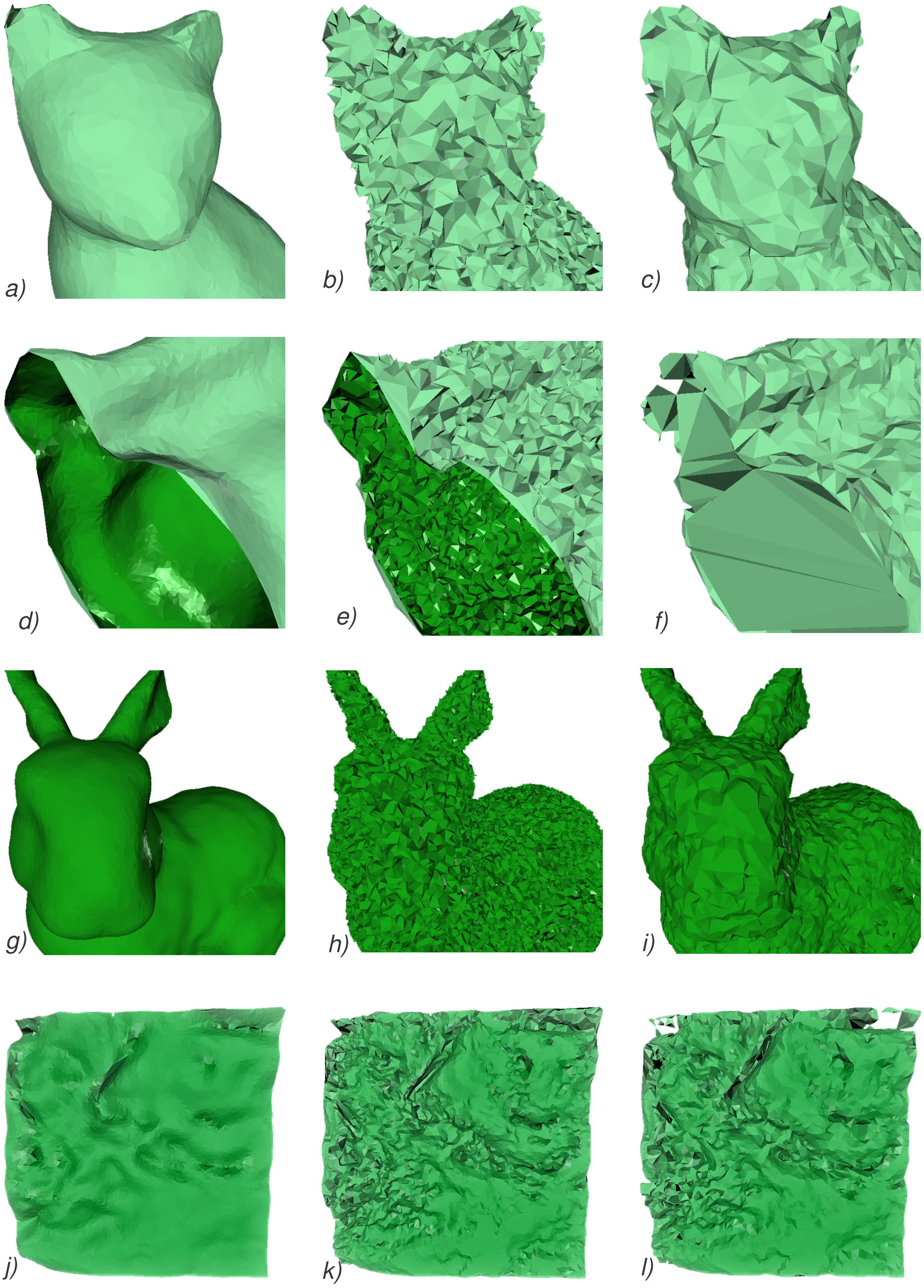


Fig. 7.12: The complete reconstruction of the noisy datasets, for comparison are presented in a), d), g), j) the reconstructions using our approach of normal denoising, in b), e), h) and k) the reconstructions using our approach with the points translation and in c), f) i) and l) the reconstruction by the Dey's robustCOCONE algorithm.

7.3. Points decimation of large data

Surface reconstruction algorithms consume a lot of memory, especially for datasets containing several millions of points. In our implementation of the CRUST, we need at least 1GB of memory to process 250K of points. In practice, it is often necessary to deal with even larger datasets. For example, the model of Lucy from Stanford Repositories having 14 millions of points requires about 10 GB memory.

As our implementation runs on the 32-bit version of the OS Windows, we can use maximally 2GB of address space for the computation (other 2GB are accessible only for the system). Therefore, we are limited now to up 500K of points, which is not so much. We have few possibilities what to do with such a limit. The first possibility is to use less memory consuming data structures. We have observed that in our implementation the most greedy structure is the DAG. It is used in the computation of the Delaunay tetrahedronization for fast search of the tetrahedra containing the inserting point. But even when we changed the DAG to a simple tetrahedra walk – from some starting tetrahedron we choose the direction to the point to be inserted and by walking through the tetrahedra neighbours we find the tetrahedron containing the point (for details see [Devillers01]) – we are not able to reconstruct more then 1.5M of points on 1GB of memory. It means that this is not the right way for the reconstruction of large objects.

The second possibility is to use an existing approach and to divide the datasets to small groups of points, to reconstruct the surface and to merge the mesh together. This approach is used, e.g., by Dey in his superCOCONE algorithm [Dey01b] which uses octree for the space subdivision. Dey's largest reconstructed dataset contained about 3.5M of points. The problem of such approach is that merging is not easy and the surface may suffer from artefacts, especially for the data not passing the sampling conditions.

The most complicated way is to use the distributed processing. We cooperated with our colleague Ing. J. Kohout who have developed the toolkit based on sharing the data on the network. This approach seems to be the best because it allows to reconstruct the datasets with theoretically unlimited size. We have some good preliminary results but it is not still complete. There is a lot of work in computer communication which works correctly although slowly and in the implementation of the reconstruction. We will discuss this approach in Chapter 10.

The simplest implementing possibility is to use points decimation approach to reduce the amount of data and to reconstruct such a reduced dataset. We have observed that when we merge two points which are the nearest neighbours, the geometry of the reconstructed surface will not change to much, see Fig. 7.13. The process has to be repeated until the number of points is lower than some maximum. As it is necessary to store coordinates of all points and to use some structure for faster nearest neighbours search (the 3D-grid in our case), this approach is theoretically useful up to about 60M of points on 2GB system.

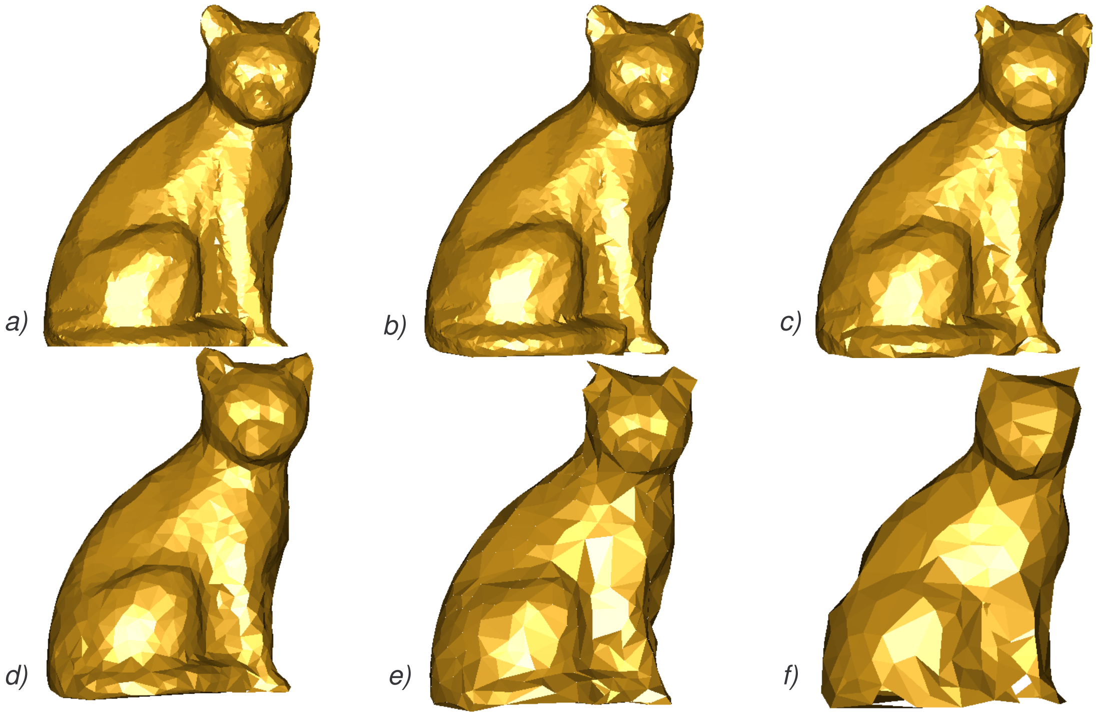


Fig. 7.13: The test of the points decimation before the reconstruction, a) the reconstruction of the original point cloud with 10000 points, b) 5000 points, c) 2500 points, d) 1250 points, e) 600 points, f) 300 points.

We have also tried to decimate and to reconstruct large dataset, the results are presented in Fig. 7.14. All datasets were decimated to 200K points, the biggest reconstructed dataset has 10M points and the time for its decimation was about 80 minutes.

In this chapter we have presented three approaches what to do with large and noisy data. The output of these steps is the modified set of points which are more appropriate for the reconstruction by the surface reconstruction step described in following chapter.

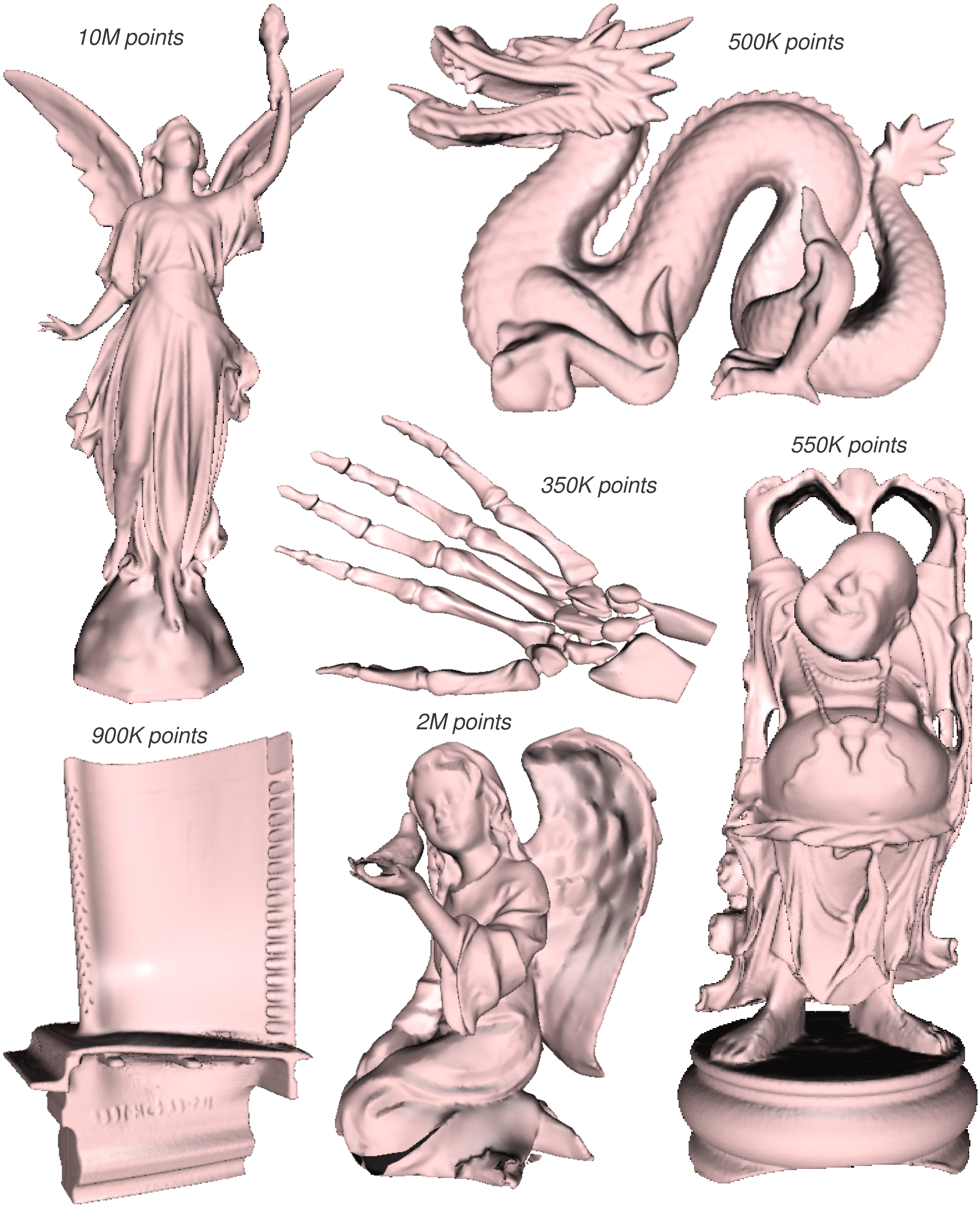


Fig. 7.14: The reconstruction by points decimation to 200K of the large datasets from Stanford.

8. Surface reconstruction

The input to the reconstruction steps is the set of points sampled from some surface. This chapter presents the steps needed to “conversion” of this initial point set to the triangle mesh.

8.1. Delaunay tetrahedronization

For the tetrahedronization we use a code implemented in our team [Kolingerova02]. The code is fast and robust (it utilizes the numerically stable geometric predicates library [Maur03, Schewchuck96]), it uses the method of incremental insertion for the creation of the tetrahedronization. The first phase of the algorithm is adding four new points which form the initial tetrahedron containing all input points. One point after another is inserted to the tetrahedronization and the tetrahedron in which the point is contained is located. This tetrahedron is divided into four new tetrahedra (if the point lies inside the divided tetrahedron) and the Delaunay conditions are locally checked. If they are not satisfied then the local tetrahedra configuration is changed by faces swaps. The algorithm continues recursively until the Delaunay conditions are restored. When all points have been included to the tetrahedronization, the last step is then deletion of the initial tetrahedron, its four points and tetrahedra incident to them.

A problematic part of the algorithm is the tetrahedron location, to which the inserted point belongs. If it is implemented by brute force, all created tetrahedra has to be processed and the one containing the point found. The algorithm complexity is $O(N^2)$ and it is unusable for larger data sets. Our implementation uses the directed acyclic graph (DAG) for point location which speeds the location, the algorithm complexity decreases to O (N log$N$) – O(N) in the expected case. Unfortunately, the use of the *DAG* consumes more memory and the algorithm is able to process only medium size sets, around 250K points on 1GB of memory. More details about the stability were mentioned in section 5.1 and the details about our implementation can be found in [Kolingerova02, Kohout03]. For the task of Delaunay tetrahedronization there exists more approaches for speeding up the process but the memory limit will always spring up. Currently, we test the implementation of the tetrahedra walk algorithm which does not need to use the memory expensive *DAG*. Using this approach we were able to tetrahedronize more than 800K of points on 1GB of memory in penalty to some slow-down.

Another way how to reconstruct large datasets is to use the distributed computing of Delaunay tetrahedronization developed by Josef Kohout, see e.g. [Kohout05]. His approach has many advantages, one of the most useful is utilization of the virtual shared memory module (*VSM*) enabling to share the data across the whole network. For the application the use is transparent, it calls just the *VSM* functions and the module obtains the data for the application. We will discuss the distributed computation in the Chapter 10.

8.2. Voronoi dualization

After the Delaunay tetrahedronization computing we can obtain by dualization the Voronoi diagram. For each point *p* we take all tetrahedra which are incident with this point *p* and compute the centres of their circumscribed spheres. The centres form the Voronoi vertices of the Voronoi diagram and they are used directly for the poles computation. We use for the computation the code from the library of geometrical stable predicates because very flat tetrahedra makes the calculation of the centres of circumscribed spheres numerically unstable.

Each triangle in the Delaunay tetrahedronization, except the triangles in the convex hull, has two coincident tetrahedra (as shown in Fig. 4.7). The centres of circumscribed spheres of these tetrahedra are vertices of the dual edge to the triangle. The edge is later used for the test whether the triangle belongs to the set of primary triangles. In the case of the triangle lying in the convex hull we cannot compute the second vertex of the dual edge, so the triangle normal is used instead.

8.3. Poles computation

The Voronoi vertex with the maximum distance from the point *p* is marked as the positive pole *p+* of the cell and the vector from the positive pole to the point *p* is the estimated normal vector *n* of the surface in this point. The negative pole is the Voronoi vertex with the maximum distance on the opposite side of the plane defined by the point *p* and the normal vector *n*. The schema of the poles computations is in Fig. 8.1.

```
input:       the set of points and its tetrahedronization
output:      the positive and negative pole for each point

for each point $p$
    //create the set T of incident tetrahedra
    $T = \emptyset$
    for each tetrahedron $t$ incident with the point $p$
        $T = T \cup t$
    end for

    //compute the positive pole p+
    $max = 0$
    for all $t \in T$
        $c$ = center of the circumscribed sphere of $t$
        if $max < |c - p|$
            $p+ = c$
            $max = |c - p|$
            $n = c - p$
        end if
    end for

    //compute the negative pole p-
    $max = 0$
    for all $t \in T$
        $c$ = center of circumscribed sphere of $t$

        if $((c - p) \bullet n < 0)$ AND $(max < |c - p|)$
            $p- = c$
            $max = |c - p|$
        end if
    end for
end for
```

Fig. 8.1: The schema of the poles computation ("•" means dot product).

The algorithm complexity of this step is O (cN), where N is the number of points and the constant c depends on the local tetrahedra configuration around each point. The constant has normally a low value, even though the value can be sometimes very high. For example, if we imagine the sphere with a point in the centre, then the number of tetrahedra coincident with this point is higher than N but it holds only for this point. On average the number is low.

8.4. Average normals

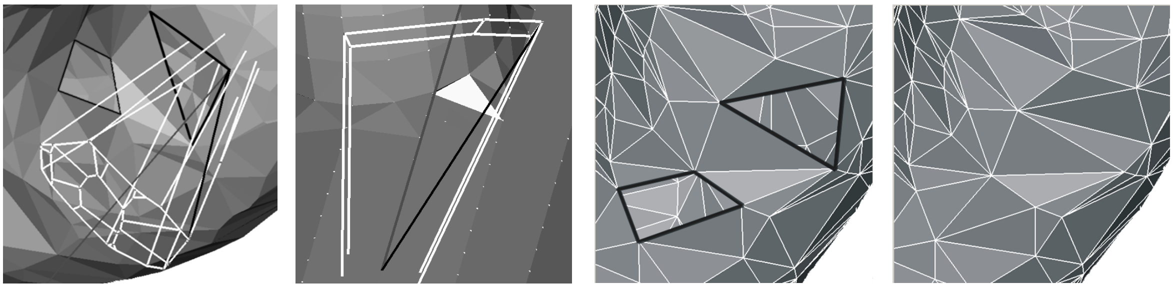
During the algorithm testing, we made the following observation. For sufficiently dense sampling the Voronoi cell is thin and long, the pole is nearly orthogonal to the surface. But in the cases, when the surface is not well sampled or has boundaries, the deviance between the estimated normal vector (vector to the positive pole from the sample point) and the real surface normal can be big. It is because the Voronoi cell in these places does not satisfy the condition to be thin.

So a simple improvement was made. Instead of computing only the farthest vertex (the positive pole *p+*) and taking *p+ - p* it as a normal vector, we take this vector as the normal vector of a temporary plane and we sum the vectors from the point *p* to each Voronoi vertex which lies in the same halfspace as *p+* (see Fig. 8.2 for details). We do not normalize the summed vectors because this would make the sum weighted. If we normalized vectors, then each vector would have the same weight in the resulted sum. But the vectors from the point to the poles which are far away (far from the surface) have better direction than near poles, so we have to give them bigger weight (and the weight of the vector is its length).

```
input:        the point p, its Voronoi vertices and estimated normal n
output:       the average normal vector for point p

plane = plane equation (estimated normal n, point p)
average = (0, 0, 0)

for each Voronoi vertex v of a Voronoi cell of a point p
      if v lies in the same half plane as p+
                   average = average + (v - p)
end for
```

Fig. 8.2: The schema of the average normals computation.

Our implementation shows that average pole technique brings better results than working only with positive poles especially in the cases presented before. In Fig. 8.3 some examples are shown, Fig. 8.3a) and Fig. 8.3b) show a part of the reconstructed surface with a Voronoi cell and a computed average normal. The direction of the average normal is more precise than the normal vector from point *p+* to *p*. Fig. 8.3c) is a part of a reconstructed surface with the original normal vectors computation while Fig. 8.3d) presents the same part reconstructed using average normals where no holes appear.



a) *b)* *c)* *d)*

Fig. 8.3: a), b) Two examples of Voronoi cells with original estimated normal vectors (black lines) and average normal vectors (gray line), c) a part of a surface reconstructed using original normal estimation with holes in the surface, d) the same part of a surface reconstructed using average poles, the holes disappear.

8.5. Surface triangles selection

After the poles and normals computation (no matter how the normals were estimated-using poles or average normals), we can extract from the set of triangles contained in the tetrahedronization the surface triangles using the test described in section 4.4. The triangle (if it is not on the convex hull) is shared by two tetrahedra so we have to be careful and not to test a triangle twice.

We take one tetrahedron after another and for all four faces, triangles, we compare whether the opposite tetrahedron sharing the same face has been processed. This test can be simply implemented by looking to the number of tetrahedron because all tetrahedra are stored in an array. If the number is smaller than the number of currently processed tetrahedron, the shared triangle was tested before and we can continue with the next tetrahedron, otherwise the surface test for this face is computed.

The surface test for a triangle f is computed as follows: because we know which two tetrahedra share the triangle f, we can easily compute the dual Voronoi edge e (with the vertices w_1, w_2) of this triangle. It is the edge between the centres of the tetrahedra's circumscribed spheres (recall Fig. 4.8). Then for each triangle vertex p the angles α, β are calculated. The angle α is defined as the angle between the vector (w_1, p) and the normal n_p, the angle β as the angle between the vector (w_2, p) and n_p. If the angle interval $\langle \alpha , \beta \rangle$ intersects the interval $\langle \pi/2-\theta , \pi/2+\theta \rangle$ (where θ is the input parameter) and this condition holds for each vertex p of the triangle f, then the triangle is on the surface (see for section 4.4 for details).

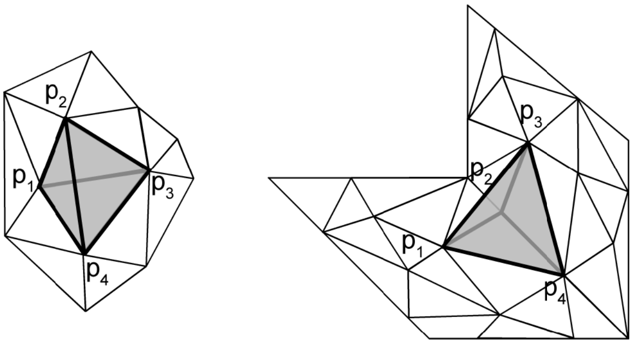
In the case that the triangle f is on the convex hull, no other tetrahedron shares this triangle and we have only one vertex w_1 of the dual edge e (the Voronoi cell is not closed and the edge e goes from the point w_1 to infinity). The surface test can be simplified and we mark the triangle as a surface triangle if the angle between the vector from w_1 to p and the normal n_p at each point p of the triangle is less than $\pi/2 + \theta$, the code is shown in Fig. 8.4.

8.6. Manifold extraction

The triangles, which pass the previous test are marked as a surface but they do not form the manifold yet. There can be more than two triangles incident on some edges or some triangles may be missing on the places of local discontinuity. For example, very flat tetrahedra in a smooth part of the surface (Fig. 8.5a) or tetrahedra on the surface edge (Fig. 8.5b) may have all faces marked as surface triangles. The number of overlapping triangles differs from model to model and depends on the surface smoothness. For a smooth surface it is in tens percent and when the surface is rough, the rate decreases.

```
input:      the tetrahedronization of the input set
output:     the set of primary surface triangles

for each tetrahedron t
    for each triangle f of a tetrahedron t
        //test whether the triangle f has been processed
        $t_{op}$ = tetrahedron neighbouring to t over f
        if ($t_{op}$ number < t number)
            continue
        end if

        //compute vertices $w_1$, $w_2$ of the dual edge to triangle f
        $w_1$ = center of circumscribed sphere of t
        if $t_{op}$ <> NULL
            $w_2$ = the center of the circumscribed sphere of $t_{op}$
        end if

        //test the triangles
        correct = 0
        for each vertex p of the triangle f
            $\alpha = \angle(w_1 - p, n_p)$

            if $t_{op}$ <> NULL
                $\beta = \angle(w_2 - p, n)$
                if $(\alpha, \beta)$ intersects $(\pi/2 - \theta, \pi/2 + \theta)$
                    correct++
                else break
                end if
            else    if $\alpha < \pi/2 + \theta$
                    correct++
                end if
            end if
        end for

        if correct = 3
            mark f as a surface triangle
        end if
    end for
end for
```

Fig. 8.4: The scheme of the primary surface triangles computation.



a) b)

Fig. 8.5: Two examples where overlapping triangles may appear, a) a flat part of the surface, b) a part with a sharp edge. One pair of triangles is $p_1p_2p_3$ and $p_1p_3p_4$, the second pair is $p_1p_2p_4$ and $p_2p_3p_4$.

That is why the surface extraction step must be followed by a manifold extraction step. The input to the manifold extraction step is just the set of triangles. Manifold extraction step is independent of the reconstruction method, therefore, it could be combined with other algorithms than CRUST. We have developed our own algorithm because the manifold extraction methods were explained very briefly in existing papers, however, this step is important. Our approach uses the priority queue where the triangles are sorted according to the angles of neighbouring triangles.

The preprocessing step of the extraction is creation of two structures which help in the phase of triangle neighbours searching. The first structure is a list of all triangles incident to each point, see an example in Fig. 8.6. The algorithmic complexity of the structure creation is $O(T)$, where T is a number of all triangles in the primary surface.

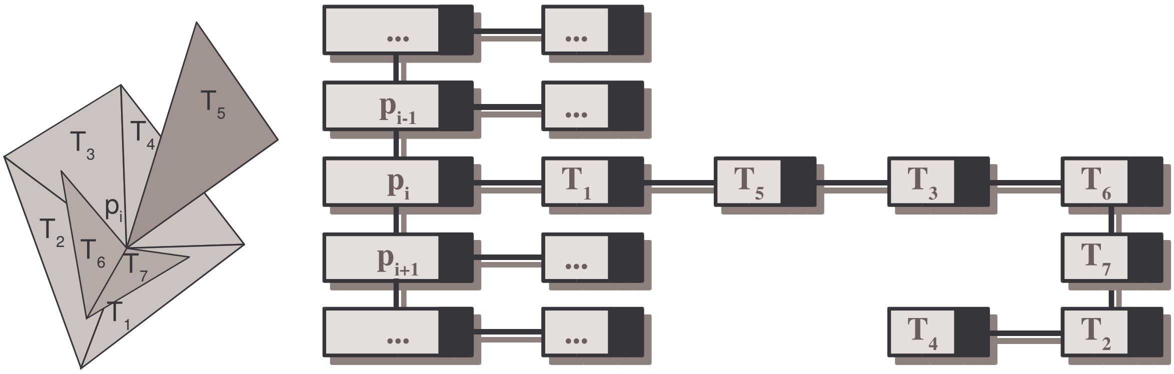


Fig. 8.6: An example of the data structure where each point p_i has a list of its incident triangles T_i.

The other structure is created using the previously described structure and is called a multiple neighbours list. It is a list of triangles, where each triangle has pointers to the incident triangles at each edge. As the primary surface extraction does not ensure that the marked triangles form a manifold, then on one triangle edge more triangles than two can incide, see example in Fig. 8.7. These two structures are used for fast triangle location.

The structure creation has algorithmic complexity $O(T^2)$, where T is the number of triangles, because for each triangle we have to look for its neighbours in the whole mesh. But we can use the previously presented structure and look just for those triangles that are incident with the triangle vertices. Then the algorithmic complexity decreases to linear $O(cN)$, where N is number of input points and c is a constant, which depends on the mesh complexity. For uniformly sampled data the number of incident triangles for every point is from four to eight and for nonuniform data this number is not too big, either.

We can start now with the extraction. First we have to find the starting triangle where our extraction begins. The starting triangle is the first found triangle which have on each its

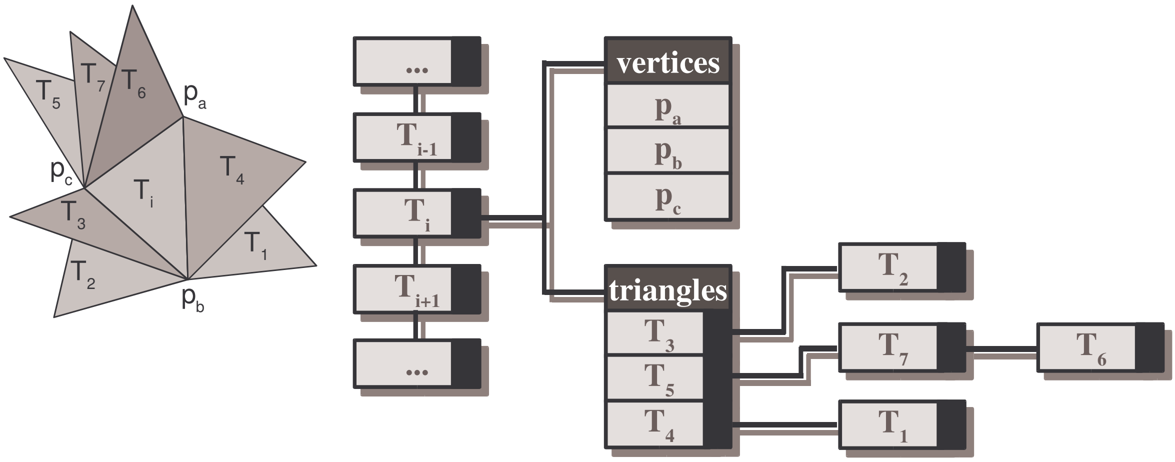


Fig. 8.7: An example of the structure which contains for each triangle the list of all possible triangle neighbours on its edges.

side only one neighbouring triangle. When we find such a triangle we mark it as a manifold triangle and we add its neighbours to the priority queue according to the angle between the normal vectors of the starting triangle and the neighbours. The neighbour whose angle between its normal and the normal of the starting triangle is the smallest is put to the beginning of the queue.

Then we take the first triangle in the queue and we have to choose next triangles on the non-processed edges where the extraction can continue. Fig. 8.8 shows the situation when some triangles (in dark gray) have already been extracted and we have to extract another triangle where the surface (manifold) continues at one edge of the triangle T_1. Three triangles T_2, T_3 and T_4 exist there (candidates for the surface) and we have to choose which of them is the correct one. For the recognition of the correct one the direction of the triangle normal is important. The direction is inherited from the previously extracted triangle, the direction of the starting triangle is estimated using the normal vectors of the starting triangle vertices.

There are three strategies how to choose the correct triangle:

- the smallest triangle angle
- the first tetrahedron found
- the shortest edge length

The first two tests are described in [Dey01d]. We assume that the triangle T_1 has been already accepted to the manifold and is correct. First we have to orientate the triangles T_2, T_3, T_4, see Fig. 8.8. As we need to accept the triangle nearest to the correct surface on the

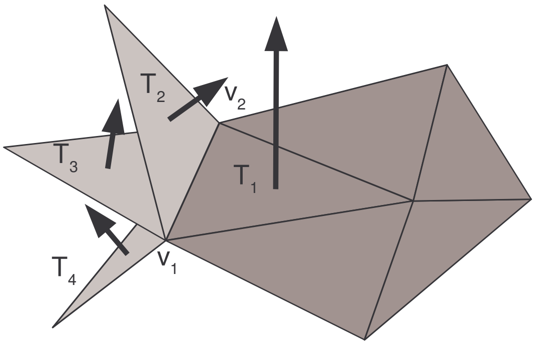


Fig. 8.8: The dark gray triangles have been already extracted, the light gray triangles T_2, T_3, T_4 are incident on one edge to the triangle T_1 and one of them has to be chosen.

edge v_1v_2 , we have to take the one whose angle with the correct triangle T_1 on the edge v_1v_2 is the smallest. That is the method of the smallest triangle angle.

This method is numerically unstable on flat surfaces, because the angles between triangles are very small. But we can use the tetrahedra that are incident with the edge v_1v_2 (Fig. 8.9). As we know tetrahedron neighbours, we can walk through these neighbours on the edge v_1v_2 . We start with the tetrahedron whose one face is the triangle T_1 and which lies in the direction of the triangle T_1 normal. Then we walk through the neighbours and we choose the first marked triangle we find. The disadvantage of this and previous approach is that both approaches prefer the angle of the adding triangle. This is problematic in the case that the surface is not good sampled. Then the angle has not to be the right criterion.

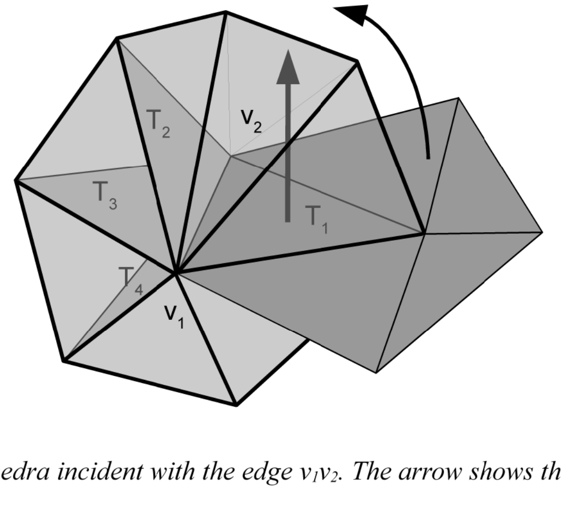


Fig. 8.9: The set of tetrahedra incident with the edge v_1v_2. The arrow shows the direction of walk.

For our approach we use the third method. This method assumes that the triangle must be small to create a correct surface, so we take the one, which has the smallest edge length (the minimum of length of the T_2, T_3 and T_4 triangle edges). The biggest advantage of this method is that it is very simple for computation and does not prefer angles. Although it is a heuristic, we did not find any problem with it.

Always when we choose a next manifold triangle, we add it to the priority queue. All points which were used in extracted triangles are marked with a flag. When we are not able to continue the extraction because all extracted triangles are connected together or there is no triangle left, we have to look at the points and if the flag of all points is set, the extraction finishes. In other case there are more unconnected parts of surface (or more objects) and we continue from the beginning with the unprocessed points. The schema of the manifold extraction can be found in Fig. 8.10.

```
input:      primary surface triangles
output:     manifold triangles

//set all points flags to false
for all points p
      set p.flag to FALSE
end for

//extraction of the manifold
for all points p
      //try to find starting triangle and put it to the queue
      if NOT p.flag AND there is some starting triangle t around p
            add t to queue

      //for all triangles in the queue, extract the neighbours
      while queue NOT EMPTY
            for all unprocessed edges $e \in t$
                  $t_{new}$ = get correct triangle at e

                  //put the triangle to the queue
                  if $t_{new}$ <> NULL
                        add $t_{new}$ to the queue according to the angle
                              between $t,t_{new}$

                        for all points $s \in t_{new}$
                              s.flag = TRUE
                        end for
                  end if
            end for
      end while
end for
```

Fig. 8.10: The schema of the primary surface triangles computation.

The advantage of this manifold extraction method is that it has not big memory requirements and it is very fast, however, we cannot compare with the time of the original algorithm due to different platforms. In our older paper [Varnuska03] where we presented the manifold extraction step we used a little different approach which was built on the breadth first search. The root of the search tree was the first starting triangles and other levels of the extraction tree were created from previous ones by appending triangles on the non-processed edges. The advantage of the newer method is that the triangles are extracted first on the flat parts of the surface while more complicated parts are extracted at the end of extraction.

8.7. Tetrahedra prefiltering

During testing the manifold extraction, we have detected some problems, which may occur. We already mentioned that the CRUST algorithm has very good results for smooth surfaces. However, even with datasets of smooth objects, sometimes small triangle holes appear in the reconstructed surface. It is not a problem to find and fill them in the postprocessing step, but the question is why they appear.

Each tetrahedron has four faces - triangles. The CRUST marks them whether they belong to the set of the primary surface *T*. We have found that the triangle holes appear in the smooth places where very flat tetrahedra lie whose three faces may be marked as surface triangles. See Fig. 8.11a) for an example: the dark gray triangles are already extracted and we are looking for the triangle neighbour on the bold edge of the triangle T_1. The light gray triangles are marked triangles from one tetrahedron (there are three overlapping triangles), two of them are incident with the bold edge of triangle T_1 and we have to choose only one of them. When we select the bad triangle then in the next step of extraction a triangle hole occurs. Fig. 8.11b).Fig. 8.11c) shows the correct configuration.

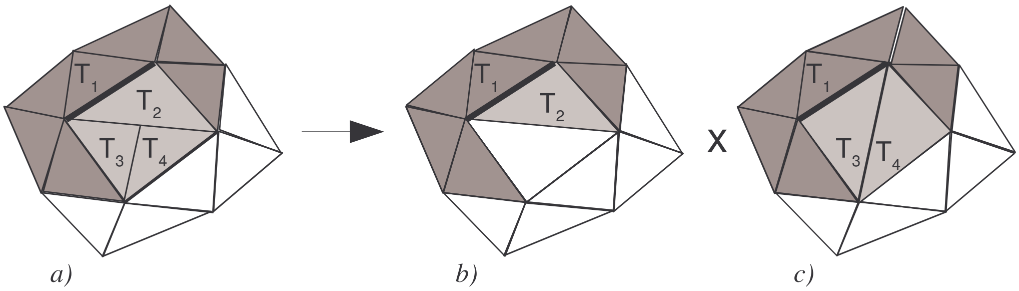


Fig. 8.11: Two configurations in the manifold extraction of the tetrahedron with three marked surface triangles, a) the initial status, on one edge of the triangle T_1 there are two connected triangles belonging to one tetrahedron, b) the wrong choice, c) the correct choice.

In order to avoid such situations, before the manifold extraction step it is necessary to detect the tetrahedra, which have three marked faces, and remove one overlapping face. So we take one tetrahedron after another and mark surface triangles (faces) using the CRUST algorithm. If there are three marked faces on one tetrahedron, we preserve only those two faces whose normals make the smallest angle (the tetrahedron is flat, so the triangles on the other edges make together sharp angle), the third face is deleted. We have to be careful with the orientation of the triangle normals, they have to be oriented in the direction of the tetrahedron center of gravity (see an example in Fig. 8.12). The best configuration is in Fig. 8.12d), the angle between triangle normals incident to the edge is the smallest (the dot product of the normals is close to one, in Fig. 8.12b) and Fig. 8.12c) is close to minus one).

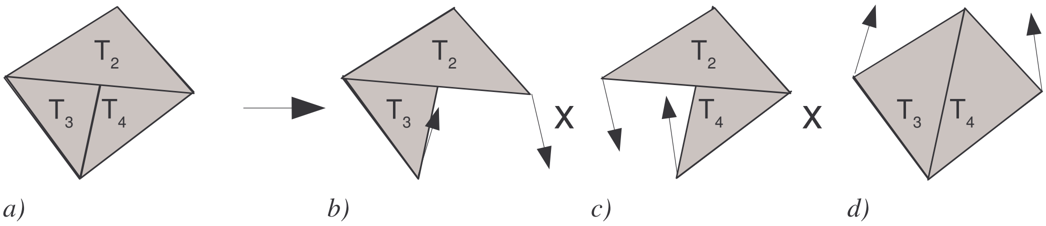


Fig. 8.12: a) The tetrahedron with three marked faces T_2, T_3 and T_4 and three possibilities b), c) and d) which two triangles to choose. Arrows correspond to the triangle normals.

This approach converts tetrahedra with three marked triangles to tetrahedra with two marked triangles. We can use it to filter tetrahedra with four marked triangles, too. Besides removal of problematic places, the prefiltering approach reduces the number of triangles in the primary surface. After converting all tetrahedra with four and three marked faces to tetrahedra with two good faces, the set of primary surface triangles is ready for extraction.

When the reconstruction without the prefiltering improvement ran, several triangle holes appeared. The number of triangle holes was not too high but when looking closer to the reconstructed object, it can disturb the visual perception and the reconstructed object does not form a manifold. We have tried also the Dey's COCONE algorithm (in his implementation) and the triangle holes appeared there, too (Fig. 8.13a). After applying prefiletring (Fig. 8.13b) and Fig. 8.13c), the situation changed and our algorithm was able to reconstruct the surface with much less triangles holes. Sometimes a triangle hole still appears but the cause is different, the missing triangles were not chosen to belong to the surface.

The next consequence of this prefiltering improvement was a reduction of the amount of triangles in the primary surface. Nine datasets were tested (see Tab. 8.1, the row "points" is the number of points in the tested dataset) and the number of redundant triangles, which are necessary to be removed from the triangulation, measured. The row "without" presents the number of redundant triangles marked as surface triangles without the prefiltering applied. The number of redundant marked surface triangles computed with the help of the prefiltering is in the row "prefilter". The last row presents the rate in percents of the number of marked triangles before applying prefiltering and the number of triangles after prefiltering. It can be seen that 38-99 percent of the redundant triangles are removed by prefiltering.

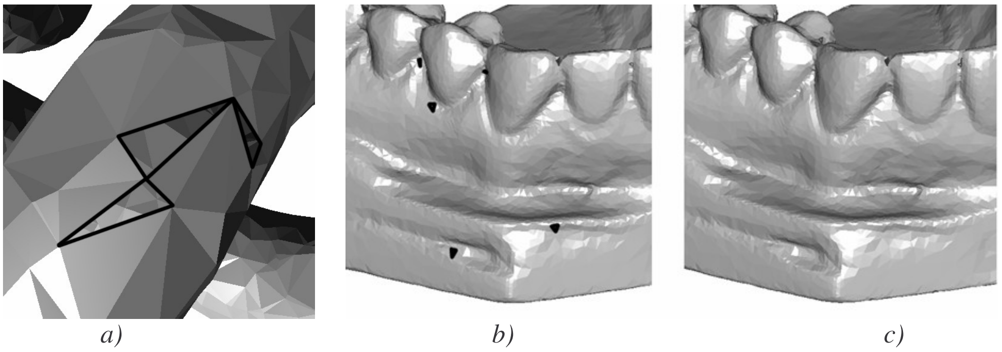


Fig. 8.13: a) The detail to the surface reconstructed by Dey's COCONE, triangle holes in the surface are black, b) shows the reconstruction by our algorithm without the help of prefiltering (missing triangles are black) and c) with the help of prefiltering.

	bone	bunny	teeth	engine	nascar	mann	knot	hypsheet	x2y2
points	68537	35947	29166	22888	20621	12772	10000	6752	5000
without	8106	11937	4642	9835	992	926	2017	1451	358
prefilter	111	71	145	33	297	54	70	898	122
% rem	98	99	96	99	70	94	96	38	65

Tab. 8.1: The datasets used for testing of prefiltering, the number of points in each dataset is in row “points”, number of triangles marked as surface without prefiltering (row "without"), number of triangles with prefiltering (row "prefilter") and the percent rate of the removed triangles using the prefiltering in the row ("rem").

9. Postprocessing steps

The surface postfiltering steps process the output triangle mesh from the reconstruction algorithm and they try to repair bad triangles configurations, to delete big boundary triangles and to retriangulate possible holes.

9.1. Triangle mesh filtering

When we have the data, which is not uniformly sampled, with some noise or some features missing due to undersampling, the manifold extraction may fail because the CRUST selects bad surface triangles and unwanted triangle configurations occur. Fig. 9.1 shows an example. This detail is taken from a dataset which is not uniformly sampled and contains some noise. The highlighted part presents an erroneous place after the manifold extraction, missing and overlapping triangles.

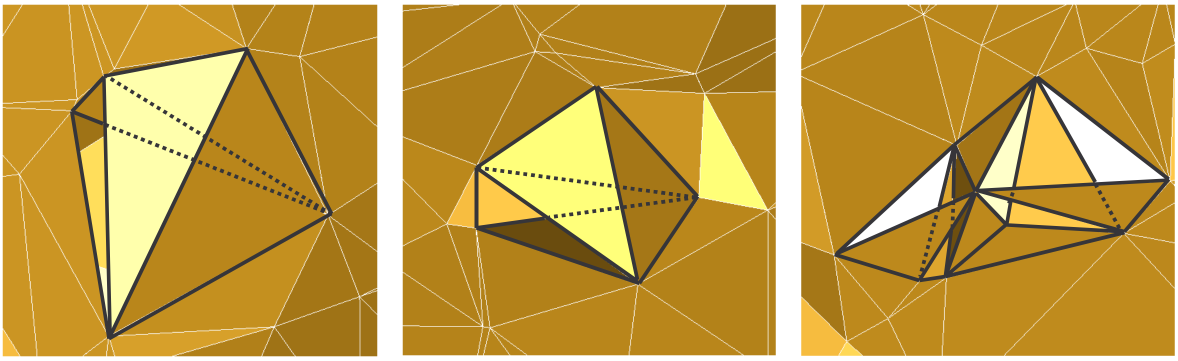


Fig. 9.1: Some parts of the reconstructed surface with errors after manifold extraction.

Missing and overlapping triangles appear there due to bad normal vectors arisen from the incorrect shape of Voronoi cells. We have analysed triangle fans around the points obtained after the reconstruction. These configurations may be detected using an undirected graph. The nodes of the graph correspond to the fan triangles. A graph edge *e* exists in the graph if the nodes of the edge *e* correspond to neighbouring triangles (see Fig. 9.2).

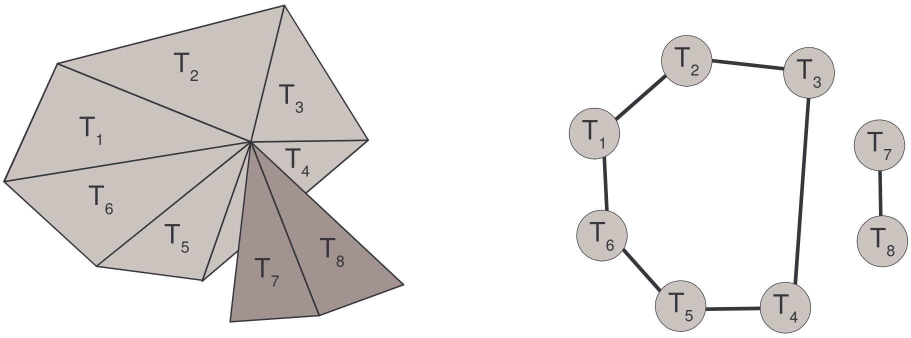
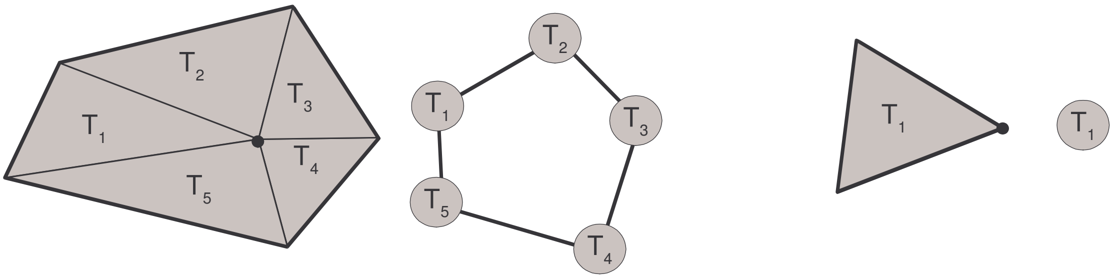


Fig. 9.2: An example of a fan configuration and a graph corresponding to the fan.

Two acceptable configurations of the triangle fan exist. Fig. 9.3a) presents a full fan around a point. It can be detected as the graph cycle which contains all nodes. Fig. 9.3b) is just one single triangle, which can appear, e.g., on the corners of the surface with a boundary. Detection of these configurations is simple.



a) *b)*

Fig. 9.3: a) Full fan and one graph cycle corresponding to the fan, b) one triangle with its graph.

Other configurations are incorrect and some triangles have to be deleted. When we are able to find one cycle in the graph, we can delete all triangles whose graph nodes are not included in the cycle. The most common configuration is shown in Fig. 9.4a), one full triangle fan with one separated triangle. The Fig. 9.4b) is some hypothetic situation with more than one cycle but we did not find any occurrence and it looks practically impossible to extract this configuration.

The configurations presented in Fig. 9.5 are more problematic. When there are only subfans (we denote the fan as subfan if it does not form a cycle), searching a good fan configuration is not so simple and it will be explained in the following text. Here we cannot avoid utilization of the normal vectors (we are testing these configurations in the projecting plane) which can bring problems. The normal vectors are correctly estimated only on smooth parts of the surface but the problematic configurations of the fans appear on the places where the sampling is not correct.

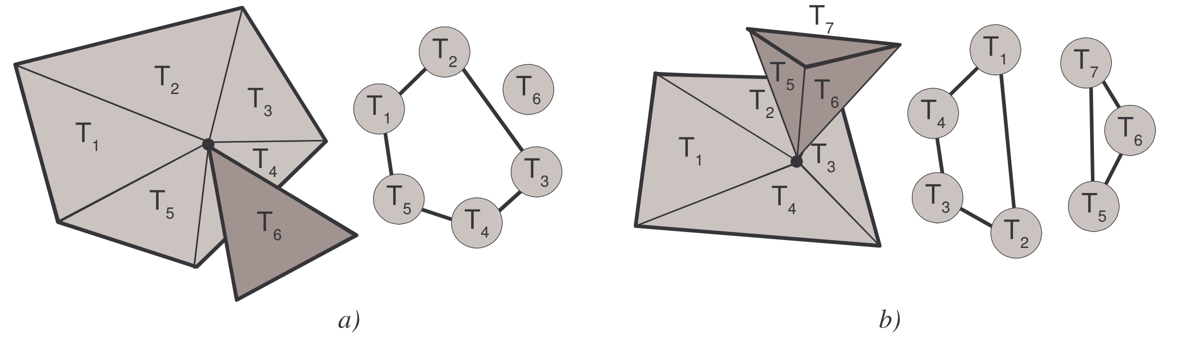


Fig. 9.4: a) One full fan with another separated triangle, b) more full fans.

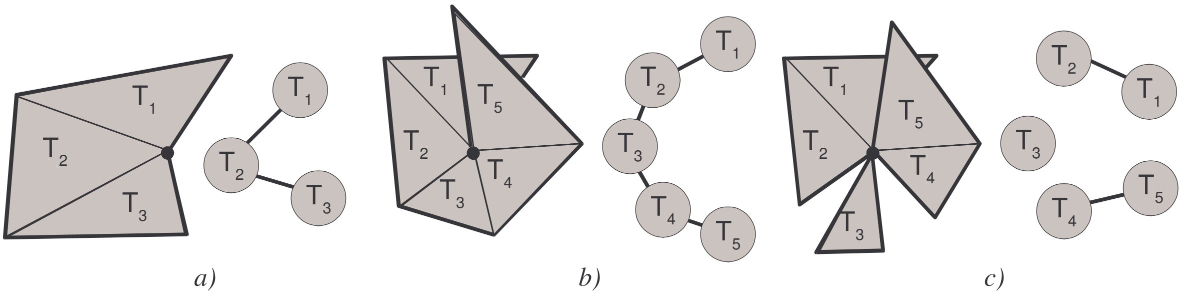


Fig. 9.5: Some fan configurations formed only by subfans, a) one subfan without overlapping triangles in a projection, b) one subfan with overlapping triangles in a projection, c) more subfans.

All the triangles around the fan are projected to the plane given by the point (center of the fan) and its normal vector (although the normal direction probably is not correct). The detection is more simple for the configuration in Fig. 9.5a) and Fig. 9.5b) than Fig. 9.5c) because the triangles create only one subfan. When the sum of angles of the projected triangles (angle between two edges incident with the point) has less than 2π in the projection (Fig. 9.5a), the configuration is accepted and no changes in the triangle mesh is done. When it is more (Fig. 9.5b), we delete triangles from one end of the subfan until the angle is less than 2π. Fig. 9.5c) represents the worst case, a set of more subfans. This configuration occurs fortunately very rarely and we remove all triangles except the subfan with the largest sum of angles.

In Fig. 9.1 three examples of bad triangle fan configuration were shown. Fig. 9.6 shows the reconstruction of the same parts of the surface with the postfiltering applied. The overlapping "flying" triangles disappear and the remaining triangle holes are filled with triangles. The triangle holes are filled automatically because they are easy to find and to triangulate. More complicated holes are retriangulated in the last step of the postprocessing steps.

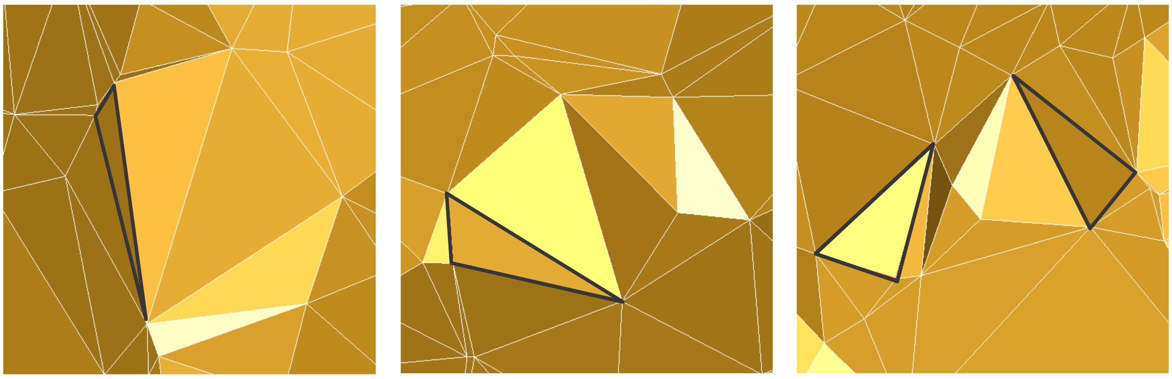


Fig. 9.6: The same parts of the surface as in Fig. 9.1 after applying prefiltering. The black triangles present the triangle holes formed after postfiltering which were filled with triangles.

The problem with overlapping triangles appear in the COCONE algorithm too, we found some bad fan configurations on the reconstructed surface (Fig. 9.7). In this case it was not possible to reproduce Fig. 9.7 for comparing with our algorithm because although the algorithms are similar, the code is not the same and the reconstructed meshes differ a little for the same models.

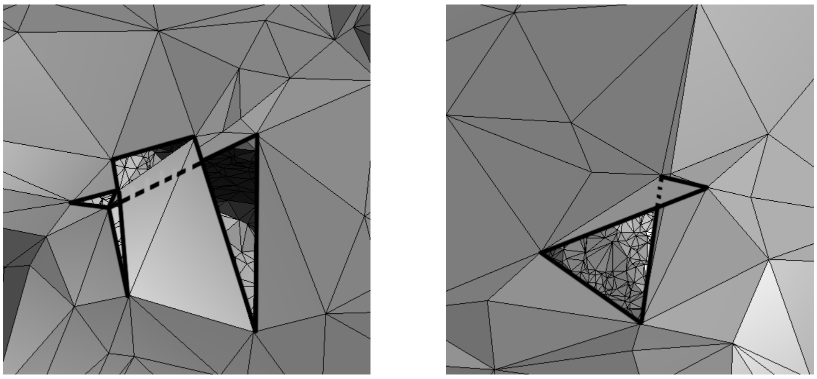


Fig. 9.7: The overlapping triangles in the surface reconstructed using COCONE algorithm.

9.2. Boundary filtering

When the surface contains a boundary, the CRUST has problems with its recognition and it marks the boundary triangles as surface triangles. This is a problem of all algorithms, there is no chance to find if some place represents a boundary or just a local undersampling and we are left to heuristics. Dey presented a heuristic algorithm, which was mentioned in Chapter 4, for a recognition whether a point lies on a boundary or not.

We present now another heuristic approach based on the observation of the boundary triangles, see Fig. 9.8. There are two examples of a surface, where boundary triangles appear incorrectly. Fig. 9.8a) presents the case when the boundary triangles are perpendicular to the surface triangles while Fig. 9.8b) show the case where boundary triangles are parallel to the surface triangles. When we look closer at the figures, the length of the edges of boundary triangles differ from the length of surface triangle edges. To avoid the situation when the surface is not uniformly sampled and the length of edges incident to a point differ, we developed an adaptive criterion based on the edge length and the angle between the point and incident triangle normals.

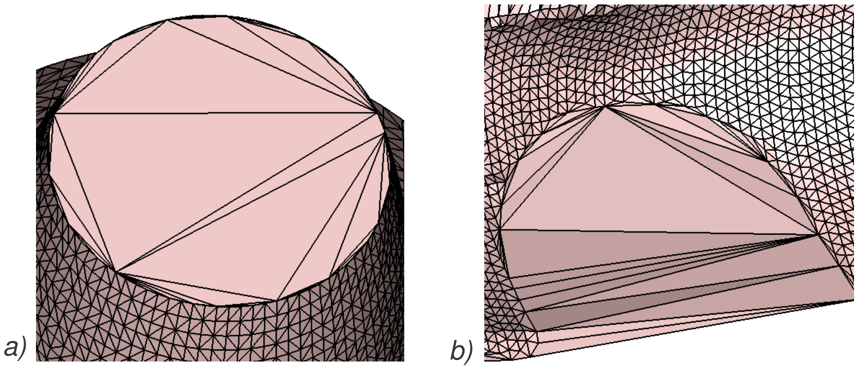


Fig. 9.8: Two examples of badly triangulated boundary, a) the boundary triangles perpendicular to the surface triangles, b) the boundary triangles parallel to the surface triangles.

The boundary test is computed as follows. For each point *p* the set *E* of all incident edges is created. One edge *e* is chosen as referential. All triangles whose both edges incident to the point *p* are longer than the length of the referential edge multiplied by some variable *m* are filtered out. This variable depends on the angle of the triangle and point normals. The multiplicator *m* is computed :

$$m = c_{be} + c_{se} |n_p \cdot n_t| \qquad (18)$$

The constant c_{be} is the maximal allowed length of the triangle edge when the angle between the normal of the triangle n_t and the normal at the point n_p is 90° (then the dot product of these normals is zero). When we compute *m* on a flat smooth part of the surface, the dot product will be one and the multiplicator *m* is the sum of c_{be} and c_{se}. These constants were set experimentally and almost for any data sufficient results were achieved for c_{be} equal to 2.0 and c_{se} equal to 5.0 (e.g., on the boundary where boundary triangles are perpendicular to the surface triangle, *m* is 2.0 and when the boundary triangles are parallel, *m* is 7.0).

The most important question is how long should be the referential edge. First we have tried to take the median of the edges lengths as the referential edge, but it did not give good results because at one boundary point there can be more boundary triangles than surface triangles (see Fig. 9.9a) and the boundary triangles were not filtered. Then we have tried to take the smallest edge in the set *E* (Fig. 9.9b). But some datasets have due to errors in the scanning process some points very close together, so more triangles were filtered than we wanted. Another criterion, and we have used it for a long time, works better – when the referential edge *e* is taken as the third smallest edge. It is a heuristic criterion and it is built on the observation that even when there are some points mutually very close, the third shortest edge represents the length of some "normal" triangle.

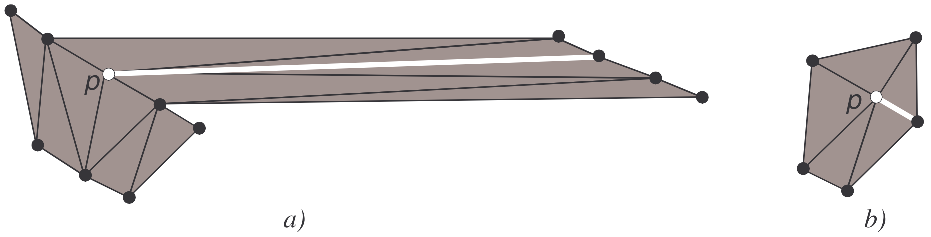


Fig. 9.9: a) The referential edge is taken as the median of all edge lengths (the white edge), b) the referential edge is the shortest edge.

Although the last presented approach for referential edge choosing worked well for many datasets, we have found some cases where it failed and more triangles were incorrectly deleted. According to our experiments, the referential edge is now chosen as the median of all edges coincident with the point *p* together with all edges coincident with the point *p* neighbours. Such a selection is more accurate and all boundary triangles were correctly removed. For an illustration, in Fig. 9.10 the edge printed in white is the referential edge for point *p* selected from all edges printed in grey, the grey points are the neighbours of the point *p*.

On the beginning of the boundary filtering step the boundary stack is filled with all points. The stack is an array of points which are ready for processing. Then the top point from the stack is filtered with above described procedure. When some triangles are recognized as boundary triangles and deleted, the triangles vertices (except the current vertex) are put back to the stack in the case that they were filtered in some previous filtering step. Each point has one flag which says whether the point was filtered or not. We continue in this way till the whole stack is empty, see Fig. 9.11 for the scheme.

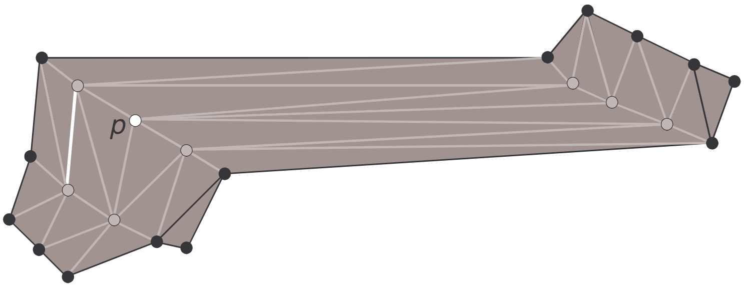


Fig. 9.10: Current selection of referential edge (white edge) of a point p, grey edges are the edges from which referential edge as a median was selected, points in grey are the neighbours of point p.

```
input:      the triangle mesh
output:     the modified triangle mesh without overlapping triangles

$c_{be}$ = 2.0
$c_{se}$ = 5.0

//initialize the boundary filtering step
for each point p from the triangle mesh
      p.filtered = FALSE
      add p to stack
end for

//make boundary filtering
while the stack is not empty
      p = point from the top of the stack
      p.filtered = true

      //choose the referential edge (described in the text)
      $n_p$ = estimated point p normal
      E = all edges incident with p
      e = choose the referential edge from E
      $l_e = |e|$

      //filter the triangles incident with point p
      for each triangle t incident with the point p
            $n_t$ = triangle t normal
            $v_1$ = the first vertex of t different from p
            $v_2$ = the second vertex of t different from p
            $m = c_{be} + c_{se} |n_p \cdot n_t|$

            //if the triangle is boundary, delete it
            if $(m|v_1 - p| > l_e)$ AND $(m|v_2 - p| > l_e)$
                  delete t

                  if $v_1$.filtered
                        add $v_1$ to the stack
                        $v_1$.filtered = FALSE
                  end if
                  if $v_2$.filtered
                        add $v_2$ to the stack
                        $v_2$.filtered = FALSE
                  end if
            end if
      end for
end while
```

Fig. 9.11: The scheme of the boundary filtering.

In Fig. 9.8 the reconstruction of some surface with a boundary was shown. After applying the boundary filtering, better reconstruction was obtained, see Fig. 9.12. Both cases, when the boundary triangles are perpendicular and parallel with the surface, were correctly reconstructed.

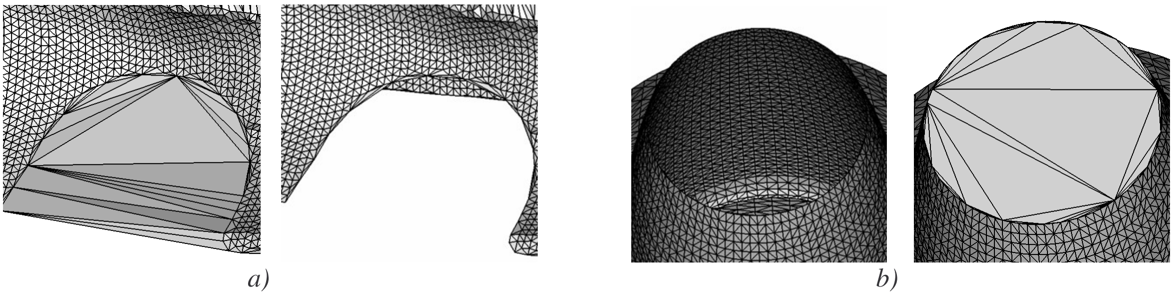


Fig. 9.12:a) An example of the boundary filtering when the boundary triangles are parallel with the surface triangles, b) when the boundary triangles are perpendicular to the surface triangles.

For comparison we tried to reconstruct the same objects by COCONE. The reconstruction worked well for the objects in Fig. 9.12b) and the reconstructed models were almost the same. The reconstruction of the object as in Fig. 9.12a) fails and many bad boundary triangles appear there. Fig. 9.13 shows the whole reconstruction of the object from Fig. 9.12a). Fig. 9.13a) is the reconstruction using COCONE. It is visible that the boundary detection failed. The output of the CRUST followed by our manifold extraction is in Fig. 9.13b), several incorrect boundary triangles occur there mainly in the windows and the wheel parts of the coachwork of the car. When we apply the boundary filtering, the situation is better, see Fig. 9.13c). The highlighted parts around the car trace the boundary contour.

The bottleneck of the heuristic algorithms is always the choice of parameters settings. Because we use here the adaptive setting depending on the local configuration, the choice of parameters described above is useful almost for all data we have. For uniform datasets we also tried successfully to set the constants to $c_{be} = 2.0$ and $c_{se} = 1.0$ but at this moment, we are not able to detect data uniformity automatically.

The car model is uniformly sampled, other test data, nonuniformly sampled, can be seen in Fig. 9.14, Fig. 9.15 and Fig. 9.16. The hypersheet model was a little problematic for both programs (Fig. 9.15). The data is not uniformly sampled, sometimes several points are mutually very close and some surface triangles were missing. The boundary triangles were successfully removed except one. COCONE has a problem with the knot dataset too, some surface triangles were marked as boundary and holes appear. The other data was successfully reconstructed by both programs.

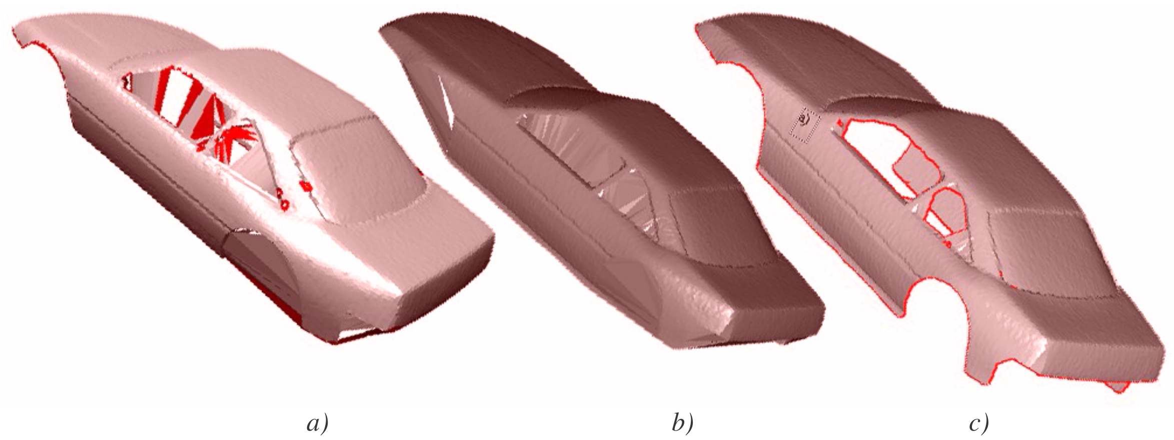

a) b) c)

Fig. 9.13: a) The model reconstructed by the COCONE , detected boundaries are the highlighted parts, b) the same model reconstructed using CRUST without any boundary improvement, c) after applying the boundary filter, highlighted parts are traced boundary edges.

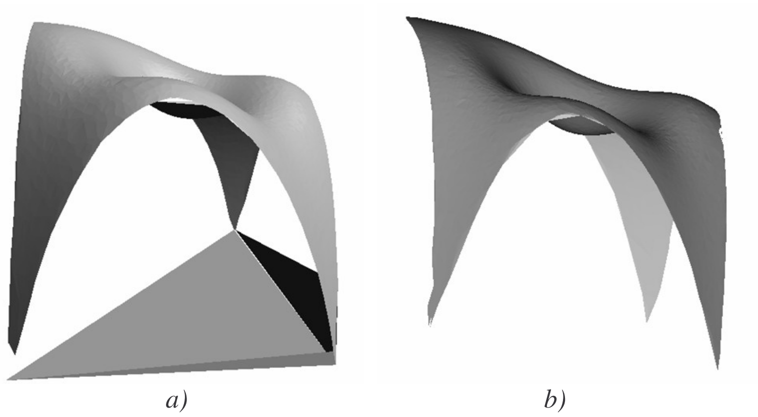

a) b)

Fig. 9.14: a) The function x^2y^2 reconstructed using COCONE, the surface is incorrectly connected with the boundary triangles, b) the reconstruction by our approach.

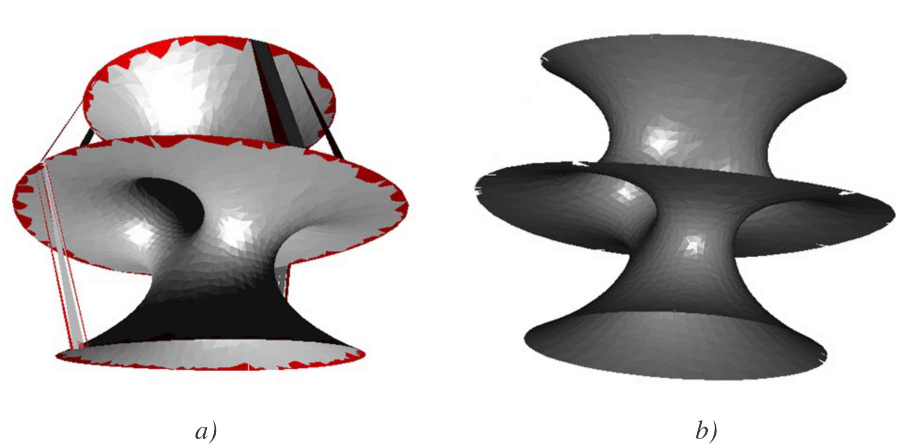

a) b)

Fig. 9.15: a) The hypersheet model reconstructed by COCONE, the boundary is correctly detected but some incorrect triangles appear, b) our reconstruction, almost all boundary triangles were removed but some of surface triangles, too.

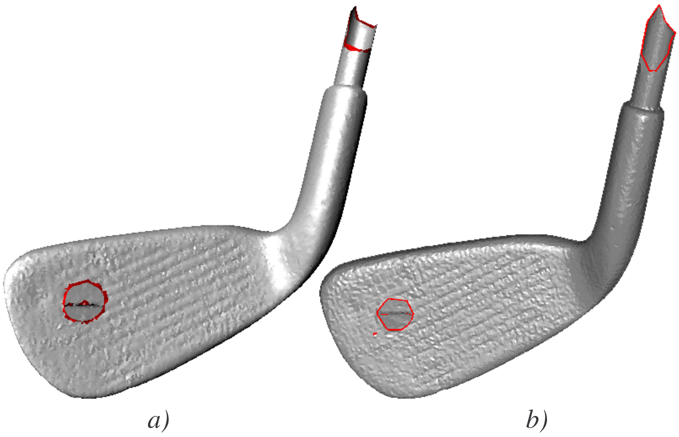


Fig. 9.16: a) The club dataset reconstructed by COCONE, the reconstruction is correct and boundaries triangles were removed, b) the reconstruction by our approach, boundary triangles successfully removed, too.

9.3. Holes filling

The output of the previously presented steps is the surface with possible holes. There are many reasons for the holes appearance described in Chapter 5. There exist some robust algorithms able to repair the incorrectly reconstructed surfaces containing complicated holes and unconnected parts, such as David's et al. volumetric diffusion approach [Davis02] or Emelyanov's bridge approach [Emelyanov04]. The problem is paradoxically with their robustness, they are too complicated to implement them and too robust for simple holes filling. However, reconstructed surfaces in this phase of our approach usually do not contain so complicated holes and most of the holes are quite simple.

In this section we introduce a simple approach able to fill the holes in the reconstructed surface. We assume (and our surface reconstruction algorithm produces such surfaces) that the reconstructed surface is well reconstructed with no overlapping triangles, correctly detected and reconstructed boundaries and with simple holes. We can divide our holes filling approach into two phases, the former is tracing of the holes edges and the latter is filling of the edges with triangles.

Firstly, the holes have to be located in the triangle mesh, we can use the following tracing approach (see Fig. 9.17a). The whole triangle mesh is processed and we look for the triangles without neighbours on some edges. When such a triangle is found (e.g., the triangle with the edge v_1v_2 in the figure) one vertex (v_1) is marked as the starting vertex. Then, we look for the next empty edge (empty edge is the edge associated with one triangle only) around the second vertex v_2 of the starting edge and using this approach the whole hole is found. The problem occurs in the case of point v_4 where more than two empty edges coincide.

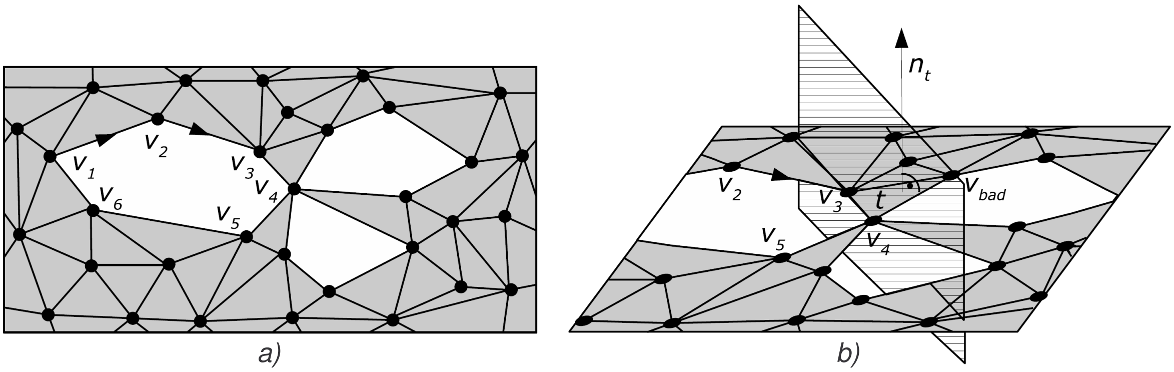


Fig. 9.17:Holes tracing, a) the vertices $v_1v_2v_3v_4v_5v_6$ represent the hole, the vertex v_4 is problematic, b) the plane created using the edge v_3v_4 and a triangle t (with normal n_t) coincident to the edge, the vertex v_{bad} is the vertex with the smallest angle to the edge v_3v_4.

As the traced hole should be as small as possible, we want to select v_4v_5 (not, e.g., the edge v_4v_{bad}, see Fig. 9.17b) as the next edge. We create the plane which separates the space into two halfspaces given by the edge v_3v_4 and the normal vector n_t of the triangle *t*. When there are other edges lying in the same halfspace (given by this plane) as the rest of the traced hole, we take the edge with the smallest angle to the edge v_3v_4. In the other case, when no other edges are in the same halfspace, we select the edge with the biggest angle to the edge v_3v_4.

This approach of choosing next hole edge works amazingly well and we have found only few cases when it did not work correctly, especially in the noisy datasets, where the configurations of holes were awful and it was difficult to decide where to continue. According to our experiments, such problems are common in other reconstruction programs, too.

When the holes tracing is finished, a set of traced holes is created and for each member of this set the holes filling is done. Because we want to fill only small holes and leave the big holes, which represent the boundary, unaffected, there has to be some limit on the hole size. Unfortunately, there is not an exact way how to determine whether the hole is small or not, the user has to have the last word, but the heuristic limit of 50 edges seems to be good enough to separate small holes from big boundary holes. Thus we perform holes filling only for small holes Fig. 9.18a) and for boundary holes Fig. 9.18b) only shape improving is performed (few triangles are added to create better boundary shape).

For the holes filling we use an approach similar to the ear cut algorithm known in polygon triangulation. The polygon, or the hole in our case, is given by the vertices $v_0v_1v_2 \ldots v_{n-1}$ and the ear is the triangle created by the vertices $v_{(i-1)\ \%\ n}\, v_i v_{(i+1)\ \%\ n}$ where “%" means modulo division, in the next text we denote v_i is the same as $v_{i\ \%\ n}$. The main difference is in the fact that polygon triangulation is done in E^2 but in our case in many

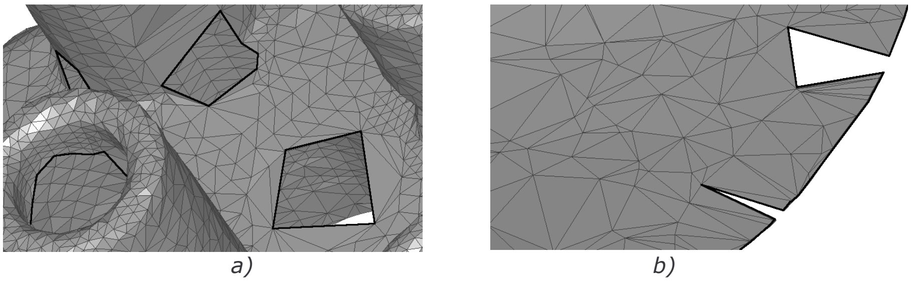


Fig. 9.18: Typical holes, a) small holes which have to be triangulated, b) boundary holes, only a few of triangles have to be added to correct the shape.

cases we are not able to project the holes to the plane due to complicated shapes, so we have to triangulate the hole in E^3. The algorithm is simple, see Fig. 9.19.

```
input:      the triangle mesh and one hole $v_0 v_1 ... v_{n-1}$
output:     the triangle mesh with filled hole

//evaluate each ear in the hole
for $i$ = 0 to $n$ - 1
        evaluate ear $v_{(i-1) \% n}$ $v_i$ $v_{(i+1) \% n}$
end for

//and try to retriangulate the ear
while the hole is not triangulated yet
        $v_{best}$ = -1

        //find the ear with the best evaluation
        for $i$ = 0 to $n$ - 1
                if ($v_i$ has better evaluation then $v_{best}$ AND
                    $\Delta$ $v_{(i-1) \% n}$ $v_i$ $v_{(i+1) \% n}$ is correct)
                        $v_{best}$ = $v_i$
                end if

        //if no good ear was found, exit
        if $v_{best}$ = -1
                exit
        end if

        //put the new triangle to the mesh
        create triangle  $v_{(best-1) \% n}$ $v_{best}$ $v_{(best+1) \% n}$

        //reevaluate the new ears
        evaluate ear $v_{(best-2) \% n}$ $v_{(best-1) \% n}$ $v_{(best+1) \% n}$
        evaluate ear $v_{(best-1) \% n}$ $v_{(best+1) \% n}$ $v_{(best+2) \% n}$

        //remove the ear from the polygonal holes
        remove $v_{best}$ from the hole

end for
```

Fig. 9.19: The schema of the holes filling

The first step in the holes triangulation procedure is the ears evaluation. We have tried three possible approaches how to evaluate an ear - a possible triangle $v_{i-1}\ v_i\ v_{i+1}$ - based on:

- the smallest angle
- the smallest length of the edges
- the smallest neighbours angle

The smallest angle approach computes the angle between the vectors v_{i-1} - v_i and v_{i+1} - v_i using the dot product, see Fig. 9.20a). The smallest length approach computes the sum of distances between ear vertices, thus $|\ v_{i-1} - v_i\ | + |\ v_{i+1} - v_i\ | + |\ v_{i-1} - v_{i+1}\ |$, see Fig. 9.20b). The last approach computes the angles α_1 and α_2 between the triangle normal vector n_1 (triangle coincident with the edge v_{i+1} , v_i), n_2 (triangle coincident with the edge v_{i-1} , v_i) and the ear normal vector n_t, see Fig. 9.20c). Both angles are then multiplied to get the final evaluation (the second possibility is the summarization of angles, but multiplication is better because it prefers ears with both angles small).

After the evaluation procedure the ears are recursively cut depending on their evaluation in the loop. First, the ear with the best evaluation is chosen. In the case that the ear is not correct (when we insert the ear triangle to the triangle mesh, the triangles will overlap, see Fig. 9.20d) we have to choose another one. The correctness is determined using the angles between the existing triangles and the new ear triangle. When we are not able to find a correct ear, the procedure ends and the hole remains triangulated only partially (it happens rarely in very complicated parts of surface, e.g., affected by the noise , when the inserted ear overlaps the existing triangulation). Otherwise, the ear is put to the mesh, the hole is reduced by one vertex and the ears $v_{best-2}\ v_{best-1}\ v_{best+1}$ and $v_{best-1}\ v_{best+1}\ v_{best+2}$ are reevaluated because the hole was locally changed in this place.

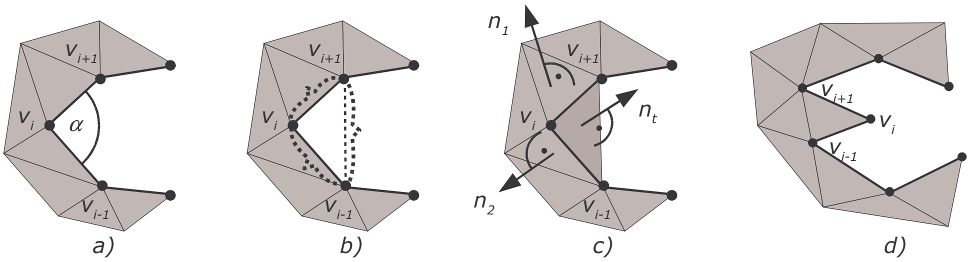


Fig. 9.20:The ear $v_{i-1}\ v_i\ v_{i+1}$ evaluation, a) the smallest angle, α_1 is the angle between the vector v_{i-1} - v_i and the vector v_{i+1} - v_i, b) the smallest length of the edges, c) the smallest angle between neighbours, n_1 is the normal vector of the triangle with the edge $v_{i+1}\ v_i$, n_2 is the normal vector of the triangle with the edge $v_{i-1}\ v_i$, n_t is the normal vector of the ear, d) an invalid ear.

The above described procedure is used for the holes triangulation of both small and boundaries holes, the difference is only in the angle limit when the correctness of the inserted ear is computed. When the boundary holes are triangulated, we use a smaller angle limit, thus the triangle normal of the newly added triangle must have a small difference from the normal vector of the coincident triangles.

We have tried the triangle filling procedure on those datasets whose triangle meshes contain holes after the reconstruction. The holes tracing procedure worked well and it traced correctly the holes with the exception of very noisy datasets. The procedure of holes filling worked correctly. All three evaluation approaches seemed to work, but they produced different results. Fig. 9.21 and Fig. 9.22 show the examples of the reconstruction followed by the holes filling.

The first approach, the evaluation using the smallest angle, produces very often high number of triangles coincident with one vertex (see Fig. 9.21c) or Fig. 9.22c). The reason is the following: when we cut the ear then on the place of cutting the angle becomes smaller than before and the next cutting will continue in the same place. The approach using the smallest edge length seems to be better (see Fig. 9.21d) or Fig. 9.22d), the triangles are not so thin as using the previous approach. But the best results, especially in the places with sharp edges, are reached using the approach with neighbours angles, the inserted triangles adapt to the local geometry, so sharp edges are preserved (see Fig. 9.21e) or Fig. 9.22e).

In Fig. 9.21 is also presented why the user has to say whether the hole should be filled or not. In our case we filled seven circle holes on the top of the object but probably these holes should remain unfilled from the view of the object correctness.

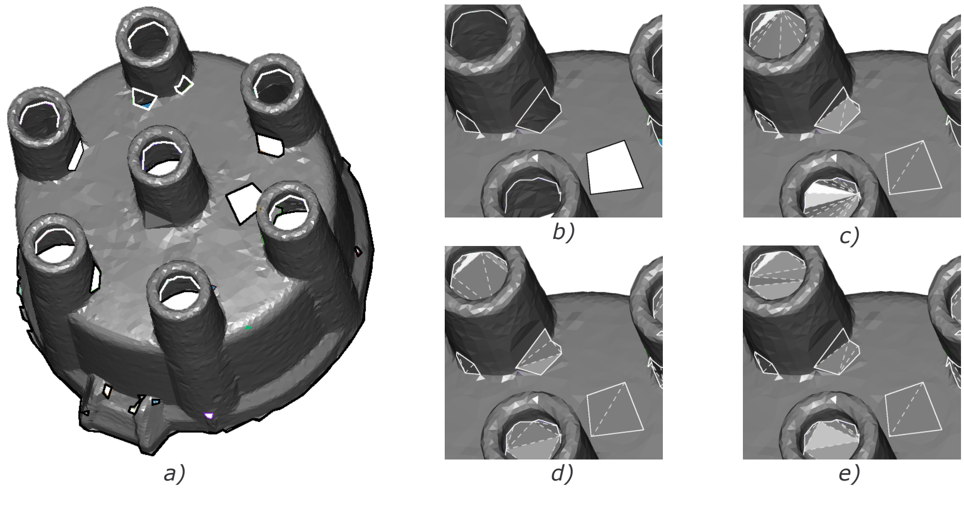


Fig. 9.21: The result of the holes filling, a) the whole mesh with holes highlighted, b) a zoom to one part with holes, c) the smallest angle filling, d) the smallest edge length filling, e) the smallest neighbours angle.

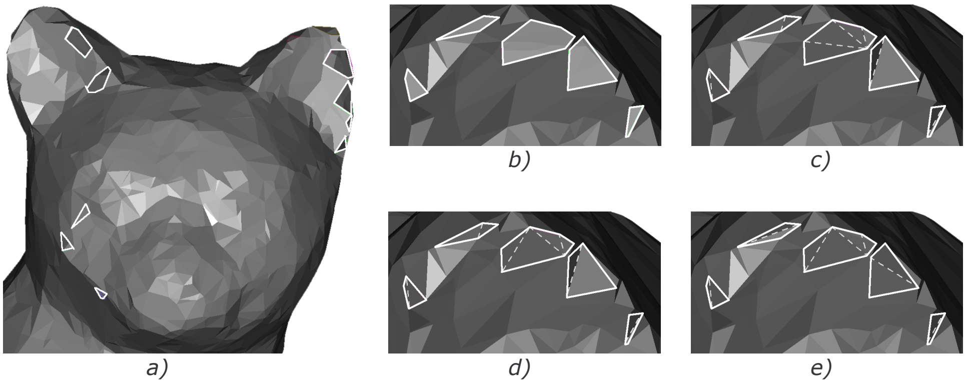


Fig. 9.22: The result of the holes filling, a) the head of the cat with highlighted holes, b) a zoom to one ear with holes, c) the smallest angle filling, d) the smallest edge length filling, e) the smallest neighbours angle.

Very problematic places are the places where one part of the surface is very close to another surface and the sampling process was not completely correct. Fig. 9.23 shows an example with one of these datasets, two parts of surface are in the problematic places connected with "bridges" and the hole cannot be correctly triangulated.

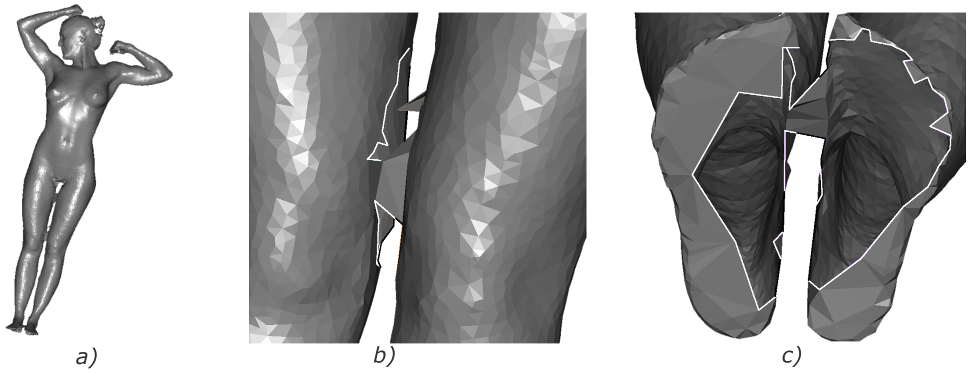


Fig. 9.23: The datafile with problematic connected places, a) the whole surface, b) the legs with big point undersampling, c) the feet with the same problem.

10. Distributed computing

In this chapter we will briefly introduce our experiments and implementation of the distributed version of our approach.

The most powerful way how to reconstruct the large datasets is to exploit all available computers for data storage and computation. The biggest advantage of such approach is that it allows by the distributed computation to process datasets of theoretically unlimited size. For this work we joined with J. Kohout [Kohout05] who has developed the toolkit simulating the shared memory, allowing to use the data read/write operations and also the synchronization between the processes. The toolkit is called "v*irtual shared memory manager* " (*VSM*) and the main feature is that the application does not access the data directly but through the set of functions. This provides a transparent access for the application to all data currently available on each computer of the cluster no matter whether the data is stored locally or not.

When the application begins, it registers in *VSM* the needed data structures, e.g. the points, triangles or tetrahedra, which are stored in local memories of the computers in the cluster. Whenever the application wants to allocate a new data type, it calls an appropriate function of the *VSM*. In the case that the data is not stored locally, the *VSM* finds where the data is stored, it loads the data to the local cache and returns a 32-bit pointer to the locally stored data. The *VSM* internally uses 64-bit pointers which allows to access more memory than available address space on the current 32-bit computers.

Whenever the application wants to modify the data, simultaneous modifications of the data must be avoided. The *VSM* locks the element in such operations for an exclusive access. If the locking is not allowed, e.g., the data is used by another process, the application is suspended until the blocking application finishes the modification. In the case that the waiting would cause a deadlock, the operation fails and the calling application has to return to the stable state and to call the CancelUpdate operation. The *VSM* then returns back all non-confirmed changes and unlocks all data. When the deadlock is detected, the application has to call from time to time the Update operation to confirm the changes and to unlock the data.

We use the *VSM* system for the distributed computation of the Delaunay tetrahedronization now. The points coordinates are the most frequently used data, so they are duplicated on each computer of the cluster in order to reduce the communication. It is not too limiting and it increases speed of the processing. The most memory consuming data structures, such as the tetrahedra mesh and the the triangle mesh, are not duplicated but distributed by the *VSM*.

The work of the distributed computation is as follows. First, each processor loads the input points into the local memory. Next, it determines which points to process by a modified Mueller algorithm [Mueller97] which ensures an almost balanced workload. It is based on a recursive division of the summed-area table constructed from the 3D grid that covers the minmax box of the data and, in every cell, contains the number of points lying in this cell. Then, the processors insert successively their points into the common Delaunay tetrahedronization. When all processors finish the insertion, one of them proceeds with the extraction of triangulation.

This extraction is performed using the approach presented in this thesis. As it is not distributed yet, we are limited now by 2GB of memory. The full distribution of the whole surface reconstruction is not so much complicated to implement but it is still a lot of programming. Fig. 10.1 shows an example of the distributed reconstruction of the dataset which has 1.4M points.

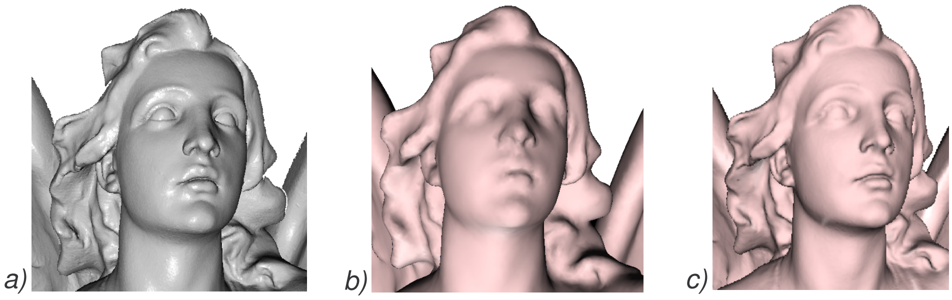


Fig. 10.1: The reconstruction of the dataset with 1.4M points, a) the original viewed by a points based rendering, b) reconstructed by our decimation approach from 200K of points, c) the reconstruction of the whole dataset using the distributed version of Delaunay tetrahedronization.

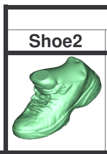
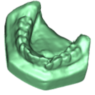
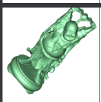
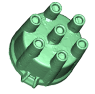
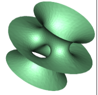
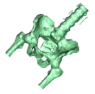
11. Results and comparison

The first part of this chapter shows the time measurements of all steps of the proposed approach for some datasets. Next, this chapter shows the comparison of the reconstruction of problematic datasets using two free academic surface reconstructors and two commercial, too.

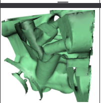
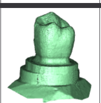
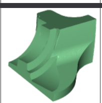
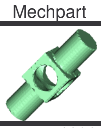
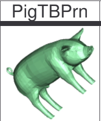
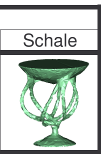
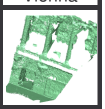
11.1. Measured time

We have tested the time for the reconstruction of all presented steps. We have selected for our tests in this chapter the datasets which were problematic for the reconstruction algorithms from the internet sources at FarField [Farfield_WEB], INRIA [INRIA_WEB], Max Planck Insitute [Max_WEB] and from VrVis Graz. We have also other larger data but not the references to them. The implementation was done in Borland Delphi in Object Pascal under the operating system Windows XP and the tests of the object reconstruction ran on the CPU AMD XP+ 1500 with 1GB of memory.

The results of all algorithms steps (except the preprocessing steps) are presented in Tab. 11.1, where the complete time for reconstruction and the recomputation to the number of points per second are shown. The table is split to several parts. The first part shows the provider and the name of the dataset and its visualization. Next, the measurements are presented for each dataset, the first three rows show the number of points, the number of tetrahedra and the number of surface triangles chosen from the tetrahedra face. Other ten rows bring the measurements of the time for particular steps and the last two rows show the time of the whole reconstruction and the recalculation of this time to the reconstructed points per second. In the separated table, the time of every step in percentage with respect to the whole reconstruction time is presented.

	FarField				Max Planck Institute			
	Shoe2	**Dentalcast**	**Toilet**	**ManqHead**	**BuddhaS**	**DistCap**	**HyperSht**	**BeckWolf**
								
points	78239	38759	22926	10113	32328	12745	6752	2277
tetrahedra	517803	220001	153521	63040	221202	86143	50806	14822
surface triangles	156458	77445	45444	19913	64636	25324	12794	4400
points loading (sec)	0.88	0.41	0.31	0.16	0.57	0.22	0.11	0.04
tetrahedronization (sec)	23.08	12.02	6.43	2.52	9.2	3.51	1.91	0.56
data preprocessing (sec)	1.05	0.48	0.31	0.12	0.45	0.17	0.1	0.03
poles computation (sec)	3.01	1.35	0.91	0.35	1.45	0.51	0.35	0.08
prefiltering (sec)	0.12	0.05	0.03	0.02	0.06	0.02	0.01	0.01
prim. surface extract. (sec)	1.72	0.82	0.5	0.2	0.75	0.28	0.17	0.05
manifold extraction (sec)	0.81	0.45	0.23	0.1	0.33	0.12	0.06	0.02
boundary filtering (sec)	23.89	13.42	7.65	3.61	10.79	3.92	2.25	0.72
postfiltering (sec)	0.79	0.45	0.25	0.11	0.43	0.12	0.08	0.02
holes filling (sec)	0.03	0.02	0.02	0.01	0.02	0.01	0.01	0
Sum (sec)	**55.38**	**29.48**	**16.64**	**7.2**	**24.04**	**8.88**	**5.05**	**1.53**
Points per second	**1412.66**	**1314.84**	**1377.98**	**1403.73**	**1344.89**	**1435.3**	**1338.28**	**1489.89**

points loading	1.59%	1.40%	1.84%	2.23%	2.36%	2.49%	2.23%	2.60%
tetrahedronization	41.67%	40.78%	38.68%	35.04%	38.28%	39.48%	37.79%	36.63%
data preprocessing	1.89%	1.64%	1.87%	1.64%	1.88%	1.90%	2.01%	2.24%
poles computation	5.44%	4.59%	5.44%	4.85%	6.02%	5.74%	6.89%	5.27%
prefiltering	0.21%	0.18%	0.21%	0.23%	0.24%	0.28%	0.21%	0.33%
primary surface extraction	3.11%	2.79%	3.02%	2.83%	3.14%	3.18%	3.35%	3.10%
manifold extraction	1.47%	1.52%	1.38%	1.38%	1.35%	1.33%	1.22%	1.25%
boundary filtering	43.14%	45.52%	45.95%	50.13%	44.87%	44.17%	44.59%	46.84%
postfiltering	1.43%	1.52%	1.52%	1.50%	1.79%	1.37%	1.57%	1.46%
holes filling	0.05%	0.06%	0.10%	0.16%	0.07%	0.07%	0.14%	0.27%

	INRIA							VRVis
	Fish_54	Tomo_54	ToothKreon	Fandisk	Mechpart	PigTBPrn	Schale	Vienna
								
points	54811	47861	36330	6475	4102	3511	2714	100000
tetrahedra	345589	323466	234011	36620	27462	22739	16933	637086
surface triangles	109311	93047	71780	12945	8209	7005	5267	171770
points loading (sec)	0.96	1.35	0.44	0.17	0.12	0.11	0.06	1.65
tetrahedronization (sec)	17.19	15.13	9.96	1.54	1.03	0.85	0.66	28.82
data preprocessing (sec)	0.72	0.74	0.46	0.07	0.06	0.04	0.03	1.39
poles computation (sec)	2.13	2.13	1.54	0.22	0.25	0.14	0.1	4.55
prefiltering (sec)	0.07	0.09	0.08	0.01	0.01	0.01	0	0.22
prim. surface extract (sec)	1.13	1.25	0.79	0.13	0.11	0.07	0.05	2.18
manifold extraction (sec)	0.57	0.49	0.38	0.06	0.05	0.03	0.03	1.03
boundary filtering (sec)	16.2	18.43	13.78	1.98	1.67	1.12	0.91	32.12
postfiltering (sec)	0.61	0.54	0.39	0.07	0.05	0.05	0.03	1.21
holes filling (sec)	0.03	0.13	0.03	0	0	0	0.01	3.89
Sum (sec)	**39.61**	**40.27**	**27.86**	**4.24**	**3.36**	**2.42**	**1.88**	**77.07**
Points per second	**1383.91**	**1188.61**	**1304.23**	**1525.65**	**1222.32**	**1452.69**	**1440.55**	**1297.49**

points loading	2.42%	3.34%	1.59%	4.01%	3.63%	4.65%	2.98%	2.14%
tetrahedronization	43.41%	37.59%	35.75%	36.37%	30.68%	34.98%	35.15%	37.40%
data preprocessing	1.82%	1.83%	1.66%	1.67%	1.88%	1.77%	1.69%	1.81%
poles computation	5.38%	5.29%	5.53%	5.15%	7.33%	5.59%	5.45%	5.90%
prefiltering	0.18%	0.22%	0.29%	0.19%	0.22%	0.21%	0.26%	0.29%
primary surface extraction	2.86%	3.10%	2.83%	2.95%	3.38%	2.98%	2.82%	2.83%
manifold extraction	1.43%	1.21%	1.36%	1.39%	1.48%	1.37%	1.36%	1.34%
boundary filtering	40.90%	45.76%	49.48%	46.62%	49.73%	46.25%	48.20%	41.68%
postfiltering	1.55%	1.34%	1.39%	1.54%	1.59%	2.02%	1.58%	1.57%
holes filling	0.06%	0.33%	0.11%	0.10%	0.07%	0.19%	0.53%	5.05%

Tab. 11.1: The time measurements of all algorithms steps for some datasets (numbers in the second and fourth table show the recomputation to percentage of time occupied by particular steps).

From the table it is clearly understandable that the speed of computation per one point is almost constant for all datasets and the time increases nearly linearly with the number of points. The same conclusion flows from the graphs in Fig. 11.1 where dependency of the whole reconstruction time on the number of points and the recalculation to the reconstructed points per second are shown. The results in the presented graph and table confirm our presumption that all the steps of algorithm have the expected algorithm complexity $O(cN)$ where the constant c depends on the step of the algorithm, but it has a low value.

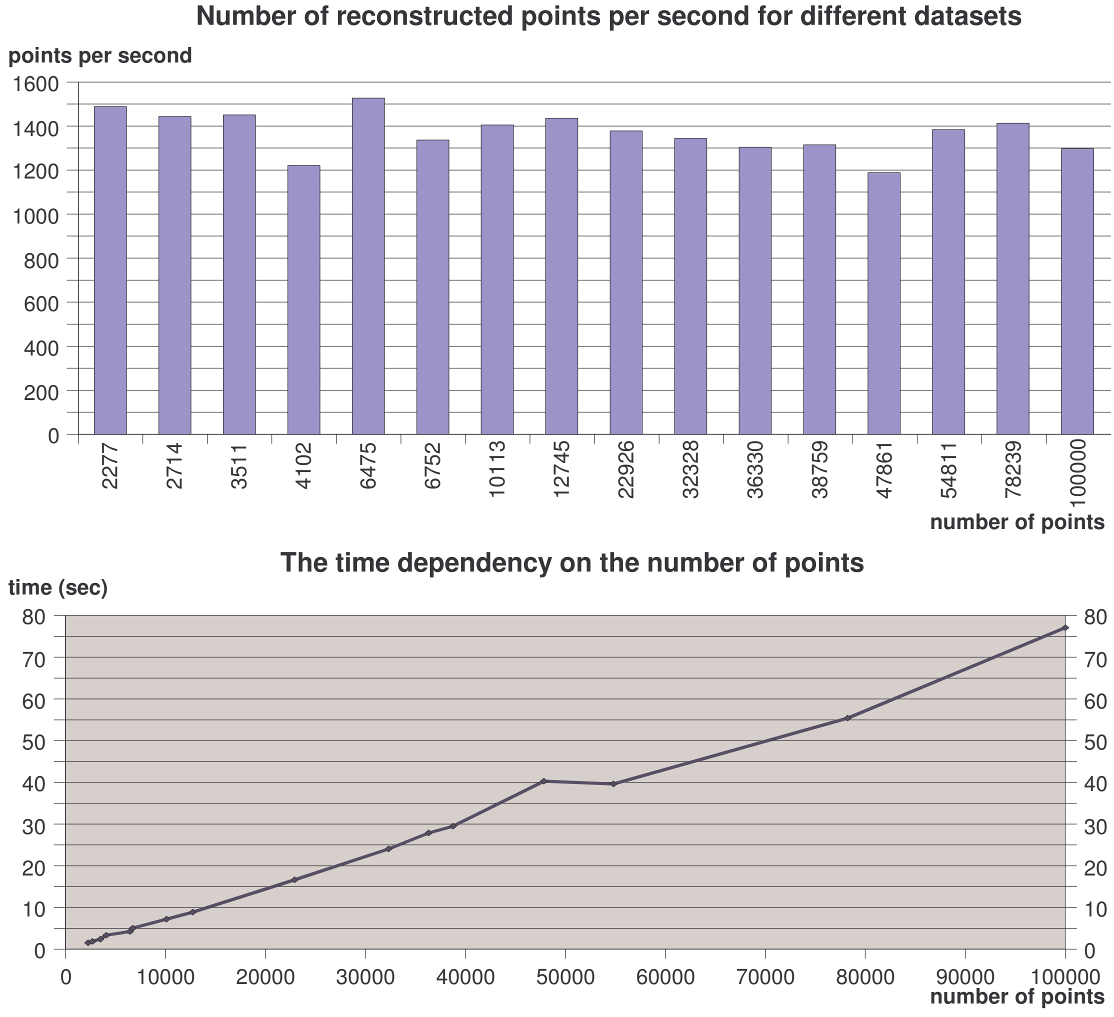


Fig. 11.1: The graphs showing the speed of reconstruction.

The most time consuming part of the algorithm is the computation of Delaunay tetrahedronization and the boundary filtering step. The boundary filtering step is not necessary if we do not expect the datasets with boundaries. It is slow because we have to find for all points their coincident edges from the triangulation and also all the coincident edges of the neighbours. These edges are then sorted and the median is chosen to select the referential edge. Here is still a lot of possible improvements, we sort the edges by a quicksort algorithm, whose expected complexity is $O(N\ logN)$, and then we take the middle edge but the median can be found in $O(N)$. Another possible acceleration is when we put the coincident edges to the array for sorting. To avoid duplicities of edges we have to look to the already inserted edges which is now done by linear search. So the best way for speed improvement is to keep the array sorted from the beginning because the duplicities can be found then very quickly and the median, too.

11.2. Reconstructors comparison

We have tried the free web surface reconstructor which was developed in the French research institute INRIA [INRIA_WEB]. The surface is grown from a seed facet by adding triangles from Delaunay triangulation one by one. The most plausible triangles are added in the first place, in a way that prevents the appearance of topological singularities. To handle objects with boundaries the user has to define one parameter. The reconstructor presented at the web pages is well working but we did not get an executable version. Thus we could not measure the time for the reconstruction and to test the parameter settings because at the web pages the user can only set whether the dataset is with boundary or not.

The second tested surface reconstructor is the COCONE which was presented in section 4.5. We have the program from the web pages of one of its author [Dey_WEB]. It is the version which can handle objects with boundaries automatically by exploring the shapes of Voronoi cells. It is very close to the CRUST algorithm, thus the comparison with our approach gives us the best information about our improvements. For the reconstruction of noisy datasets we used the newest version specially designed for noisy data reconstruction called robustCOCONE (presented in section 7.2). It produces the triangle mesh interpolating the input points, the same holds for the INRIA reconstructor. Both reconstructors have problem with the orientation of triangles because the produced meshes have not a consistent orientation. The meshes from the COCONE approach have another property, it looks that the points in the output mesh position do not correspond to the original points position, the whole input set is probably somehow scaled and rotated.

The first tested commercial program is called PointCloud and was developed by Floating point solution [Floating_WEB], its price is 250$. We have not found any information about the approach used for the reconstruction but from the result we can derive that it uses a voxel grid and some voxel algorithm, such as Marching Cubes, for extracting the surface. The surface has very low quality and artificial artefacts appear in the triangle mesh after the reconstruction very often. We did not find any parameter how to improve the reconstruction, the user may only select whether the mesh (probably height field) or surface is processed. We use the demo version which disallows to save the reconstructed triangle mesh.

The second commercial program called SilverLining developed by FarField Technology uses the same restriction in demo version – it is not possible to save the output mesh and, moreover, the internal renderer of the mesh writes the text “demo” over the rendered reconstructed object. This program costs 4500$ for a permanent license and it uses the radial based functions for the reconstruction [Carr01]. Therefore it is very robust to errors in sampling although very slow compared with the other tested approaches. Its advantage is that the user can specify the thresholds of some additional filtering rules, such as outliers removal, large triangles deletion, decimation or additional smoothing, although these filters are not automatic. The output surface is the approximation which contains different number of points than the input and any sharp and small features of the surface are not preserved, the surface is very smooth.

We have also tested the superCRUST algorithm developed by Nina Amenta but due to the algorithm properties described below we did not present its results here. It is strictly designed for watertight surface reconstructions and these reconstructions were pretty good (with small mistakes, such as some parts of surface incorrectly connected), but the objects with boundaries were reconstructed completely incorrectly. The next problem is that it does not produce the triangles but general polygonal faces and the last problem is the amount of output data because it adds extra points to the triangulation. For example, the datasets with 10K points had after the reconstruction about 90K of points and 175K triangles. We obtained some other reconstruction programs (from authors websites), the original CRUST and the Hoppe's approach, unfortunately, we got them as a source code in C and we were not able to compile them because some special libraries were missing.

We will not show the results of the reconstruction of well sampled datasets because almost all algorithms have no problems with such data. Thus we have chosen the datasets with incorrectly sampled places, boundaries, very closed different parts of surface, undersampling and noise. Some datasets used for comparison are not presented in Tab. 11.1 because in the table the measurements only of few our datasets are shown.

The worse reconstruction was always achieved by the PointCloud software, therefore, the details are not presented in the figures, just one reconstruction is shown for comparision.

We cannot correctly measure the time of all reconstructors because we do not have the executable of INRIA reconstructor (it is provided as the WEB service), the PointCloud has not any internal timer and the timer in SilverLining was not precise. It means that sometimes the reconstruction lasted several minutes but the program reported time in seconds. But the SilverLining seems to be almost every time the slowest due to *RBF* computation. The PointCloud reconstructor is sometimes faster, sometimes slower than our approach. The approaches based on the Delaunay tetrahedronization, which is the most time consuming operation, have approximately similar speed. Tab. 11.2 shows the measured reconstruction time for the compared datasets (except INRIA surface reconstructor).

	SilverLining	PointCloud	COCONE	Our approach
Woman	101.0	19.0	47.2	18.0
PigTBPrinceton	5.0	1.5	5.0	2.4
Toilet	~80min	14.5	38.8	16.6
Fish_54	1041.0	58.0	121.6	39.6
Schale	4.0	1.5	2.0	1.8
Cat_noise	14.0	5.0	17.2	12.0

Tab. 11.2: The comparison of the reconstruction time (sec) by different approaches.

The dataset “woman” which is presented in Fig. 11.2 has four problematic places due to local undersampling. The first is on the top of the head as the arm is very close and these two parts of the object may be connected together by mistake. The same problem occurs in other places, e.g., the legs are connected together by triangles and the feet, too. The ears are reconstructed badly because there were not enough points for a correct output. Visually, our approach followed by INRIA seems to be the best as there is not so many connections between different parts of surface. The COCONE had more problems, the SilverLining software reconstructed correctly the head with the lost of small details. The legs were connected, too, and the feet were merged together.

The next tested problematic dataset is the set “PigTBPrinceton” (Fig. 11.3). It contains parts with very small details, mainly in the area of hoofs, tail and head. The COCONE reconstructs the surface with many holes in these places. The INRIA reconstructor is more successful and except the hoofs, where some holes and unwanted connection appear, it reconstructs the surface correctly as well as our approach. The SilverLining looses again all small details, it disconnects the tail of the body and merges the back hoofs.

Next, we have tried to reconstruct the dataset “Toilet” (Fig. 11.4) which has two parts of the surface in the main part very close and sharp edges in the bottom. These places bring big problems mainly for the COCONE which is not able to reconstruct these parts correctly and large holes with unconnected parts occur. INRIA reconstructor was again more successful but in few cases it connects the surface parts together and holes appear in the bottom. Our approach reconstructs the body of the toilet quite well only with small holes in the bottom on the sharp edges. Surprisingly, the SilverLining have the biggest problems. Its reconstruction was terrible - the body of the toilet contains a big hole, on the bottom some artefacts appear and the top is unconnected.

The dataset "Fish_54" (Fig. 11.5) is very problematic for the reconstruction as it contains very narrow parts, mainly the top fin. All approaches have problems with the reconstruction of this part, especially the COCONE which connects the top fin with the back fin and produced more holes then other approaches. The INRIA reconstructor was more successful, its result was similar to our approach although we produces less holes. The SilverLining reconstructed only one part of the fin and did not take into the account the points on the other side. This software has problems on other parts of surface, too, on the bottom some artificial artefacts appear, front down fins are unconnected with the body and the mouth is closed with some triangles. The mouth is incorrectly reconstructed by the COCONE, many holes appear there, the INRIA reconstructor is better but holes are there, too. The eyes are reconstructed as elliptical objects separated from the fish body but they lie very close to the body. Therefore the COCONE and INRIA reconstructor connect with some triangles the eyes with the body, our approach reconstructed these places correctly.

Next dataset "Schale" (Fig. 11.6) contains very low number of points so it is very undersampled. The top and bottom part are reconstructed with small mistakes (missing triangles) by our approach and INRIA reconstructor and both approaches correctly filled the top part. The COCONE has more problems, it leaves the top empty without triangles although there are some points. The most problematic parts is the middle of the cup, the COCONE reconstructed only some small parts, the reconstruction with the INRIA is better but some places are wrong, only our approach reconstruct this part correctly (only with a few small holes). The reconstruction with SilverLining was completely bad, almost no part was reconstructed correctly.

The reconstruction of noisy datasets and some results were presented in section 7.2, we show in Fig. 11.7 another example of the reconstruction of one noisy dataset "Cat_noise" for comparison. As the INRIA surface reconstructor is not designed for such dataset reconstructions, its results are unusable. The same holds for the reconstruction by the COCONE reconstructor, therefore, we used the robustCOCONE version suitable for the noisy dataset reconstruction which results are quite good with small errors in triangle mesh. The ears contain holes and the bottom of the dataset is incorrectly connected with another part of the surface. The reconstruction by our approach with points denoising (the normal denoising, k is set to 20 and one iteration without points translation) is nearly correct with a few holes and the boundary triangles in the bottom are correctly removed. Surprisingly, the PointCloud software reconstructed almost correctly this dataset, the only problem is again in the bottom where more triangles, which do not correspond with the points, appear. The SilverLining had no problems with the reconstruction, either, although we had to change the parameters for the reconstruction because the reconstruction with the automatic setting was too coarse.

To conclude the comparison of reconstructors, we have to make a short recapitulation. The worst reconstructor with almost unusable results is the commercial PointCloud software. The surface does not correspond to the input points and it fails in more complicated parts. The SilverLining software is the slowest reconstructor due to the *RBF* computation and isosurface extraction but it produces nice results in the case of smooth objects. The size of the output mesh depends on the parameters set by the user, we used the default setting. The sharp edges are oversmoothed and many small details are lost, the worse effect is that some parts of surface are merged or disconnected. To avoid this effect, the user can specify a more precise computation but then the speed of computation is very low.

The COCONE reconstructor has some problems in the parts which do not fulfil the *LFS* sampling criterion (it is similar to the CRUST), this is the reason of all the problematic parts presented in the comparison figures. We did not have the version for watertight reconstruction, this version could be more successful in these places. The INRIA surface reconstructor is a very good program with well reconstructed surfaces and its results are similar to our approach. Although in the presented test we usually got better results with our approach then with the compared programs, we do not want to say that our approach is better because we processed the tests on the datasets we used for our implementation debugging. If we processed the data from the authors of the INRIA reconstructor, the results would probably be better for the INRIA reconstructor.

In Chapter 3 we said that the approaches, which approximate the points, are not very sensitive to noise. This fact was confirmed by our observation and all tested programs reconstructed these datasets well (for the PointCloud software the “Cat_noise” dataset was the only one correctly reconstructed dataset). But the interpolation approaches specially designed for the noisy datasets reconstruction are respectable competitors to the approximation approaches.

The results of the reconstruction of large datasets were shown in Chapter 7.3. Generally, the approaches based on Delaunay tetrahedronization are limited with the amount of data which can be processed by the tetrahedronization as it is the most memory consuming step. Our implementation is now able to process 500K points on the system with 2 GB of memory (with the use of *DAG*), the authors of INRIA reconstructor have tried to reconstruct 500K points.

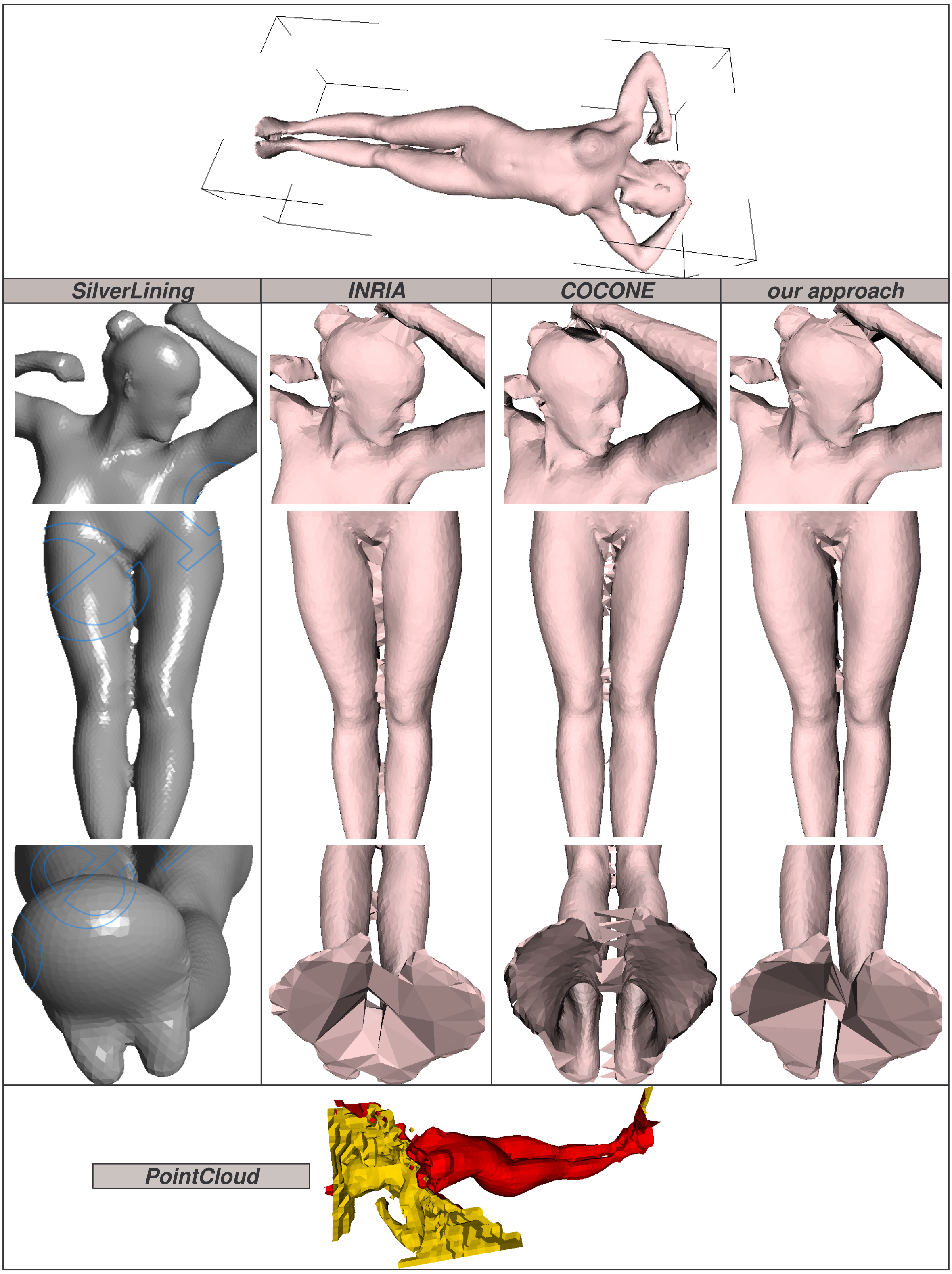


Fig. 11.2: The comparison of the reconstructions of the "Woman" dataset by different approaches.

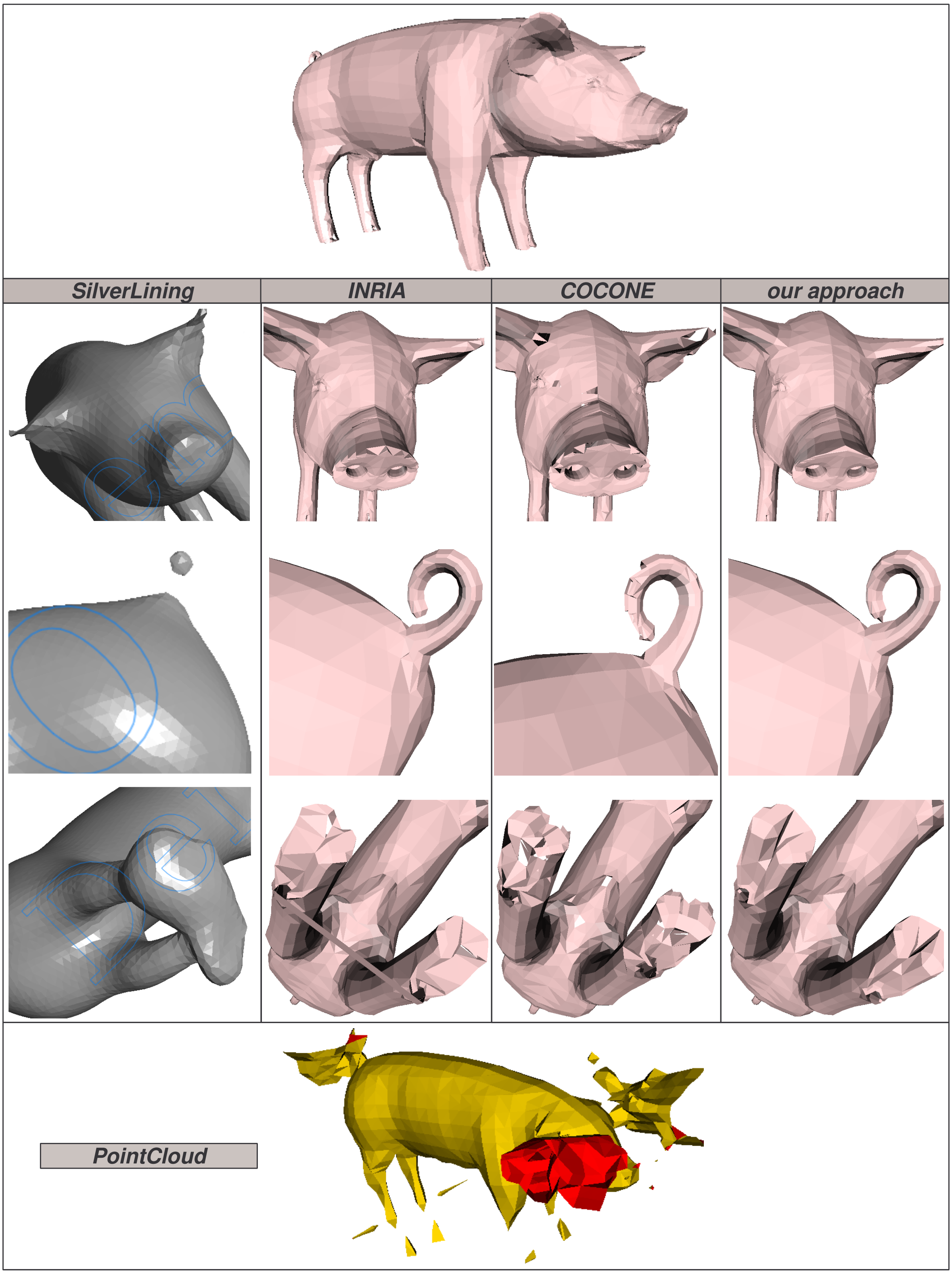


Fig. 11.3: The comparison of the reconstruction of the "PigTBPrinceton"dataset by different approaches.

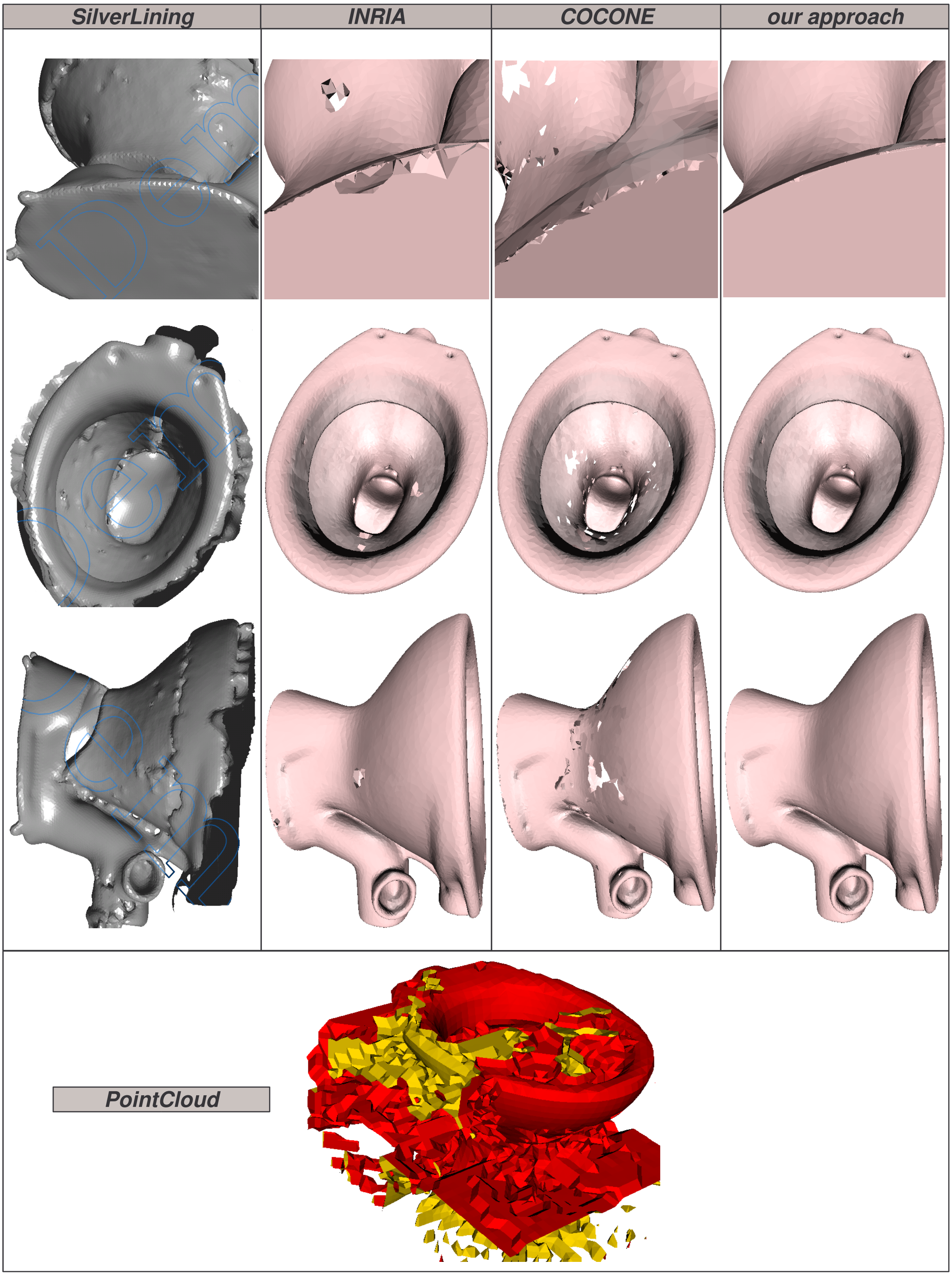


Fig. 11.4: The comparison of the reconstruction of the “Toilet”dataset by different approaches.

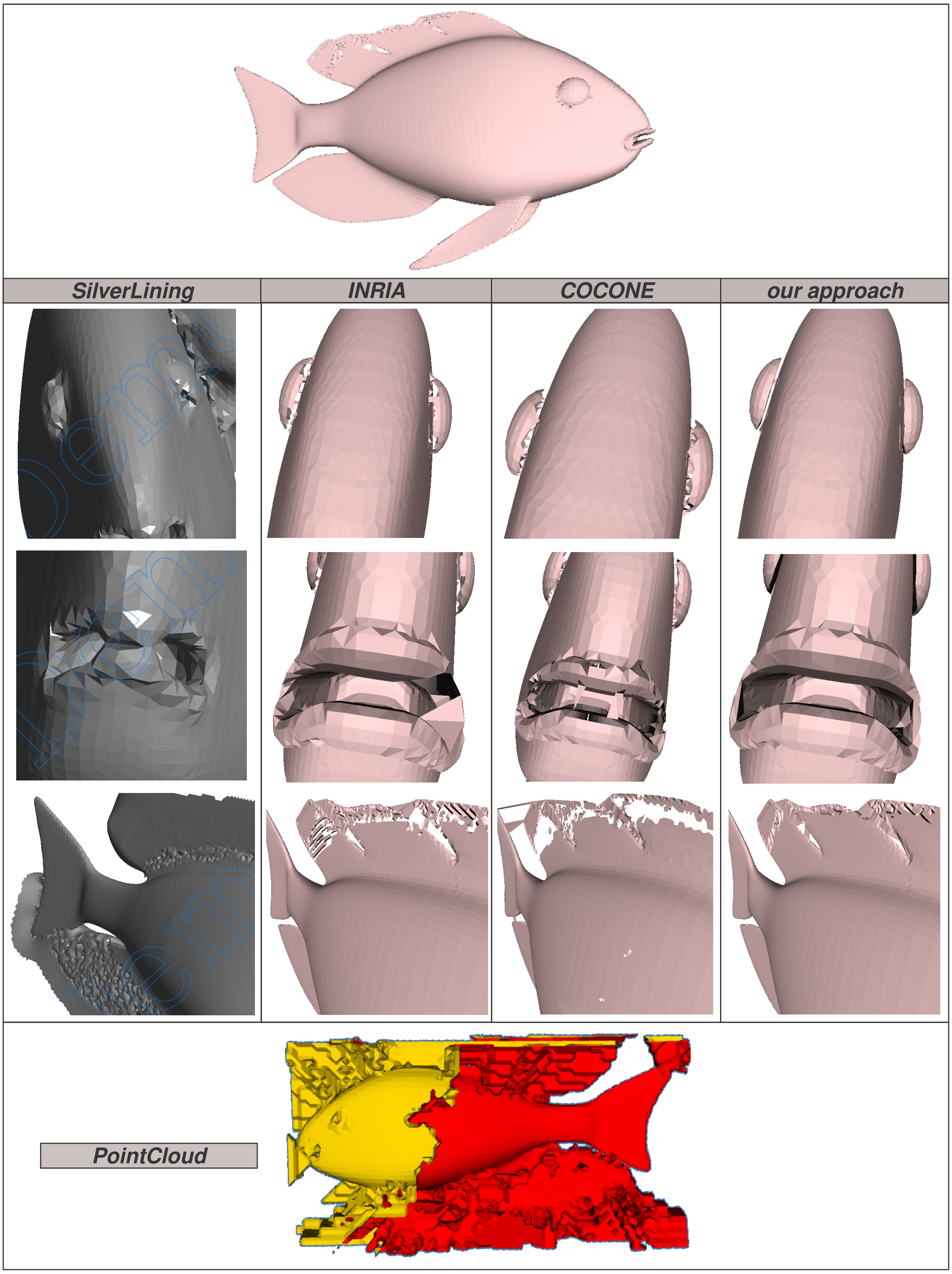


Fig. 11.5: The comparison of the reconstruction of the "Fish_54"dataset by different approaches.

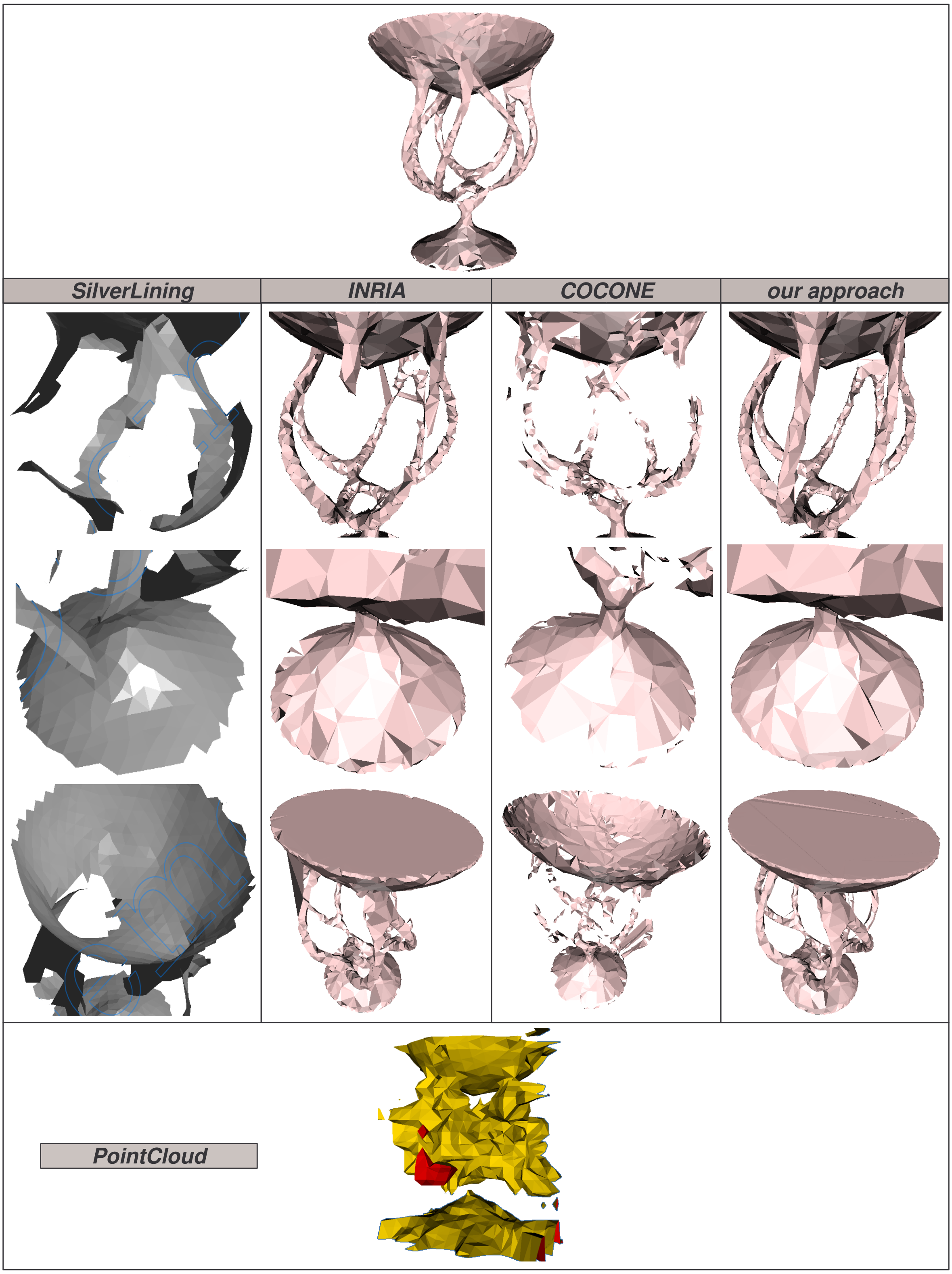


Fig. 11.6: The comparison of the reconstruction of the “Schale”dataset by different approaches.

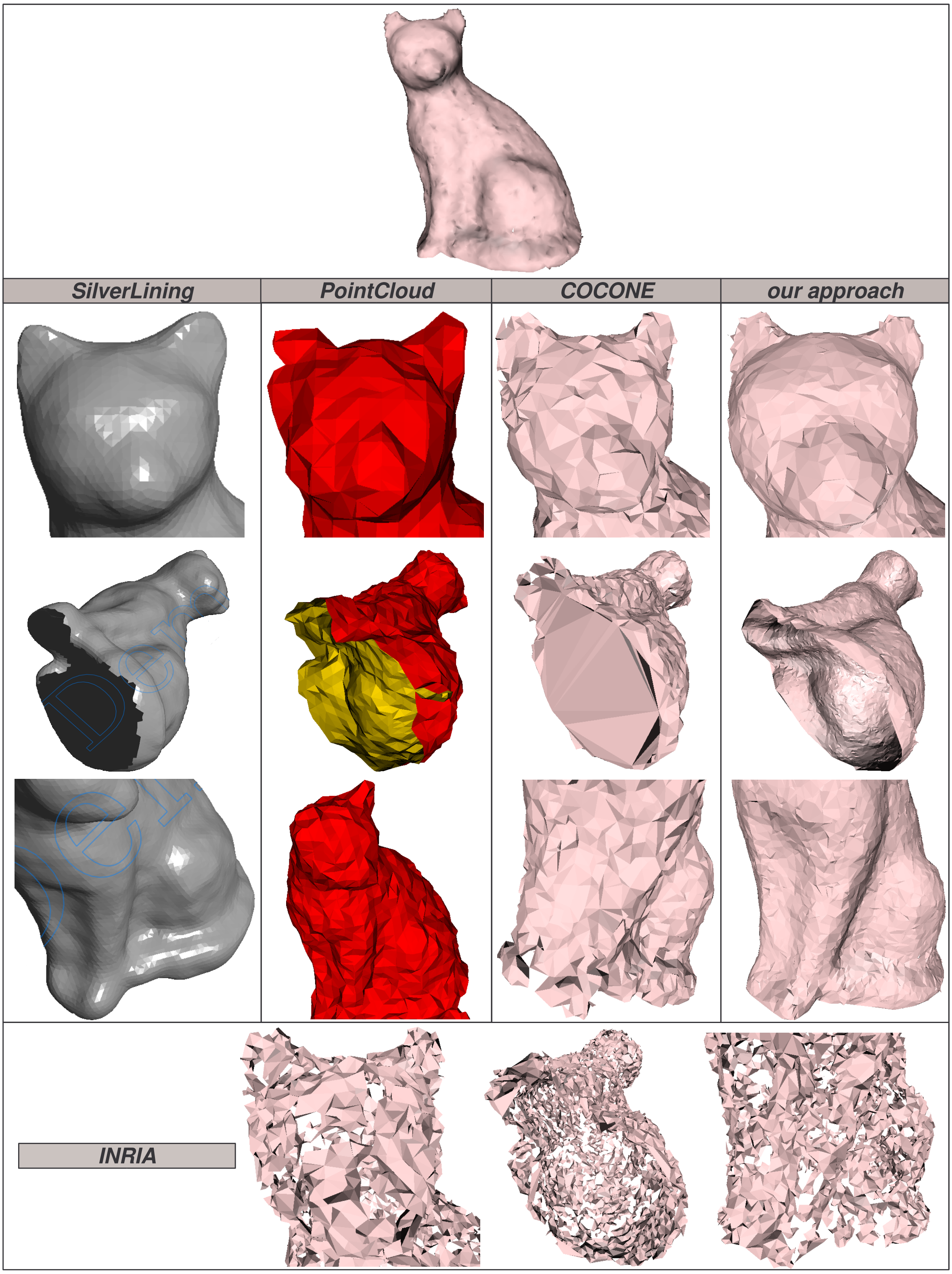


Fig. 11.7: The comparison of the reconstruction of the “Cat_noisy”dataset by different approaches.

12. Conclusions and future work

Although we have implemented several improvements and the reconstruction of all datasets is better than before, there are still many improvements which may be done. This chapter shows some ways where the work could continue and then we conclude the thesis.

12.1. Points decimation with backward transformation

One of other possibilities how to reconstruct large data is to extend our points decimation procedure. In each iteration of the decimation process we remember which input points formed the output points and we create the graph of such process, see an example in Fig. 12.1.

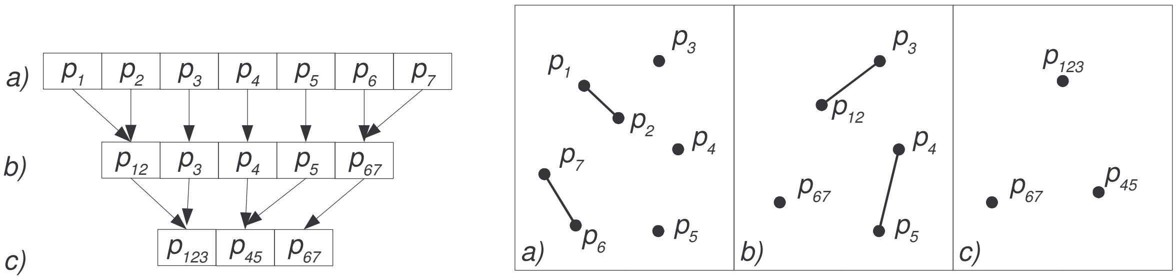

a)
p_1 p_2 p_3 p_4 p_5 p_6 p_7
b)
p_{12} p_3 p_4 p_5 p_{67}
c)
p_{123} p_{45} p_{67}
p_3 p_1 p_2 p_4 p_7 p_6 p_5
a)
p_3 p_{12} p_4 p_{67} p_5
b)
p_{123} p_{67} p_{45}
c)

Fig. 12.1:The points decimation procedure with three iterations and the graph related to each iteration.

After we finish the decimation, we reconstruct the surface using our approach. As we know the original points and the way how we obtained the decimated points, we can now split the decimated points back to their parents by an inverse transformation. By projecting the merged points to the created triangle mesh, we can obtain the triangle mesh with original points after several iterations.

12.2. Fully distributed version

The distributed version is now restricted only to the most time and memory consuming part, to the Delaunay tetrahedronization. But the utilization of the *VSM* functions allows us to implement all the presented steps without extensive knowledge of distributed programming. Full implementation promises the reconstruction of very large datasets and it is supposed to be done as a successive student project.

12.3. Conclusion

We have presented a complete approach to the task of surface reconstruction. It consists of several steps which improve the process of the reconstruction in comparison with the original CRUST algorithm. The presented steps can by divided into three groups. The first group of improvements deals with the preprocessing phase of sampled points and helps mainly in the case of noisy data. It prepares the points to be better processed by the reconstruction algorithm. In the case of large data we have presented one simple approach based on points decimation and one approach based on distributed computation. The second group of improvements deals with the reconstruction itself. We have developed better normal vectors estimation and the step called prefiltering which removes overlapping tetrahedra faces. The third group of improvements deals with postprocessing steps of the reconstructed mesh and it tries to remove overlapping triangles and large boundary triangles from the mesh and to retriangulate the holes.

All the presented steps improve the overall process of reconstruction. We have tested it by the reconstruction of many datasets and by comparing the reconstruction with other reconstruction approaches. The comparison gave us positive results, all data we have were better reconstructed than without these steps and the reconstruction was of the same quality or even better than the reconstruction of the compared methods. There is another advantage of our approach, all the steps which do not deal with the reconstruction itself can be used with any other reconstruction algorithm. Also the speed of all steps is good.

References

[Algorri96] M. E. Algorri, F. Schmitt. *Surface reconstruction from unstructured 3D data.* Computer Graphic Forum, 1996, pp. 47 - 60

[Amenta98a] N. Amenta, M. Bern, D. Eppstein. *The CRUST and β-skeleton: combinatorical surface reconstruction.* Graph. Models and Image Processing, 1998, pp. 125 - 135

[Amenta98b] N. Amenta, M. Bern, M. Kamvysselis. *A new Voronoi-based surface reconstruction algorithm*. SIGGRAPH, 1998, pp. 415 - 421

[Amenta99] N. Amenta, M. Bern. *Surface reconstruction by Voronoi filtering*. Discrete and Computational Geometry, 22 (4), 1999, pp. 481 - 504

[Amenta00] N. Amenta, S. Choi, T. K. Dey, N. Leekha. *A simple algorithm for homeomorphic surface reconstruction.* 16th. Sympos. Computational Geometry, 2000, pp. 125 - 141

[Amenta01] N. Amenta, S. Choi, R. Kolluri. *The Power Crust.* Proc. of 6th ACM Sympos. on Solid Modeling, 2001, pp. 127 - 153

[Attali97] D. Attali. *R-regular shape reconstruction from unorganized points.* Symposium on Computational Geometry, 1997, pp. 248 - 253

[Baader93] A. Baader, G. Hirzinger. *Three dimensional surface reconstruction based on a self-organizing Kohonen map.* Proc. 6th Int. Conf. Advan. Robotics, 1993, pp. 273 - 278

[Baader94] A. Baader, G. Hirzinger. *A self organizing algorithm for multisensory surface reconstruction.* Intern. Conf. on Robot. and Intellig. systems IROS, 1994, pp. 81 - 88

[Bernardini97] F. Bernardini, C. Bajaj. S*ampling and reconstruction manifolds using* α*-shapes.* Proc. 9th Canad. Conf. on Comput. Geometry, 1997, pp. 193 - 198

[Bernardini99] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, G. Taubin. *The ball-pivoting algorithm for surface reconstruction*. IEEE Trans. on Visual. and Computer Graphics, 1999, pp. 349 - 359

[Bernardini00] F. Bernardini, H. Rushmeier. *The 3d model acquisition pipeline.* State of the art report, EUROGRAPHICS 2000, pp. 41 - 62

[Bittar95] E. Bittar, N. Tsingos, M. P. Gascuel. *Automatic reconstruction of unstructured data : Combining medial axis and implicit surfaces.* EUROGRAPHICS 1995, pp. 457 - 468

[Boissonat84] J. D. Boissonat. *Geometric structures for three-dimensional shape representation.* ACM Trans. Graphics 3, 1984, pp. 266 - 286

[Boissonat00] J-D. Boissonnat, F. Cazals. *Smooth surface reconstruction via natural neighbour interpolation of distance functions*. Symp. on Computational Geometry, 2000, pp. 223 - 232

[Boyer00] E. Boyer, S. Petitjean. *Curve and surface reconstruction from regular and non-regular point sets*. Proc. of the Conf. on Computer Vision and Pattern Recognition, 2000, pp. 659 - 665

[Carr01] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, T. R. Evans. *Reconstruction and representation of 3D objects with radial basis functions.* ACM SIGGRAPH, 2001, pp. 67 - 76

[Cohen03] D. Cohen-Steiner, F. Da. *A greedy Delaunay-based surface reconstruction algorithm*. The Visual Computer 20(1), 2004, pp. 4 - 16

[Crossno99] P. Crossno, E. Angel. *Spiraling edge: Fast surface reconstruction from partially organized sample points*. Proc. of the Conf. on Visualization, 1999, pp. 317 - 324

[Curless96] B. Curless, M. Levoy. *A volumetric method for building complex models from range images*. SIGGRAPH, 1996, pp. 303 - 312

[Davis02] J. Davis, S. Marschner, M. Garr, M. Levoy. *Filling holes in complex surfaces using volumetric diffusion.* First Int. Symp. on 3D Data Processing, Visual. and Transmission 2002, pp. 428 - 438

[Delaunay34] B. Delaunay. *Sur la sphére vide*. Izvestia, Akademii Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk 7, 1934, pp. 793 - 800

[Devillers01] O. Devillers, S. Pion, M. Teillaud. *Walking in a triangulation*. Proc. of the Seventeenth Annual Symp. on Comput. Geometry, 2001, pp. 106 - 114

[Dey99] T. K. Dey, P. Kumar. *A simple provable algorithm for curve reconstruction.* Proc. ACM-SIAM Sympos. Discr. Algorithms, 1999, pp. 893 - 894

[Dey00] T. K. Dey, K. Mehlhorn, E. A. Ramos. *Curve reconstruction: connecting dots with good reason.* Comput. Geom. Theory Appliacation, 2000, pp. 222 - 244

[Dey01a] T. K. Dey, J. Giesen. *Detecting undersampling in surface reconstruction.* Proc. of 17th ACM Sympos. Comput. Geometry, 2001, pp. 257 - 263

[Dey01b] T. K. Dey, J. Giesen, J. Hudson. *Delaunay based shape reconstruction from large data*. Proc. IEEE Sympos. in Parallel and Large Data Visualization and Graphics, 2001, pp. 19 - 27

[Dey01c] T. K. Dey, J. Giesen, N. Leekha, R. Wenger. *Detecting boundaries for surface reconstruction using co-cones.* Intl. J. Computer Graphics & CAD/CAM, vol. 16, pp. 141 - 159

[Dey01d] T. K. Dey, J. Giesen, W. Zhao. *Robustness issues in surface reconstruction.* Proc. Intl. Conf. Comput. Science, San Francisco, California, 2001, pp. 658 - 652

[Dey03] T. K. Dey, S. Goswami. *Tight Cocone: A water-tight surface reconstructor.* Proc. 8th ACM Sympos. Solid Modeling application (2003), pp. 127 - 134

[Dey04] T. K. Dey, S. Goswami. *Provable surface reconstruction from noisy samples*. Proc. of 20 annual symposium on Computational Geometry, SCG, 2004, pp. 330 - 339

[Dey_WEB] http:\\www.cse.ohio-state.edu\~tamaldey\cocone

[Edelsbrunner92] H. Edelsbrunner. *Weighted alpha shapes.* Technical report UIUCDCS-R92-1760 DCS University of Illinois at Urbana-Champaign, Urbana, Illinois, 1992

[Edelsbrunner94] H. Edelsbrunner, E. P. Mücke. *Three-dimensional alpha shapes.* ACM Trans. Graphics 13, 1994, pp. 43 - 72

[Emelyanov04] Emelyanov A. I. *Surface reconstruction from clouds of points.* PhD thesis, University of West Bohemia, Pilsen, Czech Republic 2004

[Fabio03] R. Fabio. *From point cloud to surface: the modeling and visualization problem*. Int. Workshop on Visualization and Animation of Reality-based 3D Models, 2003

[Farfield_WEB] http://www.farfieldtechnology.com/products/silverlining/

[Freitag97] L. Freitag. *On combining laplacian and optimization-based mesh smoothing techniques*. Proc. Symp. Trends in Unstructured Mesh Generation,1997, pp. 37 - 43

[Floating_WEB] http://www.fpsols.com/point_cloud.html

[Giesen02] J. Giesen, M. John. *Surface reconstruction based on a dynamical system*. Proc. of the 23rd Annual Conf. of the European Association for Computer Graphics (Eurographics), Computer Graphics Forum 21, 2002, pp. 363 - 371

[Gopi02] M. Gopi, S. Krishnan. *A fast and efficient projection-based approach for surface reconstruction*. 15th Brazilian Symposium on Computer Graphics and Image Processing, 2002, pp. 179 - 186

[Gross96] W. I. Gross, O. G. Staadt, R. Gatti. *Efficient triangular surface approximations using wavelets and quadtree data structures*. Visualization and Computer Graphics, 1996, pp. 130 - 143

[Hoppe92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle. s*urface reconstruction from unorganized points.* Computer Graphics 26 (2), 1992, pp. 71 - 78

[Hoppe94] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, W. Stuetzle. *Piecewise smooth surface reconstruction.* SIGGRAPH, 1994, pp. 295 - 302

[Isselhard97] F. Isselhard, G. Brunett, T. Schreiber. *Polyhedral reconstruction of 3D objects by tetrahedra removal*. Technical report, Fachbereich Informatik, University of Kaiserslautern, Germany, 1997, Internal report No. 288/97.

[INRIA_WEB] http:\\cgal.inria.fr\Reconstruction\

[Kobbelt00] L. P. Kobbelt, S. B. M. Botsch, K. Kähler, Ch. Rössl, R. Schneider, J. Vorsatz. *Geometric modeling based on polygonal meshes*. EUROGRAPHICS 2000, pp. 1 - 47

[Kolingerova02] I. Kolingerová. *Modified DAG location for delaunay triangulation*, Computational Science - ICCS 2002, Part III, Amsterdam - The Netherlands, pp.125-134, LNCS 2331, Springer-Verlag, 2002

[Kohout03] J. Kohout, I. Kolingerová. *Parallel Delaunay triangulation in E^3: make it simple*. The Visual Computer 2003, 19 (7&8), pp. 532 – 548

[Kohout05] J. Kohout. *Delaunay triangulation in parallel and distributed environment*. PhD thesis, University of West Bohemia, Pilsen, Czech Repubic, 2005

[Kolluri04] R. Kolluri, J. R. Shewchuk, J. F. O'Brien. *Spectral surface reconstruction from noisy point clouds.* Symposium on Geometry Processing, Nice, France, 2004, pp. 11 - 21

[Lorensen87] W. E. Lorensen, H. E. Cline. *Marching Cubes: A high resolution 3d surface reconstruction algorithm.* Comp. Graphics 21 (4), 1987, pp. 163 - 169

[Lukasova85] A. Lukasová, J. Šarmanová. *Metody shlukové analýzy.* SNTL - Nakladatelství technické literatury, 1985, 04-014-85, typové číslo L11-E1-IV-41f/11865

[Maur03] P. Maur. *Delaunay triangulation in 3D.* Technical report DCSE/TR-2002-02, University of West Bohemia, Czech Republic, 2002

[Mencl95] R. Mencl. *A graph based approach to surface reconstruction.* Comp. Graph. forum 14 (3), EUROGRAPHICS 1995, pp. 445 - 456

[Mencl98a] R. Mencl, H. Müller. *Interpolation and approximation of surfaces from three-dimensional scattered data points.* EUROGRAPHICS 1998, pp. 223 - 233

[Mencl98b] R. Mencl, H. Müller. *Graph based surface reconstruction using structures in scattered point sets*. Proc. CGI, 1998, pp. 298 - 312

[Michelangelo00] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, D. Fulk. *The digital Michelangelo project: 3D scanning of large statues.* SIGGRAPH 2000, pp. 131 - 144

[Morse01] B. S. Morse, T. S. Yoo, D. T. Chen, P. Rheingans, K. R. Subramanian. *Interpolating implicit surfaces from scattered surface data using compactly supported radial basis Functions*. Proc. of the Int. Conf. on Shape Modeling & Applications, 2001

[Miller91] J. V. Miller, D. E. Breen, W. E. Lorenzem, R. M. O'Bara, M. J. Wozny. *Geometrically deformed models: A Method for extracting closed geometric models from volume data.* Proc. SIGGRAPH, 1991, pp. 217 - 226

[Mueller97] C. Mueller. *Hierarchical graphics databases in sort-first*. Proc. of IEEE Symp. on Parallel Rendering, 1997, pp. 49 - 57

[Muraki91] S. Muraki. *Volumetric shape description of range data using "Blobby model".* Comp. Graphics, 1991, pp. 217 - 226

[Mücke93] E. P. Mücke. *Shapes and implementations in three-dimensional geometry.* PhD. thesis, DCS University of Illinois at Urbana-Champaign, Urbana, Illinois, 1993

[Pauly01] M. Pauly, M. Gross. *Spectral processing of point sampled geometry.* SIGGRAPH, 2001, pp. 379 - 386

[Roth97] G. Roth, E. Wibowoo. *An efficient volumetric metod for building closed triangular meshes from 3d images and point data*. In Graphics Interface, 1997, pp. 173 - 180

[Schewchuck96] J. R. Schewchuck. *Robust adaptive floating-point geometric predicates.* Proc. of 12th ACM, 1996, pp. 141 - 150

[Szeliski92] R. Szeliski, D. Tonnesen. *Surface modeling with oriented particle systems.*Comp. Graphics 26, 1992, pp. 185 - 194

[Terzopoulos88] D. Terzopoulos, A. Witkin, M. Kass. *Constrains on deformable models, Recovering 3d shape and nongrid motion*. Art. Intelligence, 1988, pp. 91 - 123

[Terzopoulos91a] D. Terzopoulos, D. Metaxas. *Dynamic 3d models with local and global deformations: Deformable superquadrics.* IEEE Transactions on Patern Analysis and Machine Intelligence, 13(7), 1991, pp. 703 - 714

[Voronoi07] G. Voronoi. *Nouvelles applications de paramétres continus á la théorie des formes quadritiques. Premier Mémorie: Sur quelques propriétés de formes quadritiques* *positives parfaites*. Journal fur die Reine and Angewandte Mathematic 133, 1907, pp. 97 - 178

[Varnuska02] M. Varnuška. *Rekontrukce povrchů geometrických objektů z roztroušených bodů*. Diplomová práce, University of West Bohemia, Pilsen, 2002

[Varnuska03] M. Varnuška, I. Kolingerová. *Improvements to surface reconstruction by CRUST algorithm*. SCCG 2003 Budmerice, Slovakia, pp. 101 - 109

[Varnuska04a] M. Varnuška. *Surface reconstruction from scattered point data*. Technical report, ZCU, Pilsen, Czech Republic, 2004

[Varnuska04b] M. Varnuška, I. Kolingerová. *Manifold extraction in surface reconstruction.* ICCS 2004, Krakow, Poland, pp. 147 - 155

[Varnuska04c] M. Varnuška, I. Kolingerová. *Boundary filtering in surface reconstruction*. ICCSA 2004, Assissi, Italy, pp. 682 - 691

[Varnuska05a] M. Varnuška, J. Parus, I. Kolingerová. *Simple holes triangulation in surface reconstruction*. Algoritmy 2005, Slovakia, pp. 280 - 289

[Veltkamp92] R. C. Veltkamp. *Closed object boundaries from scattered points*. PhD thesis, Center for Mathematics and Computer Science, Amsterdam 1992

[Yu00] Y. Yu. *Surface reconstruction from unorganized points using self-organizing neural networks.* In Proc. of IEEE Visualization, 1999, pp. 61 - 64

Appendix: Activities

Publications related to the dissertation

- M. Varnuška, I. Kolingerová. *Boundary filtering approach in surface reconstruction*. International Journal of Computational Science and Engineering, Inderscience publishers, 2005, ISSN 1742-7185, accepted for publication
- M. Varnuška, J. Parus, I. Kolingerová. *Simple Holes triangulation in surface reconstruction*. Algoritmy 2005, Podbáňské, Slovakia, Vydavaťelstvo STU, Bratislava, ISBN 80-227-2192-1, pp. 280 - 289
- M. Varnuška, I. Kolingerová. *Manifold extraction in surface reconstruction*. International Conference on Computational Science (ICCS) 2004, Krakow, Poland, Springer Verlag, Heidelberg, ISBN 3-540-22129-8, pp. 147 - 155
- M. Varnuška, I. Kolingerová. *Boundary filtering in surface reconstruction*. International Conference on Computational Science and Its Applications (ICCSA) 2004, Assissi, Italy, Springer Verlag, Heidelberg, ISBN 3-540-22056-9, pp. 682 - 691
- M. Varnuška. *Surface reconstruction from scattered point data*. Technical report, University of West Bohemia, Czech Republic, 2004
- M. Varnuška, I. Kolingerová. *Improvements to surface reconstruction by CRUST algorithm.* Spring Conference on Computer Graphics (SCCG) 2003, Budměrice, Slovakia, Comenius University Bratislava, ISBN 80-223-1837-X, pp. 101 - 109
    - this paper won the Springer 2nd Best Paper Award on the SCCG conference
    - proceedings also published under ACM, New York, ISBN 1-58113-861-X
- M. Varnuška. *Rekonstrukce povrchů geometrických objektů z roztroušných bodů*. Master thesis, University of West Bohemia, Czech Republic, 2002

Other publications

- G . A. Triantafyllidis, M. Varnuška, D. Sampson, D. Tzovaras, M. G. Strintzis. *An efficient algorithm for the enhancement of JPEG coded images.* Computers & Graphics: An International Journal of Systems & Applications in Computer Graphics, Volume 27, Issue 4, August 2003, ISSN 0097-8493, pp. 529 – 534

Related talks

- M. Varnuška, I. Kolingerová. *Surface reconstruction from scattered point data.* Technical Univesity of Graz, Austria, 28.10.2003
- M. Varnuška, I. Kolingerová. *Surface reconstruction from scattered point data.* Center of Computer Graphics and Data Vizualization, University of West Bohemia, Pilsen, Czech Republic, 28.11.2003
- M. Varnuška, I. Kolingerová. *Rekonstrukce povrchů geometrických objektů z roztroušených bodů.* Invited talk at the Technical University of Ostrava (VSB), Czech Republic, 14.1.2004
- M. Varnuška. *Introduction to Topology.* Center of Computer Graphics and Data Visualization, University of West Bohemia, Pilsen, Czech Republic, 12.3.2004
- M. Varnuška, J. Kohout, I. Kolingerová. *Surface reconstruction from scattered point data.* Technical Univesity of Graz, Austria, 14.9.2004
- M. Varnuška, I. Kolingerová. *Rekonstrukce povrchů geometrických objektů z roztroušených bodů.* Invited talk at the Technical University of Ostrava (VSB), Czech Republic, 7.12.2004
- M.Varnuška, A. Jeměljanov, J. Parus, I. Kolingerová. *Rekonstrukce povrchů geometrických objektů z roztroušených bodů.* Obhajoba FRVŠ grantu G1/1349/2004, 8.2.2005.

Planned talks

- M. Varnuška. *Rekonstrukce povrchů geometrických objektů z roztroušených bodů.* Center of Computer Graphics and Data Visualization, University of West Bohemia, Pilsen, Czech Republic, May 2005
- M. Varnuška. *Surface reconstruction from scattered point data*. University of Maribor, Slovenia, May 2005

Grants

- FRVŠ G1/1349/2004 – *Rekonstrukce povrchů geometrických objektů z roztroušených bodů* – project leader

- MSM 235200005 – *Information technologies*
  – sub-project: Computer Graphics and Data Visualization - researcher
- AKTION 36p9 – *Bilateral Research Cooperation in the Geometric Models Construction and Visualization for Virtual Habitat and Virtual Archaeology*
  – cooperation with TU Graz, Austria, researcher
- KONTAKT 16-2003-04 – *Algorithms of Applied Computational Geometry*
  – cooperation with University of Maribor, Slovenia, researcher

Stays abroad

- TU Graz, Austria, September 2004, 2 weeks
- TU Graz, Austria, October – November 2003, 4 weeks
- Aristotle University of Thessaloniki (Αριστοτελιο Πανεπιστιμιο Τηεσσαλονικι) Greece, February – June 2001, one semester

Planned stays

- University of Maribor – May 2005, one week

Teaching activities

- 2002-2003: SOJ - Machine Oriented Languages (assembler x86)
- 2002-2003: ZPG - Fundamentals of Computer Graphics (OpenGL)
- 2002-2003: ZIT - Fundamentals of Information Technologies (Microsoft Office)
- 2003-2004: POT - Computer Systems (assembler H8S)
- 2004-2005: ZPG - Fundamentals of Computer Graphics (OpenGL) for ERASMUS students