table of contents

1. introduction

The past years have dramatically decreased the prices of three dimensional scanning equipment as well as the visualization possibilities of modern computers. It is possible to use the 3D models in applications which need to work with the piecewise linear approximation of the objects surfaces, such as in computer graphics, CAD, scientific visualizations, even in non-computer science applications, such as medicine, architecture or archeology.

For example, the National Council of Canada has conducted several projects over the past 18 years of scanning historical artifacts ranging from oil painting to archaeological sites. Zheng, of the Kyushu Institute of Technology in the collaboration with the Museum of Qin Shihuang Terra Cotta Warriors and Horses is conducting now an extensive scanning project to build models of relics found at the site.

Well known in this area is the Digital Michelangelo project [MLe00], solved at the Stanford university. The leader professor Marc Levoy and his students have been working on this project since 1992, they have developed methods for digitizing the shape of three-dimensional objects using laser scanners and algorithms for the surface reconstruction. From 1998 they moved to Italy promoted by Italy government and scanned there 10 historical statues from different museums, the Michelangelo's David statue (Fig. 1.1) is the best known and biggest, it is not till today fully reconstructed because of the data amount. The biggest reconstructed statue St. Matthew contains 386,488,573 polygons.

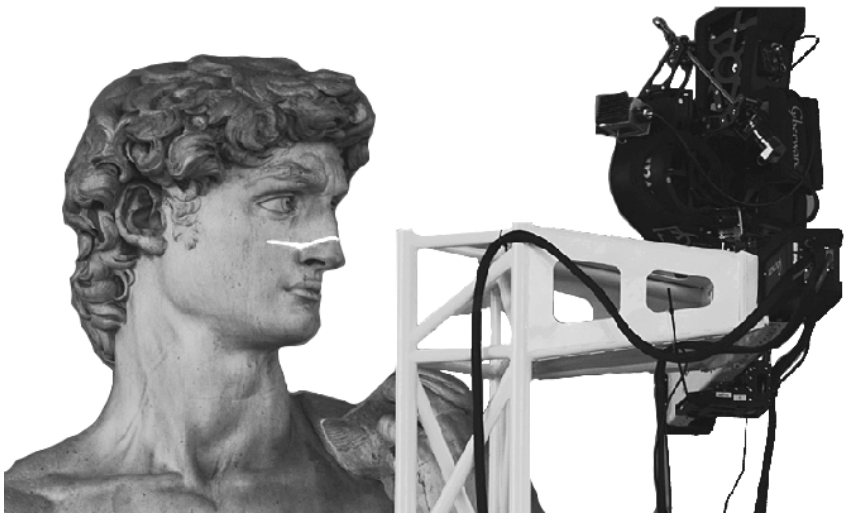


Fig. 1.1: An example of David's statue scanning (courtesy of Levoy at al).

The real object scanning is the first piece of the 3D computer model acquiring mosaic. The 3D data - points lying on the object - should pass then through several next steps such as data filtering, segments clustering, triangle mesh smoothing, NURBS fitting etc. This report orients to the most important step of this pipeline, the surface reconstruction. The input to our “black box”, surface reconstruction method, is the set of 3D point cloud without any

additional information, such as normal vectors, and on the output of the surface reconstruction we would have the surface triangle mesh of the scanned object.

This report will be organized as follows. The first chapter presents the introduction to the given problem, it shows the basic principles of the data acquiring and sampling. The second chapter orientates to the explanation of the Voronoi diagram, Delaunay tetrahedronization and the basic topological terms. The third chapter focuses to the state of the art in the problematic of surface reconstruction and the description of popular algorithms, while the fourth, fifth and sixth chapters elaborate the CRUST and COCONE algorithms and theirs problems. The improvements published on the conferences are in the seventh chapter and the future work is analyzed in the eight chapter. The ninth chapter concludes this report.

1.1. data acquiring

The shape of 3D 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 [FBe00]. On the high cost end, the object can be CAT (computerized axial tomography) scanned and the object surface obtained with 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 fulfill these conditions:

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

It is not simple to satisfy all these conditions, especially the condition that there should not be holes in the scanned surface. The scanned object usually sits or hangs somewhere, so there exists places of contact invisible for the scanner and they look as the nonexistent boundaries in the model, or there can be some inner hole 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[1] are scanning resolution and accuracy. Accuracy is a statement 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 submillimeter accuracy, but because the stripes have some width, the system is able to scan data over the surface in a millimeter resolution.

1 This word is used in the text for any device for data acquiring

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. 1.2) 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, sample 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 color or a normal vector, which can substantially simplify the surface reconstruction.

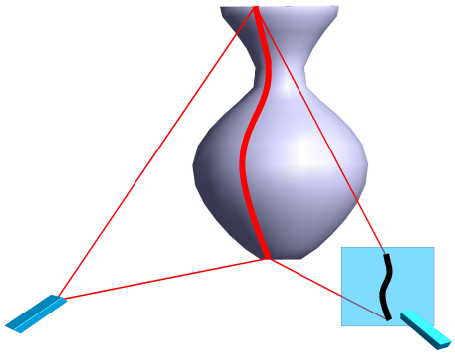


Fig. 1.2: Principle of E^3 scanning by laser scanner.

1.2. input and output

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 obtain 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

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

- *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 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 at Fig. 1.3.

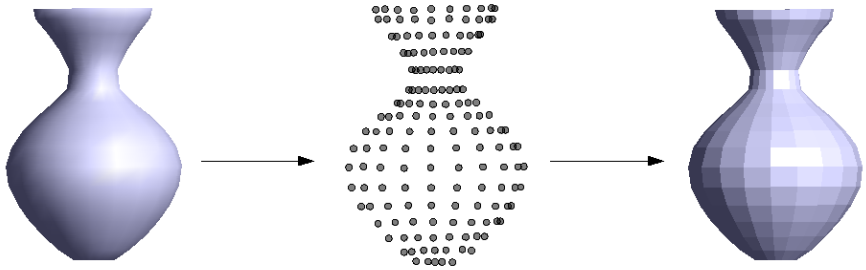


Fig. 1.3: 3D model acquisition pipeline. Some 3D object is scanned and the sampled data are used for the computer model reconstruction.

1.3. sampling

The surface (triangle mesh) obtained after the reconstruction from the set P will be just a surface approximation of the object because some information is lost during the process of digitization. E.g., there is no way how to obtain an exact reconstruction for the surfaces with sharp edges (Fig. 1.4) 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.

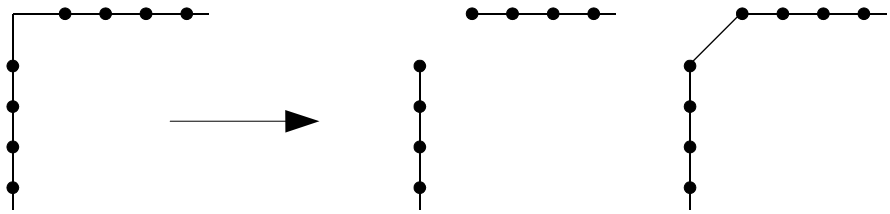


Fig. 1.4: Some reconstruction possibilities of the sharp edge.

Shannon sampling theorem presents the relation of f_{nyq} to sampling frequency. It says 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 criterion is based on 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. 1.5 presents an example of ε sampling path. The smooth part of the surface (left) and the sharp edge are sampled according to the ε sampling path, which is visible on the circles with radius ε. The reconstructed edge is on the right. It is visible that this kind of sampling produces the uniformly sampled datasets without regards to the scanned object features.

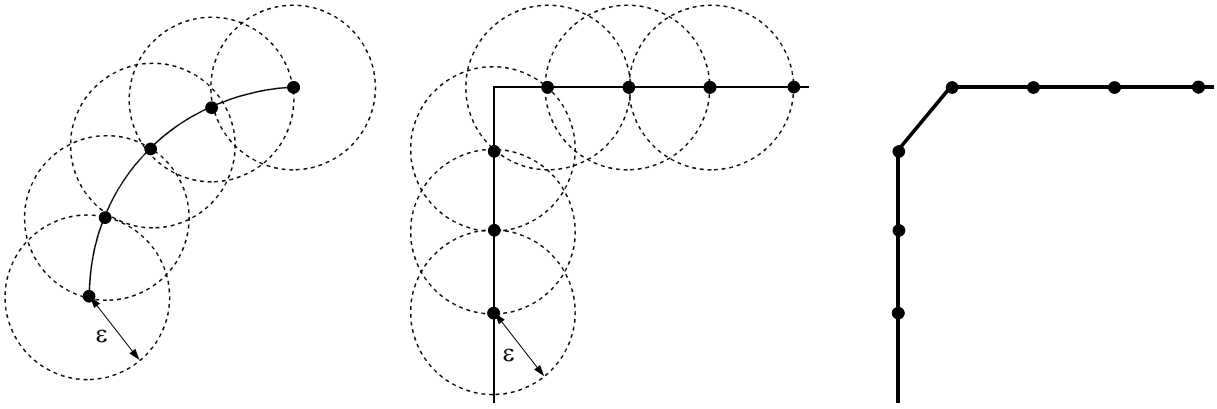


Fig. 1.5: example of the sampling with the sampling path ε.

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 6).

Successful reconstruction depends on the ε-sampling [NAm98b]. 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 $\varepsilon\, LFS\,(s)$. It is proved in [NAm99] 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 using it we can show, why this class of algorithm has a problem with the reconstruction of sharp edges (Fig. 1.6). On the left there is a projection of a part of a smooth surface, the big point represents the medial axis. The *LFS* of each point on the surface is the same. To achieve the ε-sampling good enough, so around 0.3 in this case (it is the rate of the distance of two closest black points and the distance of one point to the medial axis), the surface must be scanned uniformly as the visible points present.

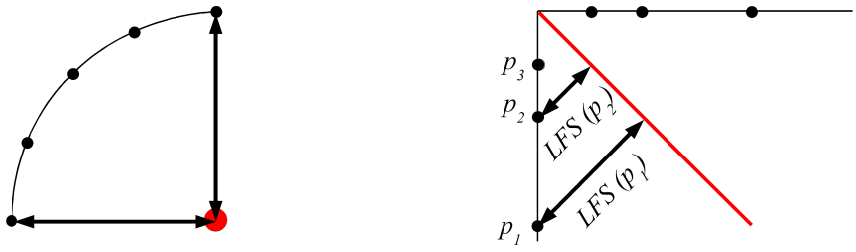


Fig. 1.6: An example of the projection of two surfaces, left is a part of the sphere surface, right is a surface with a sharp edge.

The other case is presented in the right part of Fig. 1.6. The medial axis painted bold touches the sharp edge. Because we want to achieve the ε-sampling at least 0.5 to be just enough for the reconstruction, the closest point to the point p_1 must be in the $0.5*LFS\ (p_1)$ distance. The point p_2 satisfies the condition. The similar condition $0.5*LFS\ (p_2)$ has to state

for the point p_2 whose appropriate closest point is p_3 and we continue till we touch the point in the corner. But that is impossible because we need an infinitely dense sampling.

The big advantage of this kind of sampling is that on the planar part of a surface, with regards to the distance to the medial axis, it is not necessary to have many sampled points, but on the places with small details the sampling density changes to handle them.

2. basic terms

This chapter describes the most frequently used terms in this work. Due to the algorithm used for a reconstruction, the concept of Delaunay triangulation, Voronoi diagrams and basic topological terms will be presented.

2.1. basic topological terms

This report deals with the problem of surface reconstruction in E^3, thus in an *Euclidean space*. 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:

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

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

$$E^1 \subset E^2 \subset ... \subset E^{n-1} \subset E^n$$

For the definition of manifolds we need to explain the definition of an *open unit disk*. The open unit disk is a set of all points (x_1, x_2) in a plane for which states that $x_1^2 + x_2^2 < 1$. This definition can be extended to bigger space than E^2 , then the term an *open unit ball* or an *open unit sphere* is used.

In most papers dealing with the surface reconstruction problems the term *homeomorphismus* appear. This is a topological term and we say that two objects X and Y (object is the set of points in a space) 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*.

The *neighborhood* of some objects X is defined as follows. Let p be a point of some object X, then a *basic neighborhood* is a set of points in X which lie strictly within the distance $d \in R$ from p. By a *neighborhood* we mean the subset of X containing the basic neighborhood. The definition of these terms is in most mathematic books more complicated but for our purpose it is enough.

The *2-manifold* is the set of all points in a space and it holds that all points must have the neighborhood homeomorphic to the open planar disk. We denote the 2-manifold as the *manifold* in this report. Finally, the *surface* is a compact manifold, the term compact means that the manifold consists only of one component.

In our approach we are working with the surface which contains boundaries so we need to improve a little the definition of the manifold and add the definition of a *manifold with the boundary*. This manifold has a neighborhood homeomorphic to the open planar disk while the boundary points have a neighborhood homeomorphic to the open half-planar disk.

2.2. Delaunay triangulation

A finite point set $P \subseteq E^3$ defines a special triangulation known as a *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 [BDe34]. 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, $\forall b \cap P = \varnothing$. 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, the tetrahedronization is unique except these places locally,
- the boundary of tetrahedronization forms the convex hull,
- the Delaunay tetrahedronization minimizes the maximum radius of a simplex enclosing sphere,
- some edges of Delaunay tetrahedronization belong to the following graphs:

$$EMST \subseteq RNG \subseteq GBG \subseteq DT$$

  - *EMST* is the Euclidean Minimum Spanning Tree,
  - *RNG* is the Relative Neighborhood Graph,
  - *GBG* is 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 $DT(P)$. Fig. 2.1 shows a planar Delaunay triangulation of some point set sampled from some object.

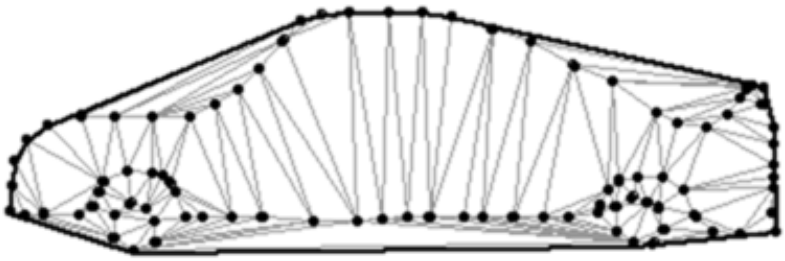


Fig. 2.1: The Delaunay triangulation of the black points, black edges present the convex hull, gray edges are the triangulation edges.

2.3. Voronoi diagram

The concept of *Voronoi diagrams* was first introduced by G. Voronoi in [GVo07]. Voronoi diagram is a space partitioning which decomposes the space into the convex polyhedral cells. 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.

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

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 the 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. Fig. 2.2 shows the Voronoi diagram of the set of points from Fig. 2.1.

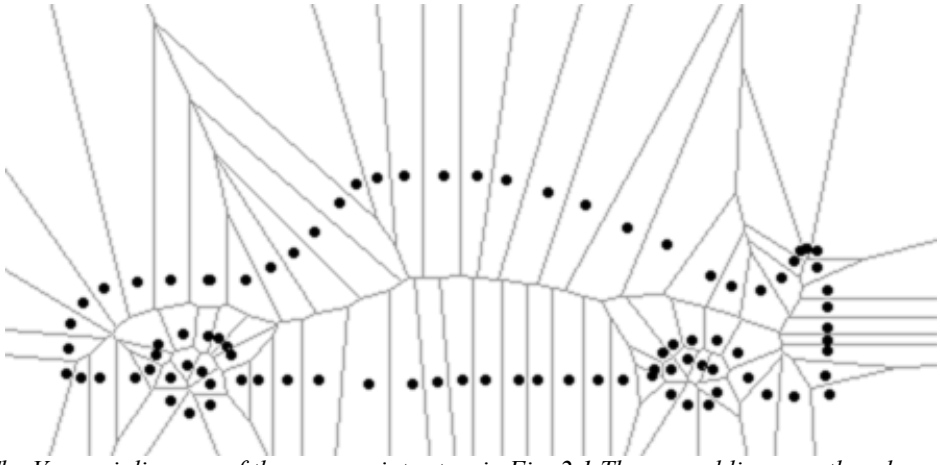


Fig. 2.2: The Voronoi diagram of the same point set as in Fig. 2.1.The grayed lines are the edges of the Voronoi diagram.

An important property of the Voronoi diagram and Delaunay tetrahedronization is that they are mutually dual (Fig. 2.3). 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. 2.4).

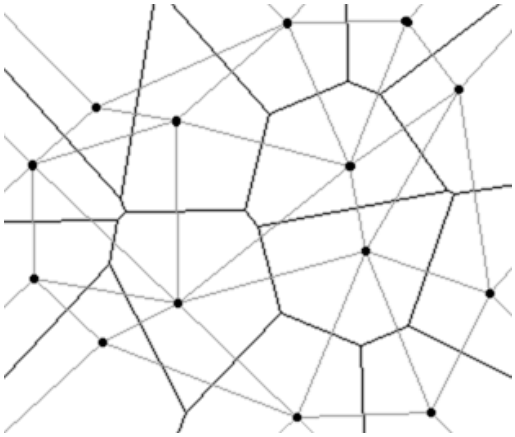


Fig. 2.3: A planar example of Voronoi diagram in gray and Delaunay triangulation in black.

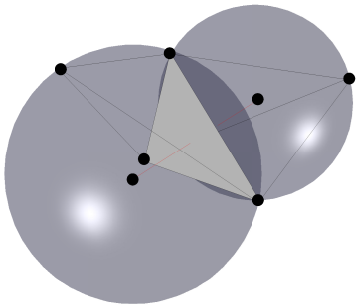


Fig. 2.4: The gray triangle presents the shared triangle (face) of two tetrahedra. The dual representation of this triangle in the Voronoi diagram is the bold edge which connects the centers of two tetrahedra circumspheres.

3. existing algorithms

Many methods for solving the problem of surface reconstruction were developed during recent years. For curve reconstruction, the E^2 version of our problem, many methods exist with strong theoretical background (e.g., [NAm98a], [TKD99], [TKD00], [DAt97]). The E^3 problem has been addressed by many researchers in computer graphics and computer vision. Methods can be divided into four groups [RMe98a] (division is not strict, some methods can belong to more groups):

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

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 Müller's approach [JVM91].

The idea of warping is relatively old and basic methods are e.g. Muraki “blobby model” for 2.5 mesh approximation [SMu91] or deformable superquadrics by Terzopoulos [DTe91a], [DTe88]

Other approach was introduced by Szeliski [RSz92]. He uses oriented particles, whereas every particle has some parameters whose values are updated during the modeling simulation. Every sample point has a particle with corresponding parameters and the surface is created by an interaction modeling between particles (attraction x repulsion).

Boissonat [JDB84] presented another approach (the method of incremental surface reconstruction) how to obtain the model from point clouds. It begins on the shortest edge from all edges between points and incrementally appends points to create a triangle mesh. Mencl and Müller [RMe95] [RMe98b] developed a similar algorithm, its main characteristic is creation of extended minimum spanning tree, identification and extraction of typical features and using these properties for triangle mesh extraction.

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 [HHo94], [HHo92] gave an algorithm, where the surface is represented by the zero set of a signed distance function.

Curless and Levoy [BCu96] 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 [MLe00].

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 to independent areas. A typical example is the division by a regular grid, adaptive by an octree or an irregular tetrahedronization. We find the areas which have some relationship to the surface described by the input set and we extract the surface from these areas.

Algorri and Schmitt [MEA96] 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. Edelsbrunner and Mücke [HEd92] [HEd94] 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 was developed to bypass this limitation.

Bernardini and Bajaj [FBe97] 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[FBe99] describes an 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 is very fast and it can handle millions of points.

Amenta introduced a concept of CRUST, it has two versions, two-pass [NAm98b] and one-pass [NAm99] [NAm00]. 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. Dey extends the ideas of Amenta and gave an effective COCONE algorithm. The basic idea is presented in [NAm00]. Other papers presented by Dey introduced the way how to handle large data [TKD01b], which is the common problem of Delaunay based algorithms, and what to do with boundaries [TKD01c], undersampling and oversampling [TKD01a]. 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. Both authors gave an algorithm for a watertight surface reconstruction, Nina Amenta her PowerCRUST based on medial axis transformation [NAm01] and Tamal Dey his TightCOCONE based on tetrahedra removal [TKD03].

In the next subsection we orientate to the description of the most popular algorithms. The algorithms are chosen to cover different ways of surface reconstruction.

3.1. Hoppe's distance function approach

Hoppe presented his approach in [HHo92]. It uses the signed distance function, which means the closest distance from the point to the surface. For the closed surfaces the sign is negative if the point lies inside the object and positive otherwise.

First the estimation of the tangent plane is counted for every point $p_i \in P$ using *k*-nearest neighbors to the point p_i (Fig. 3.1). As we have to know which sign the function has in the case of the closed surface, the planes have to be oriented to the same direction using *Riemannian graph*. Nodes of the graph are all centroids σ_i (the arithmetic center of the *k*- nearest neighbors). Two nodes *i*, *j* (of the point p_i and p_j) are connected by the edge (*i, j*) if one node is in the *k*-neighborhood of the other node. Each edge has an evaluation computed as $1-|n_i \cdot n_j|$, where n_i and n_j are the normal vectors of the tangent planes in the point p_i and p_j. The *minimum spanning tree* (MST) is then obtained from the *Riemannian graph* (Fig. 3.2).

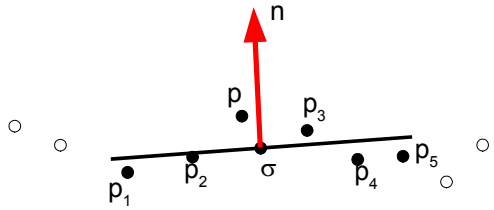


Fig. 3.1: The estimation of the tangent plane for the point p. Points p_1-p_5 are the closest points to p used for estimation computing, σ is the centroid and n is the computed normal vector of the tangent plane.

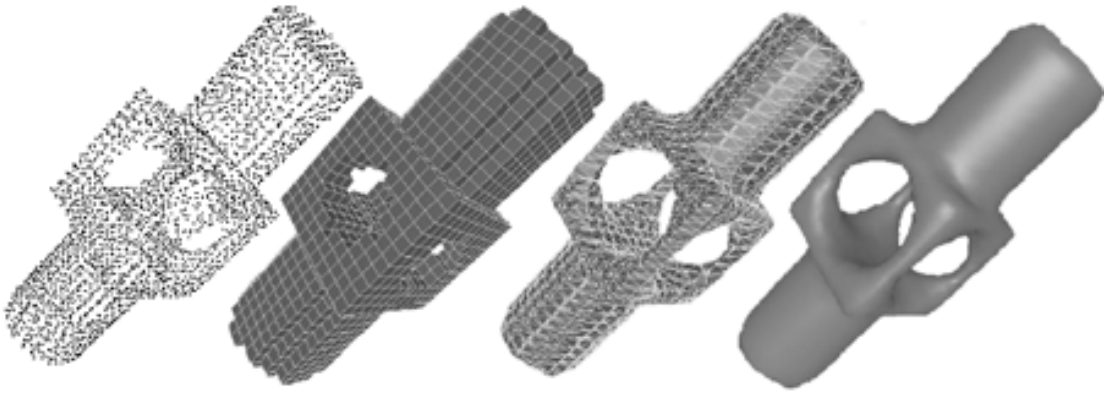
The algorithm starts with the orientation on the point whose centroid has the largest *z* coordinate. The normal vector of the tangent plane of this point has to direct to the positive z direction, if not, the orientation is changed. Then, rooting the minimal spanning tree at this initial node, the tree is traversed in depth-first order. To each tangent plane an orientation is assigned consistent with its parent.

a) *b)* *c)*

Fig. 3.2: a) The Riemannian graph, b) the minimum spanning tree created from the Riemannian graph, c) the visualization of the distance function (courtesy of Hoppe).

Using the first surface approximation from the tangent planes, the value of the distance function for the point p_i is computed as the distance of the point p_i and its centroid σ_i. The sign of the function depends on the tangent plane orientation.

Next step of the algorithm creates the regular space subdivision by the voxel grid and the *marching cubes* algorithm [WEL87] is used for the creation of the surface which has the zero distance function value. We can apply the marching cubes algorithm just to those voxels whose neighboring voxels have different distance function value, so the extraction is faster. The disadvantage of this method is that the obtained surface does not contain the original point cloud and it approximates the surface only. The surface can be improved by other postprocessing steps [Hho93, HHo94]. The example of the reconstruction is in Fig. 3.3.



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

Fig. 3.3: a) The input point cloud, b) the voxelization by the regular grid, c) the triangle mesh obtained by marching cubes algorithm, d) the resulting surface after NURBS fitting (courtesy of Hoppe).

3.2. Mencl and Müller's approach

The main feature of the Mencl and Müller's approach [RMe95, RMe98b] (it belongs to the group of incremental surface reconstruction algorithms) is the use of *euclidean minimum*

spanning tree (EMST), whose computation is the first step of the algorithm. The algorithm assumes that the edges of the EMST are lying on the surface if the surface consists of one component. If more surface components are present then the components are connected by one edge whose length differs from the length of other graph edges.

The EMST is then extended to the *surface description graph* (SDG). The extension is done by connecting the tree leafs with their neighbors under the conditions that the length of the connected edge does not differ so much from the length of the other edges connected to the point and that this edge lies in the cone defined by the end edge of the leaf and an angle (suitable choice is 90°). The formation of SDG prevents the creation of narrow and self-intersecting triangles in next steps. Fig. 3.4 shows an example, the point p_2 is the leaf of the EMST and it is being connected to other graph vertices. The length of the edge $p_1 p_2$ does not differ so much from the other edges length and this edge is in the cone defined by the edge $p_2 p_3$ and some angle so it will be added to SDG.

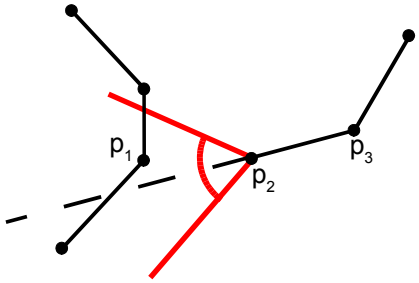


Fig. 3.4: The point p_2 is a leaf of the EMST. The point p_1 lies in the cone defined by the edge $p_2 p_3$ and the angle and the length of the edge $p_1 p_2$ does not differ much from the length of the edge $p_2 p_3$, so the edge $p_1 p_2$ is added to the SDG.

In the third phase the algorithm recognizes the main features of the surface. The surface structures such as a surface edge, a path or a ring are detected using heuristic rules. Each surface structure has its own rule.

After the feature extraction the structures are connected or disconnected according to certain rules to obtain proper objects situated in the point cloud. This step is followed by the last phase - triangle filling of the wireframe. A basic assumption for this is that when the angle between two connected edges makes smaller angle than the triangle generated by these edges belongs to the surface. The whole triangulation is then generated incrementally by choosing a pair of incident edges in a greedy fashion with respect to the following rules:

- At most two triangles may be incident to an edge.
- Triangles may not intersect each other.
- For all triangles *abc* and all points *p*, whose orthogonal projection fall to the triangle, at least one of the angles *abp*, *apc*, *pbc* and the triangle *abc* exceeds the angle 45°.
- The angle between two faces at their incident edge must not be less than some constant.

The triangles are written as other edges to the SDG so no other data structures are necessary (see an example of reconstruction in Fig. 3.5). The problem of the method is the amount of the parameters and that all rules are heuristics. The output of the algorithm is

a piecewise linear surface which exactly interpolates the input point set. It can handle points sets with different point sampling density so the surface can be described according to the local curvature. Due to the feature extraction the reconstruction of sharp edges or non-orientable surfaces is proper.

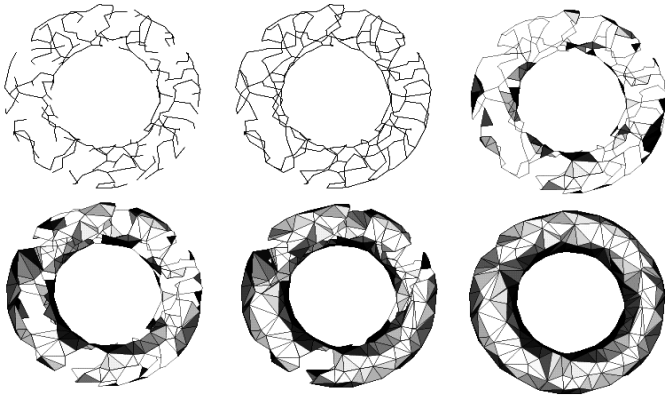


Fig. 3.5: From the top left - the euclidean minimum spanning tree of the input set, the surface description graph and the reconstruction after 100, 250, 450 and all triangles extraction (courtesy of Mencl at al).

3.3. Kohonen feature map

This method [ABa93, ABa94] belongs to the warping surface reconstruction methods and it uses the Kohonen neural network for a reconstruction. This kind of network is a single-layered neural network with forward propagation and learning without a teacher. The output value of the neuron is defined as a distance between input and weight vectors. Other areas of use are in data recognition, clustering or semantic map creation.

In a surface reconstruction approach this neural network is mapped to the input point cloud, one neuron belongs to one point. Because the Kohonen network is two-dimensional, it can be used only for a 2.5D surface reconstruction. Each neuron u_i has the weight vector w_i. These vectors are filled in the beginning by random normalized numbers (so the length of the vector is one). During the process of neuron training (the process of surface reconstruction) are the neurons inputs fed by input data which change the weight vectors (and the position in the space). The input vector i is put to the neuron j, so the output o_j is the dot product of the neuron weight vector w_i and the input vector i.

$$o_j = \vec{w}_j \cdot \vec{i}$$

The neuron which generates the highest response (the highest output value o_j) is called the *center of excitation area* and the weights of this neuron and its neighbors are actualized by:

$$\vec{w}_j(t+1) = \vec{w}_j(t) + \epsilon_j(\vec{i} - \vec{w}_j(t))$$

The value ε_j is computed by a multiplication of the constant η which describes the weight of learning and h_j which describes the influence of current neuron to the neighboring neurons (the neurons far away have to be affected by this neuron less than by closer neurons):

$$\epsilon_j = \eta h_j$$

After the computation the vectors have to be normalized again. The problem with the algorithm appears when the neurons do not respond to the input point cloud. Then some neurons are far away from the center of excitation area and are affected only by the nearest neighbors which is not very sufficient to place the neurons to the real surface position.

Due to this problem, Baader and Hirzinger have developed the reverse training method. In the classic method the corresponding neuron is created and actualized for each input point while the reverse method works in a different way. For each neuron the input with the highest influence is found and this input is utilized for the neuron actualization. In the implementation of the algorithm the combination of both training methods is used. Fig. 3.6 shows the schema of the reconstruction.

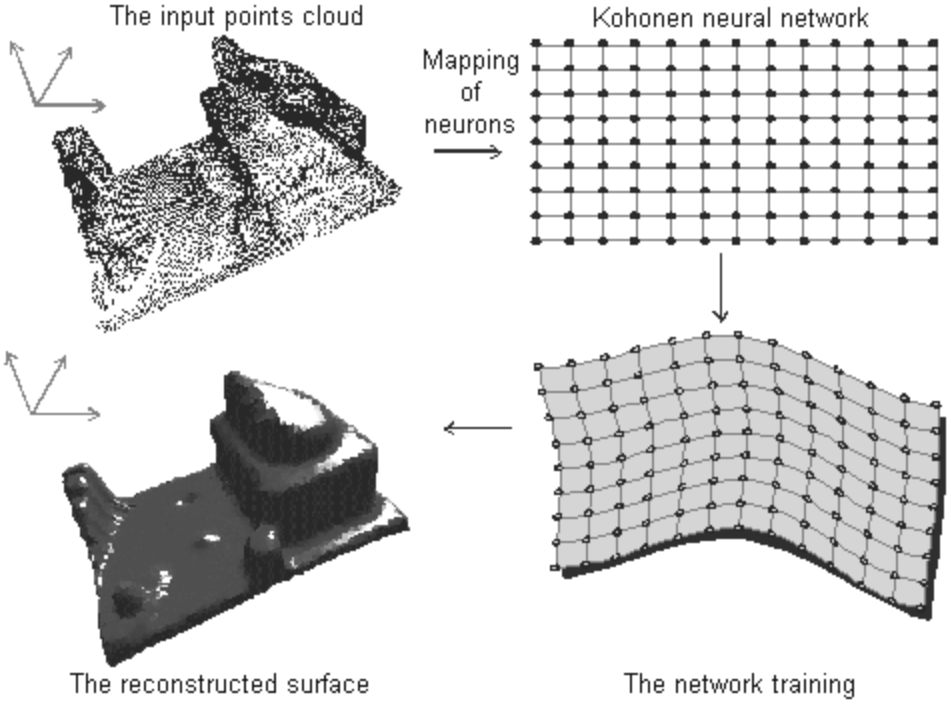


Fig. 3.6: The schema of the reconstruction by the Kohonen neural network.

3.4. Bittar's reconstruction using medial axis

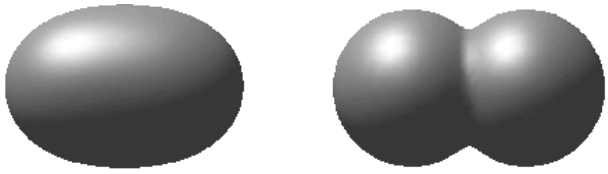
This approach [EBi95] combines the medial axis with the implicit surface. It consists of two main phases. The first phase computes the medial axis of the input point cloud. This is done by the space partitioning into a regular grid. The size of voxels impacts the resolution of the reconstructed surface, the smaller the voxels are, the finer is the reconstruction. Each voxel is labeled as *outside* if it does not contain any point or as *border* if a point is present. Then we propagate the information about the outside voxels from one edge of the bounding box to the neighbors, without passing through the voxels labeled as border. The voxels which are not affected by this process are *inside* the object.

The distance map computation and medial axis extraction is then applied to the grid. The distance map is computed using *Chamfer* distance for faster computation (the distance in a grid, “the minimum number o voxels we have to pass walking from one into another”)

instead of Euclidean. The value (weight) of the distance function is zero in the *border* voxels and it is propagated to the direction of *inside* voxels. The weight of each voxel is updated in the propagation cycle using two filters, the voxels with the highest weight belong to the medial axis.

The surface of some object is calculated using the distribution of the sphere centers on the medial axis. The radius of the sphere is equal to the distance assigned to its center on the medial axis. The *field function* is defined for each sphere. This function allows to compute the scalar field for every point in the space. The function value for the whole sphere set is defined as the sum of all function values of each sphere. The implicit surface is defined as the isosurface of the field function, so the isosurface consists of all points which have the constant value of the field function. The process of computation is time consuming so a special strategy for the computation of points lying in proper positions is used.

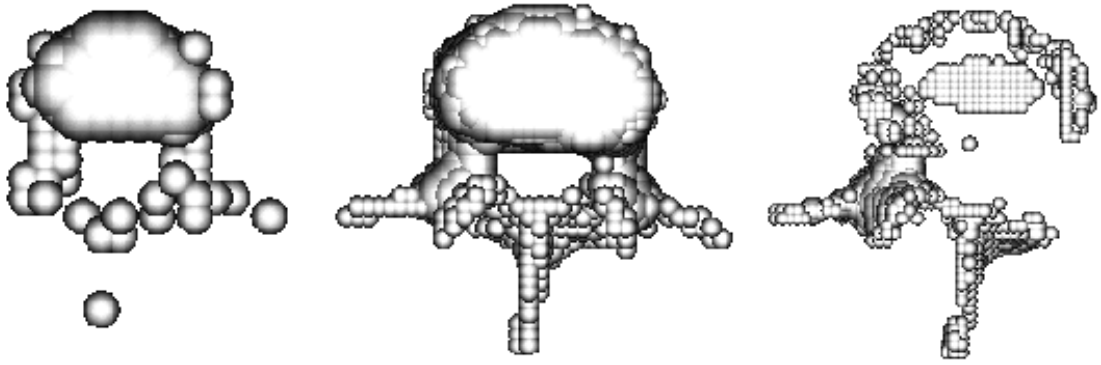
The shape of the reconstructed surface depends strongly on the field function type. Fig. 3.7 shows the difference between two types of the function, the sharp function prevents the details to be smoothed while the soft function smooths the details. The function may influence continuity of the reconstructed surface, too.



a) *b)*

Fig. 3.7: Two kinds of field. functions, a) soft function smoothing details, b) sharp function preserving details (courtesy of Bittar).

The grid generation is a crucial phase of the reconstruction success. When the resolution is small then the details will be smoothed. When the resolution is higher then the details are more preserved, but when the resolution is too high, the algorithm disconnects the surface to more components because the resolution is higher than the point sampling resolution (Fig. 3.8).



a) *b)* *c)*

Fig. 3.8: Three resolution sizes of the voxel grid and their influence on the reconstruction, a) the resolution is too low and details are lost, b) the resolution is properly set and the reconstruction is correct, c) the resolution is too high, the surface can split up (courtesy of Bittar).

3.5. Eddelsbrunner's and Mücke's α - shapes

This algorithm [HEd92, HEd94, EPM93] uses the Delaunay tetrahedronization as the first step of the algorithm and it chooses the surface triangles as a subset of Delaunay triangles. Conceptually, α-shapes are the generalization of a convex hull of the input point set, the parameter α means the radius of some sphere called α –ball. Let P be a finite set of points in E^3 and $\alpha \in <0, \infty)$ be a real number. The α-shape of P is a polytope that is neither necessarily convex nor necessarily connected. The α-shape is identical to the convex hull for $\alpha = \infty$ and when the α value decreases, the α-shape shrinks by gradually developing cavities. These cavities may join to form tunnels and holes may appear. A part of the polytope disappears when α becomes small enough so the sphere with this radius can occupy the space without enclosing any other point. When α value is equal to zero then α-shape is identical to P.

An α-ball b is empty if $b \cap P = \emptyset$. The α-hull, denoted as $\mathcal{H}_\alpha$, is defined as the complement of the unions of all empty balls. It holds that $\mathcal{H}_{\alpha 1} \subseteq \mathcal{H}_{\alpha 2}$ if $\alpha_1 \leq \alpha_2$, sample members of α-hull for $\alpha = \infty$ is the convex hull and the set P for sufficiently small α. Another interesting concept defined by α-balls is called α-diagram, denoted as $\mathcal{U}_\alpha$. The α-diagram is the union of all α-balls whose centers are points of the set P. It is known in chemistry and biology as a space filling diagram and it is not restricted only to the equally large balls (this restriction is removed using weighted α-shapes and α-diagrams). The point x belongs to $\mathcal{U}_\alpha$ if the α-ball b_x centered at x is not empty. The following relationship between $\mathcal{H}_\alpha$ and $\mathcal{U}_\alpha$ holds:

- $x \in \mathcal{U}_\alpha \Leftrightarrow b_x \cap \mathcal{H}_\alpha \neq \emptyset$
- $x \in \mathcal{H}_\alpha \Leftrightarrow b_x \subseteq \mathcal{U}_\alpha$

Consider the boundary of $\mathcal{U}_\alpha$. It consist of spherical caps, circular arcs and vertices which are called corners. These are the 2-, 1- and 0-faces of $\mathcal{U}_\alpha$. The caps, arcs and vertices are in close correspondence with the vertices, edges and triangles in α-shape.

We can imagine the work of the algorithm simply. Imagine that the space is filled with some foam and points are sampled from some object inside the foam. The tetrahedronization creates the Delaunay k-simplices ($k \in <0, 3>$) of the points. We take now a sphere with the radius α and we place it everywhere outside the convex hull. Then we erase all simplices whose radius is greater than the radius of the α-ball. The result is then called the α-shape and it consists of the set of tetrahedra, triangles, edges and points. When the parameter α is ∞, then the α-shape is equal to the Delaunay tetrahedronization, similarly, when the parameter α is zero, then the result is equal to the input point set P.

In the last step the triangles that belong to the resulting surface are extracted out of the α–shape based on the following heuristic: A triangle belongs to the surface if at least one of the two α–spheres that interpolate the triangle's vertices is empty of other points (E^2 example is in Fig. 3.9).

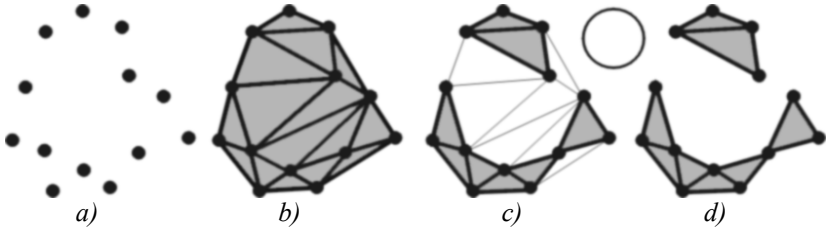


Fig. 3.9: The steps of the algorithm, a) the input points set, b) its triangulation, c) the triangles deleted using the α-sphere (shown above) and d) the extracted triangles.

The algorithm has elegant theoretical formulation, but the problem is with finding the α value which have to be set experimentally. Point clouds with varying points density are problematic for a reconstruction with this algorithm too, this trouble can be avoid using weighted α–shapes. An example of the reconstruction is in Fig. 3.10.

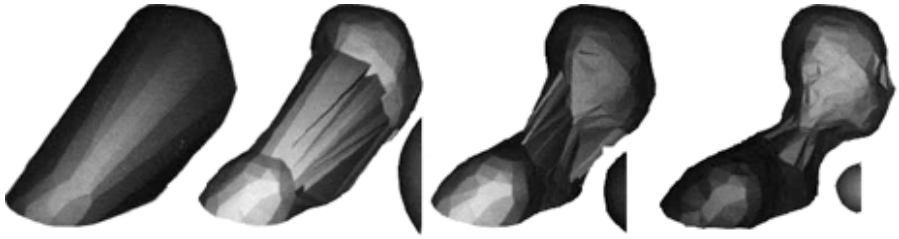


Fig. 3.10: The output of the algorithm using different α-spheres. The sphere is printed right in the figure (courtesy of Eddelsbrunner and Mücke) .

3.6. Attali's normalized mesh

Attali introduced the concept of *normalized mesh* in [DAt98]. The normalized mesh is a subgraph of the Delaunay tetrahedronization and it consists of the edges, triangles and tetrahedra whose dual Voronoi edges intersect the surface. Normalized meshes are attractive for a surface reconstruction because they provide a piecewise linear interpolant of the surface that converges to the surface when the sampling density tends to zero. When the sampling path ε is $1/n$ (where n is some real number), S is the surface and S_n the normalized interpolant of S, then it holds that the limit of S_n with n going to infinity is the original surface S.

In order to simplify the search of the normalized mesh, the *r-regularity* property is used. Denote B_0 the unit ball, then a shape X is called *r-regular* if it is morphologically open and closed with respect to disc of a radius $r > 0$:

$$X = (X \ominus rB_0) \oplus rB_0 = (X \oplus rB_0) \ominus rB_0, \text{ where}$$

- the operator $\oplus$ means the dilatation,
- the operator $\ominus$ means the erosion,
- the expression $(X \ominus rB_0) \oplus rB_0$ is the operation of opening,
- the expression $(X \oplus rB_0) \ominus rB_0$ is the operation of closing.

The concept of r-regular shapes was first introduced in a mathematical morphology, but it has many nice geometric properties, too:

- The boundary of a *r*-regular shape has at each point a tangent and a radius of curvature greater or equal to *r*.
- The boundary of an *r*-regular shape divides any ball with a radius *2r* and center on the boundary into exactly two connected components. If $\varepsilon < 2r$, then the normalized mesh retains all the topological properties of the surface.
- In R^2 space any circle passing through three distinct boundary points has radius greater than *r*. This property is crucial to the algorithm.

The addressed problem is now to determine which faces of the Delaunay triangulation belongs to the normalized mesh. In R^2, Delaunay disks tend to be the maximal disks of the object and they become tangent to the boundary. Let the set of selected Delaunay triangles denote as S_δ, if $\varepsilon < r \sin(\pi/8)$, then the set of edges $S_{\pi/2}$ is the normalized mesh of *S*. Let *X* be a *r*-regular shape of δX with a sampling path ε. For each edge *pq* of the Delaunay triangulation we have:

- If *pq* belongs to the normalized mesh and $\varepsilon < \pi/2$, then $\delta(pq) < \pi/2$.
- If *pq* does not belong to the normalized mesh and $\varepsilon < r \sin(\pi/8)$, then $\delta(pq) > \pi/2$.

Let $B(x_1, r_1)$ and $B(x_2, r_2)$ be two Delaunay discs intersecting in a point *p* and *q*, *m* the middle point of the edge *pq*, then $\delta = \pi - \angle(x_1pm) - \angle(mpx_2)$, see Fig. 3.11:

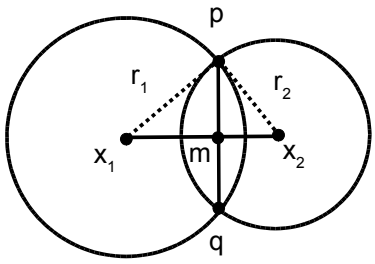


Fig. 3.11: The computation of the angle δ (courtesy of Attali).

It is not necessary to know the parameters *r* and ε to compute the normalized mesh. The sampling path does not need to be close to zero to find the correct result, it is enough that the sampling path ε and the constant *r* characterizing the shape have approximately the same order (Fig. 3.12 shows an example of a curve reconstruction).

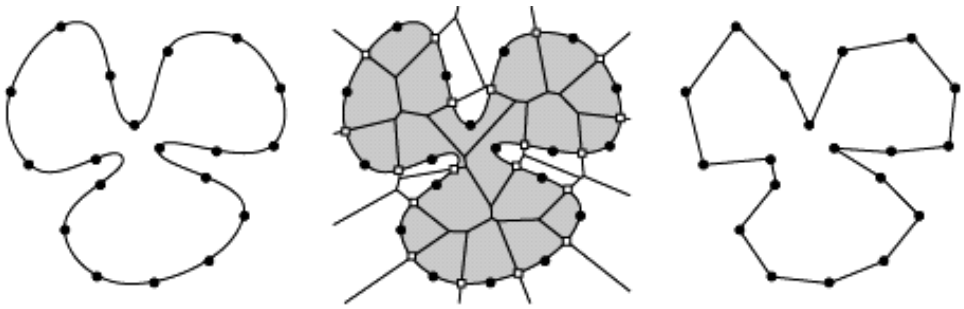


Fig. 3.12: Construction of the normalized mesh from the Voronoi diagram of the input points (courtesy of Attali).

Unless it holds for R^2 case that normalized mesh is a correct reconstruction, direct extension to R^3 is not possible, because some Delaunay spheres intersect the surface without being tangential to the surface. There are two heuristics criteria used for this case. The former is the triangulation of the appeared holes using only the triangles obtained from the Delaunay tetrahedronization.

The latter criterion is based on a volume approach. In the initialization phase it merges all Delaunay tetrahedra and the complement of the convex hull. All Delaunay triangles are put to the list and sorted according to the triangle diameter. While the list is not empty: if the the triangle having the greatest diameter separates two different objects O_1 and O_2, these objects are merged providing that :

- No triangle from $S_{\pi/2}$ disappears.
- The merge does not isolate sample points inside O_1 and O_2.

The surface is then the boundary of the computed sets (Fig. 3.13).

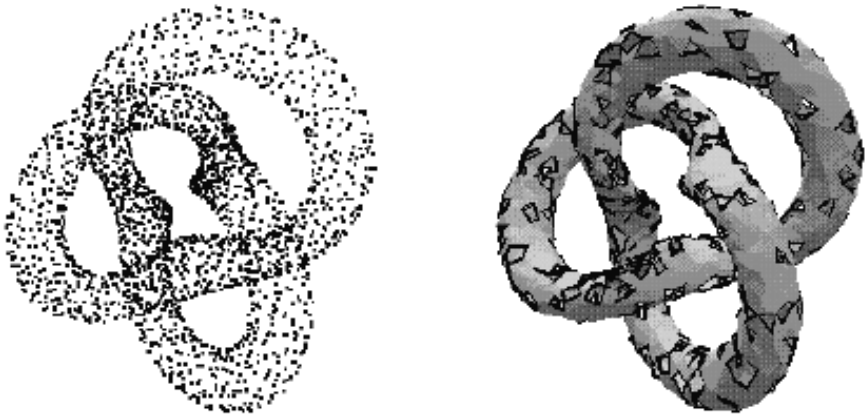


Fig. 3.13: An example of a surface reconstruction by a normalized mesh. Due to the problems presented in the previous text some holes appear in the reconstructed surface, the amount of holes depends on the points distribution (courtesy of Attali).

4. CRUST 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 use of the *DT* was the second reason why to use this algorithm because in our group we have a fast and efficiency code for the *DT* computing developed by Ivana Kolingerová. The surface forms a subgraph of the Delaunay tetrahedronization and information from the dual Voronoi diagram is used for the selection of surface triangles.

The E^2 version of the algorithm was first introduced in [NAm98a] and it is derived from the Attali's normalized mesh algorithm. The name "**crust**" means the set of edges selected from the triangulation forming the curve interpolating the points set. The E^3 version [MAm98b] is built on the same principle and is just the extension to E^3. The disadvantage of this algorithm is its 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 onepass version in [NAm00] which is able to select the surface triangles from the first *DT*.

The CRUST uses the sampling criterion built on local feature size. That is the reason why the algorithm has problem with sharp edges, outliers and boundaries, but also the reason, why it has no problem with the reconstruction of the object with big sampling density changes.

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. The information from the Voronoi diagram is used for the surface triangles selection from Delaunay triangulation. This set we call a primary surface and it has not to be a manifold, so an extraction manifold step is necessary.

4.1. the poles

The algorithm introduces the concept of poles (Fig. 4.1). They are formed by the vertices of the Voronoi diagram (at least three points have equidistant distance here). 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", so the dot product of the vectors $(p-, p)$ and $(p+, p)$ is negative.

For a successfully sampled 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 twopass

algorithm takes the poles as an approximation of the medial axis while the onepass version takes the vectors from the point to the poles as an approximation of the normal vectors.

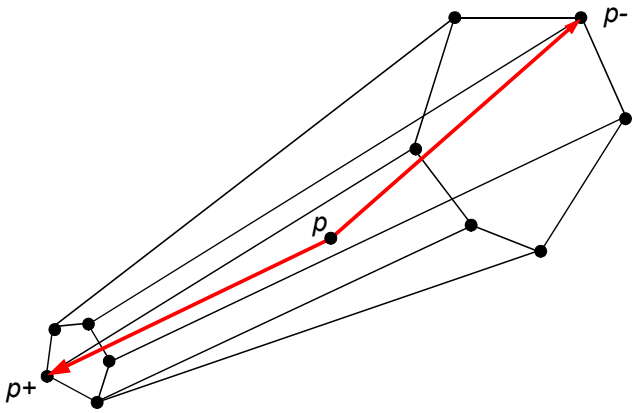


Fig. 4.1: The point p, the Voronoi cell and vertices around it. The vertex p+ is the positive pole, because it is the farthest vertex from the point p, the vertex p- is the negative pole, it is the farthest vertex on the "other side".

4.2. E^2 twopass version

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 so VV lie on or near the medial axis. The VV of the input points approximates the medial axis.

The next step is the 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).

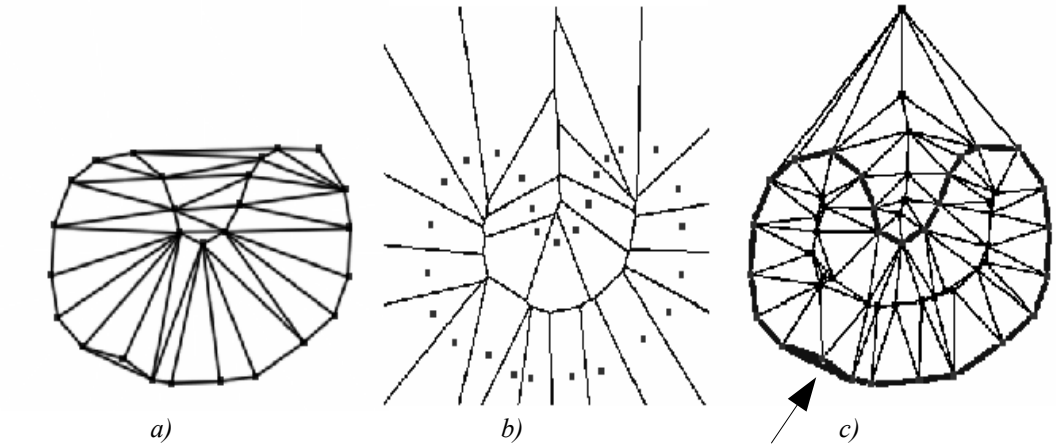


Fig. 4.2: An example of the twopass curve reconstruction, a) the input point cloud with the Delaunay triangulation, b) the Voronoi diagram. When we unify the sets P and V (Voronoi vertices) and triangulate, we get the triangulation c) where curve edges (highlighted) are separates from other edges.

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 ("crust") is a subset of both *DT*. This step is called Voronoi filtering and for sufficiently dense sampling it works perfectly. In Fig. 4.2c) down (where the arrow directs) three points form the triangles. It is a problem because the crust is not a curve and some edges need to be deleted. That is the reason why the manifold extraction step in E^3 has to be performed.

4.3. E^3 twopass version

We can simply extend the E^2 algorithm to the E^3 version. The problem is that the set of Voronoi vertices is huge and the tetrahedronization of the *P* and *V* union is almost impossible due to the big amount of data. The second problem is that not all *VV* lie near or on medial axis, on the other hand we can still found many points for which this condition holds (see Fig. 4.3).

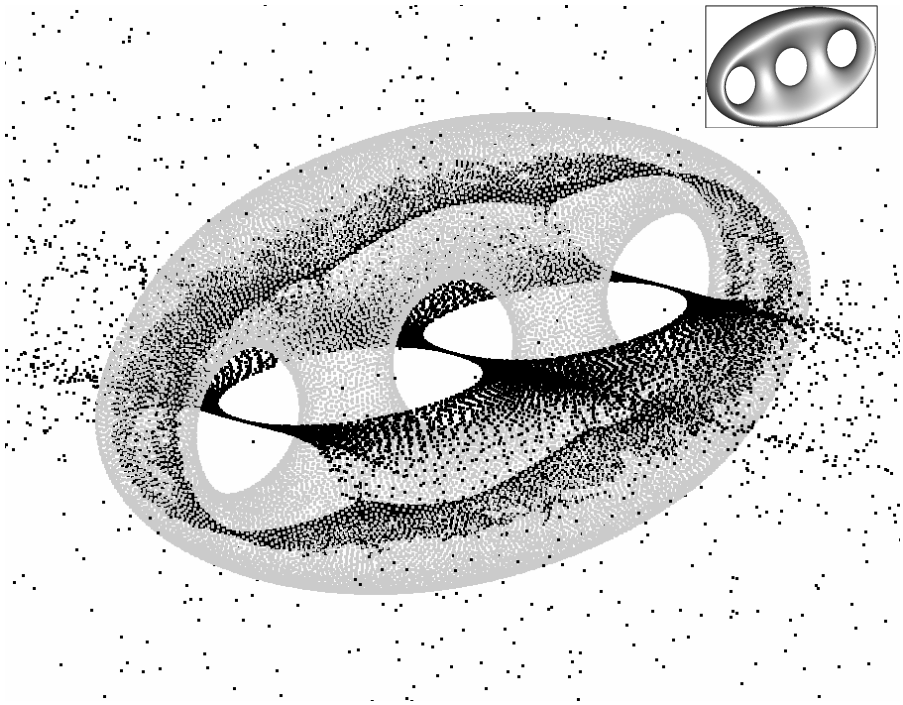


Fig. 4.3: The reconstructed object and its poles (positive and negative poles) in black. Many of them lie on or near the medial axis but some of them are far away. The small figure right up is the original surface.

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 black dots are other points sampled from the surface. It is shown that not all of Voronoi vertices (dots painted different) lie on the medial axis.

The cell is almost orthogonal 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 twopass 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.

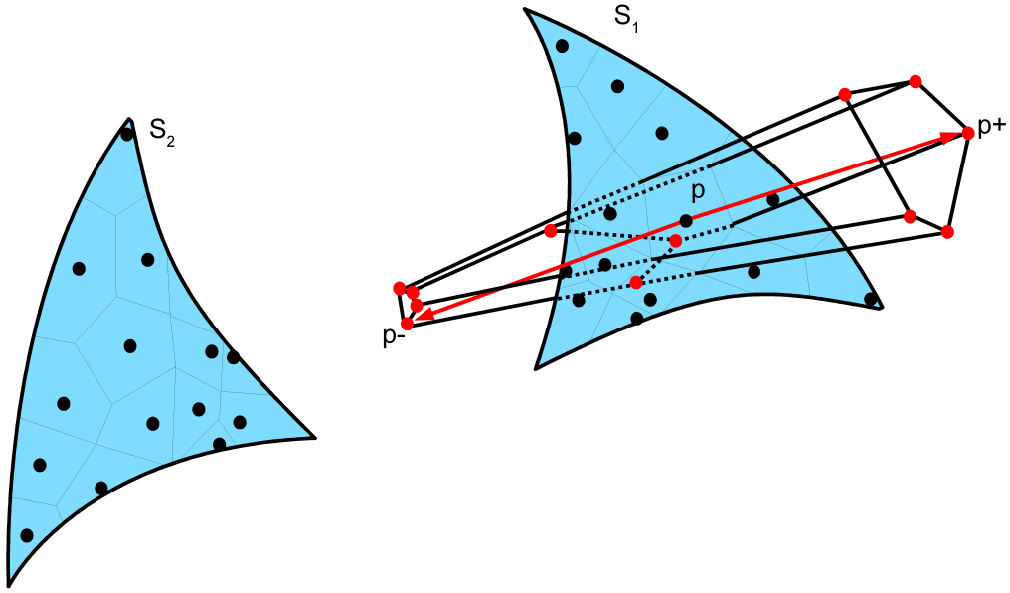


Fig. 4.4: The surface with two parts S_1 and S_2 with the Voronoi cell around the point p. The points (Voronoi vertices) p+ and p- are positive and negative poles of the Voronoi cell.

This improvement makes possible to use the twopass algorithm in E^3 (due to 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. The binding with the theory is provided by the following theorem [NAm98b]:

> *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 author tried to used 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 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 big amount of overlapped triangles.

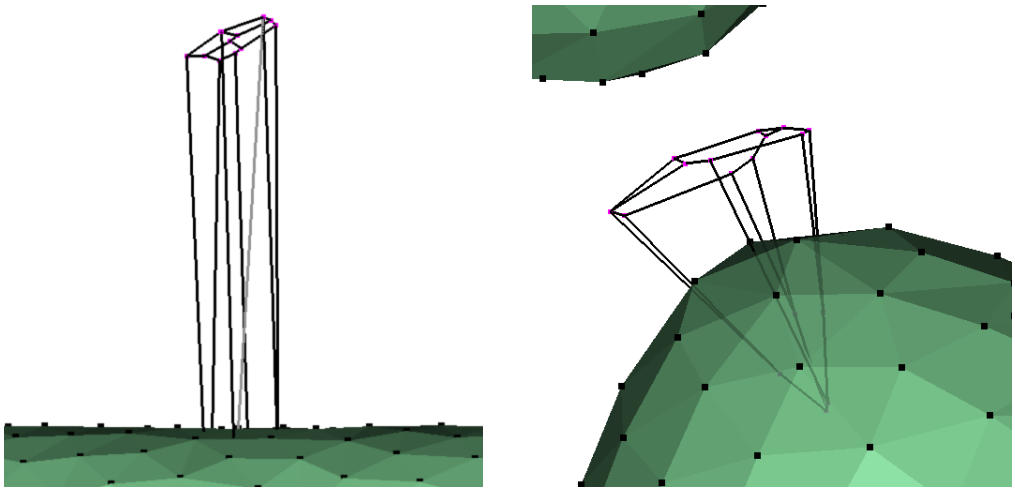
4.4. E^3 onepass version

Even if we limit the use of Voronoi vertices only to poles, the second Delaunay tetrahedronization consumes much time. 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 orthogonal 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 due to the algorithm robustness the obtained reconstruction is correct. Amenta introduced the algorithm which is based on this observation in [NAm99]. She defined for the surface triangles following three conditions. Let T be a set of surface triangles, then:

- the set T consist 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 then 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.



a) b)

Fig. 4.5: Examples of the Voronoi cells, 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).

After the tetrahedronization we can compute the triangle set T as follows (see an example in Fig. 4.6). 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. Than 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.

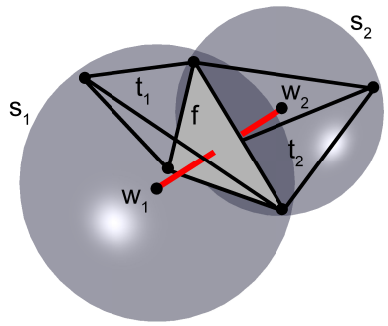


Fig. 4.6: The tetrahedron t_1 has a circumscribed sphere s_1 with the center w_1. The tetrahedron t_2 has the circumscribed sphere s_2 with the center w_2 and t_2 shares the face f (filled light gray) with the tetrahedron t_1. The edge from w_1 to w_2 (bold line) is the dual representation of the triangle f in the Voronoi diagram.

Whether the triangle f belongs to the set of surface triangles T depends on this criterion (Fig. 4.7): for the triangles on the surface, this edge has to pass through the surface S. Let us denote the angles $\alpha = \angle(w_1p, n)$ and $\beta = \angle(w_2p, n)$. When the interval $<\alpha, \beta>$ intersects the interval $<\pi/2 - \theta, \pi/2 + \theta>$ and this condition holds for each vertex p of the triangle f, then the triangle is on the surface and so 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.

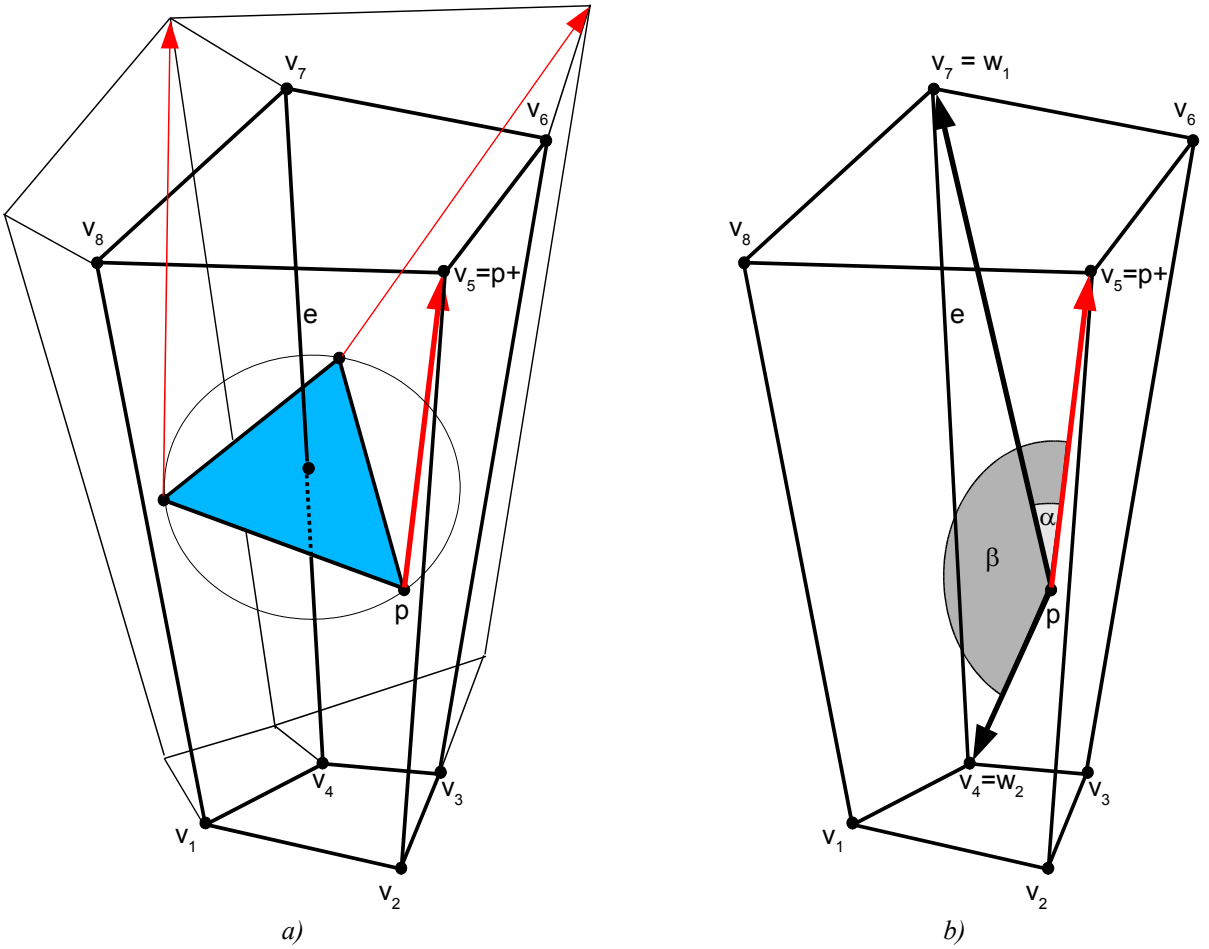


Fig. 4.7: a) The triangle (shaded) and three Voronoi cells incidenting with each triangle vertex. The edge e is the dual representation of the triangle in Voronoi diagram. The arrows represent the surface normal vectors in the triangle vertices estimated using poles. One Voronoi cell of the triangle 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. If the interval $<\alpha, \beta>$ intersects the interval $<\pi/2 - \theta, \pi/2 + \theta>$ 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 (chapter 7).

5. COCONE algorithm

This algorithm is very similar to the CRUST algorithm, the author Tamal K. Dey developed it with the help of Nina Amenta and their theoretical background is almost the same [NAm00]. 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.

Cocone C_p is a complement of a double cone (Fig. 5.1) clipped with the Voronoi cell centered at p with an opening angle $\pi/2 - \theta$ around the axis aligned with the normal vector estimated using the positive pole (θ is almost equal to $\pi/8$, the symbol < means the angle):

$$C_p(\theta)=\left\{ y\in V_p : <[(y-p),(p_+-p)]\geq \tfrac{\pi}{2}-\theta \right\}$$

Each cocone C_p has a set of its neighbors N_p:

$$N_p=\left\{ q\in P : C_p\cap V_q\neq \emptyset \right\}$$

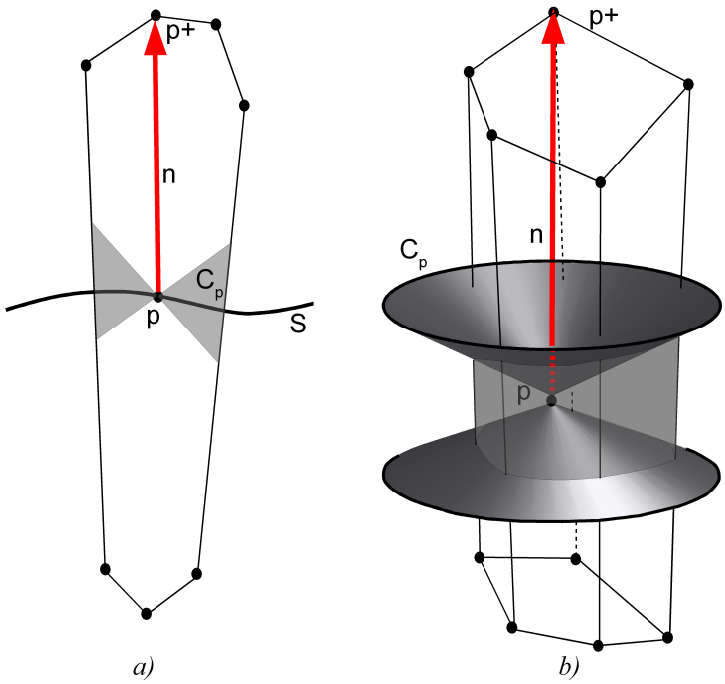


Fig. 5.1: a) A E^2 example and b) a E^3 example of the cocone C_p at the point p, where p+ is the positive pole (the farthest Voronoi vertex) and n the normal vector from the point p to pole p+. The surface S is highlighted only in E^2 example.

After the tetrahedronization we take all triangles one after another and test whether the dual Voronoi edge e of the triangle intersects three cocones C_p, where C_p are the triangle vertices. If it holds for all three triangle vertices, the triangle is put to the set of candidate triangles.

The candidate set of triangles are already close to a manifold for a sufficiently densely sampled surface but they do 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 description presented above is very close to the CRUST algorithm. What makes the COCONE different are later improvements. The author developed extensions which are able to detect boundaries, undersampling and oversampling and the most important extension which can reconstruct surfaces from large datasets.

5.1. boundaries, undersampling and oversampling

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, then the infinite dense sampling is necessary for sharp edges or corners. The boundary can be understood as a special case of undersampling, the scanning process was stopped in these places. 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. 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 presented surface properties can be detected by the different shape of the Voronoi cell. It holds for sufficiently sampled surface, that the cells are thin and long so when there some undersampled or oversampled region is presented, the shapes has to be different. Fig. 5.2 shows an example of the Voronoi diagrams in E^2, different shapes of Voronoi cells in the undersampled and oversampled regions are perceptible. The cells in undersampled regions are fat or very thin in oversampled regions in comparison to other Voronoi cells.

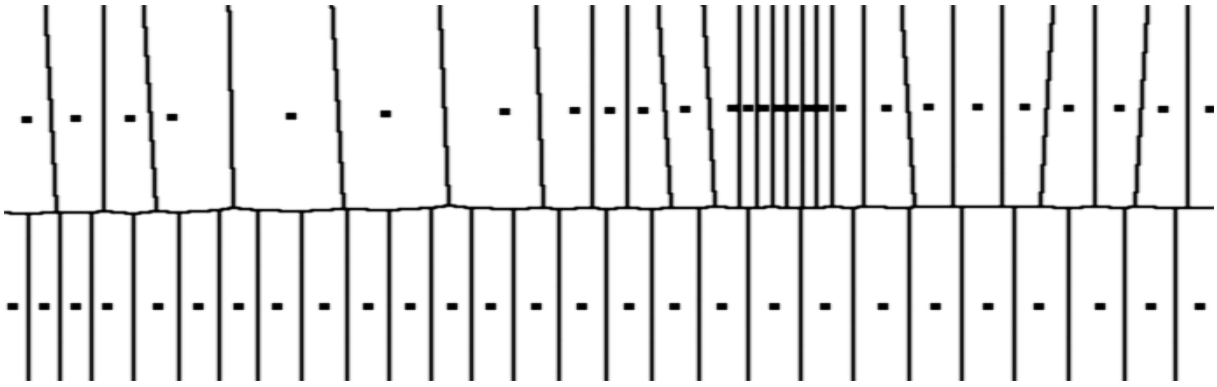


Fig. 5.2: An example of the Voronoi cells in E^2. The points at the bottom are sampled relatively uniformly, the points above are on same places undersampled and oversampled, the shapes of their Voronoi cells differ from other cells.

If we denote d_p (for each $p \in P$) as the maximum distance of any point from p that has p as its nearest neighbor on S, than the point p lies in the oversampled region if d_p is small compared to the local feature size *LFS(p)*. On the other hand, if this distance is too large compared to *LFS(p)* than the point p lies in the undersampled region. However, we cannot perform this comparison exactly due to the unknown surface S.

But we can use the approximation of d_p and *LFS(p)*, if we denote d_p as the distance from the point p to the farthest point in C_p and *LFS(p)* as the distance to the negative pole. In order to define the shape of the Voronoi cell, we define the *height* and *radius* of the Voronoi cell. The height is defined as the distance to the negative pole (thus *LFS (p)*, the positive pole is used for the approximation of the normal vector):

$$h_p = |p - -p|$$

The radius is then defined as the distance d_p:

$$r_p = d_p = max\{|y - p| : y \in C_p(\theta)\}$$

The assessment that some point lies in the flat part of a region, depends on two heuristic parameters ρ and α. If these two conditions hold:

- ratio condition: $r_p/h_p \leq \rho$
- normal condition: $\forall q, p \in N_q: \angle(V_p, V_q) \leq \alpha$

than the sample point p is called *flat*. The ratio condition captures that V_p is long and thin. But this simple condition is not enough to differentiate the interior samples from boundary ones since some V_p on the boundary can also be long and thin, so the normal condition has to be used, too. The parameters used to distinguish whether the point is flat are set to $\rho = 1.3\varepsilon$ and $\alpha = 0.14$ radians for an ε-sampled surface. An interior sample p is *deep* if it does not have any *boundary* sample as its cocone neighbor. The point p is *boundary*, if it is not flat.

The oversampling detection is little more complicated. It depends on the ratio condition presented above. As much points as possible should be interior, so the ratio condition of these points has not to exceed the value ρ. When we delete some point in the Voronoi diagram, which is interior, and the ratio of the updated Voronoi cell of the deleted point neighbors is still less than ρ, we deleted a point in the oversampled part of the surface. This step assumes to have the availability to delete points from the finished Delaunay tetrahedronization.

5.2. large data

The main difficulty of the COCONE and CRUST algorithms is that their use is limited to small or medium datasets due to the used algorithm of Delaunay tetrahedronization. E.g., our implementation of *DT* is able to process about 250K points on a system with 1GB of memory. Therefore the author of the COCONE algorithm developed an other improvement of the basic COCONE algorithm called SuperCOCONE [TKD01b] which is able to reconstruct large datasets using octree space subdivision and applying COCONE to each leaf.

First the root box is computed containing all sample points and in each subdivision step the box is split into eight subboxes. The end of subdivision depends on some parameter δ, which means the number of points in each box. If the number exceed this value in some box, than it must be splitted. Fig. 5.3 presents an example of octree subdivision of one reconstructed dataset.

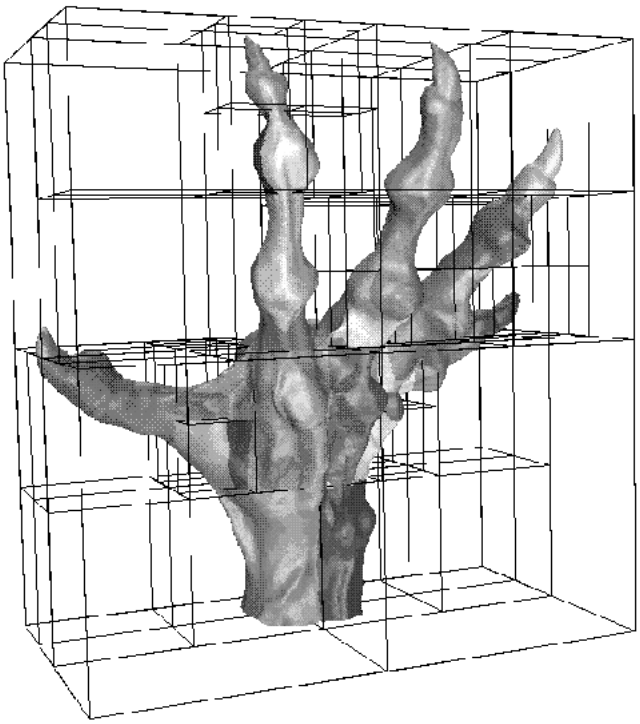


Fig. 5.3: An octree subdivision of some object. Each cell contains approximately the same number of points (courtesy of Dey).

We can proceed with the Voronoi diagram of all sample points in all leafs in octree but the problem appears on the side faces of the leaf box. Even the surface is closed, the subsurface in this leaf have incorrect boundary on the leaf sides and the reconstruction is incorrect.

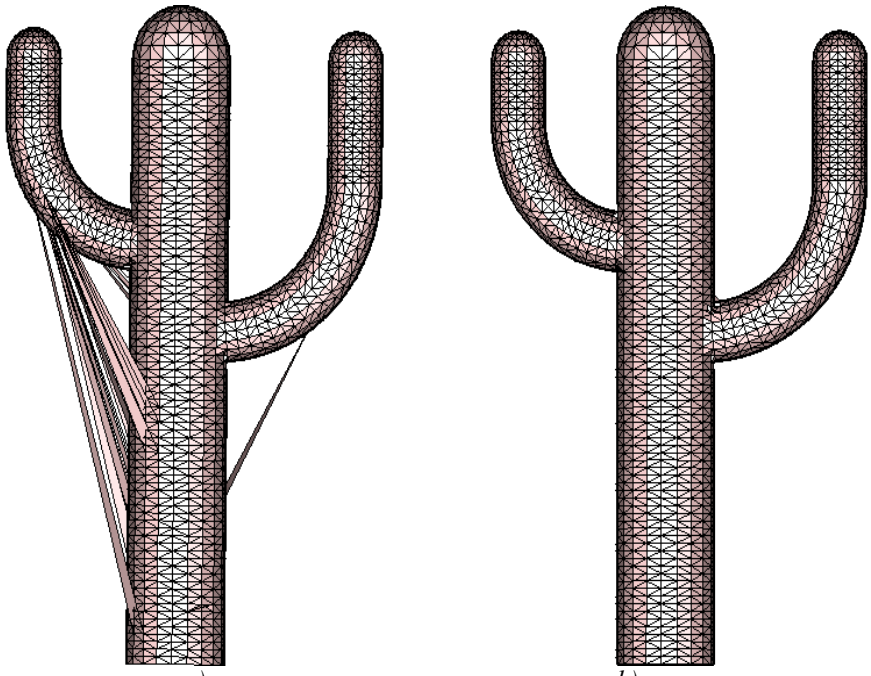
We can avoid this by taken all octree neighbors of this leaf B, but the size of the points set will be than large and only around one ninth of the computed DT is used for the local leaf reconstruction. Better way is to take a fraction from the adjacent boxes B'. Each adjacent box B' is subdivided up to a level l to produce boxes of size $1/2^l$ of the original size. Let X denote the set of all such boxes bordering B that are produced as a result of this subdivision, denote the extended box E_b as $B \cup X$. We take all sample points $P_b = P \cap E_b$. The experiments shows that l in the range 3 – 4 produces a good result.

After applying the COCONE to each extended leaf we get the set of candidate triangles and the manifold can be extracted as in the COCONE algorithm.

6. CRUST and COCONE problems

Both algorithms are based on the same principles and theories, so they have almost the same success of the reconstruction. The reconstruction of many datasets were done and the tests confirm the theoretical conclusions that the algorithms work well for sufficiently sampled data of the closed smooth objects without the requirement to be uniformly sampled. This chapter is more practical than the previous ones and it presents the results and conclusions of our experiments.

The first problem occurs in the first step of the algorithm, the computation of the Delaunay tetrahedronization. The code we had is robust and fast but, unfortunately, it uses standard FPU arithmetic and is numerical unstable. This property is fatal, when the surface has smooth or flat parts, very flat tetrahedra appear and the computation of circumscribed spheres used for Voronoi vertices position calculations produces bad results. This leaded to a little surprising observation, the uniformly sampled data of smooth objects were sometimes worse reconstructed than the nonuniformly sampled datasets with noise, because the noise made the tetrahedra not so thin so the computation was more numerically stable. Fig. 6.1 presents two examples of the same point cloud reconstructions, the cactus at Fig. 6.1a) was reconstructed using the *DT* with the normal FPU arithmetic, second cactus at Fig. 6.1b) was reconstructed using the *DT* with numerically stable geometric predicates [JRS96]. The difference is clear, the first figure contains a lot of bad triangles marked as the surface due to bad positions of Voronoi vertices.



a) *b)*

Fig. 6.1: a) An example of the reconstruction without using the numerically stable Delaunay tetrahedronization, b) the reconstruction with numerically stable Delaunay tetrahedronization.

Above described problem is the implementation detail, although 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 makes problems in many algorithms. Other question is the noise, the data with three dimensional noise (influencing the point in all spatial directions) are a problem.

Described sampling criteria or surface features influence one another, we cannot simply say that some place of the object is undersampled or if 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 are nonuniformly sampled, than it looks like undersampled or oversampled.

Generally, the CRUST and COCONE algorithms works perfectly for uniformly sampled closed smooth objects. The Voronoi cells of such a point cloud fulfill the criteria given by the theory and the set of primary surface triangles is close to the manifold (only with some overlapped triangles of very flat tetrahedra). Fig. 6.2 shows the reconstructed object with the detail of the triangle mesh after reconstruction. Fig. 6.2c) is the ε-sampling of the surface, the more gray the surface is, the higher is the ε-sampling (in all figures), thus the ratio between the nearest neighbor and the distance to the medial axis is higher, but it does not exceed the value of 0.4 (theoretical guaranties still hold).

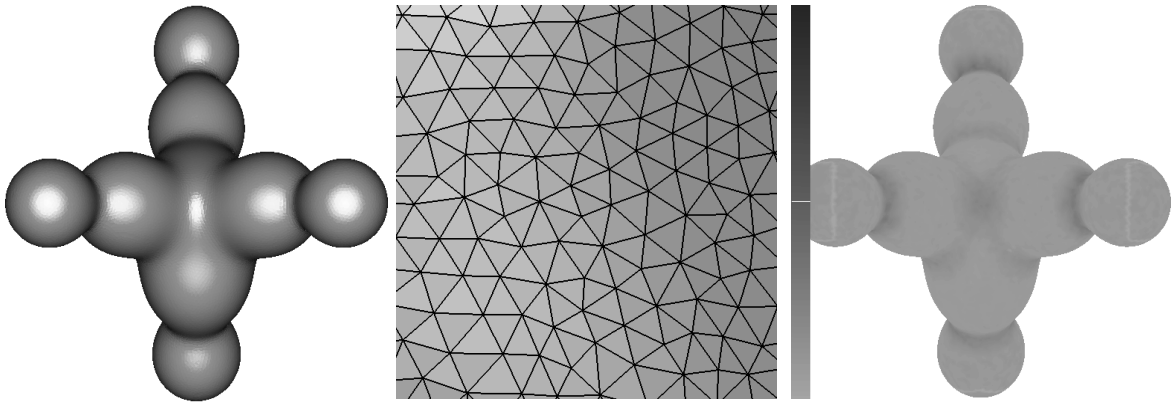


Fig. 6.2: a) An example of uniformly scanned object, b) the detail of the reconstructed surface with a triangle mesh, c) the ε-sampling of the surface.

Very problematic datasets are the ones, which are sampled in one direction more accurately than in the other, see an example in Fig. 6.3. It is a part of uniformly sampled object (cylinder), which is in Fig. 6.3a) sampled more precisely 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. 6.3b) presents the same object but sampled in both directions uniformly. We lost some details but the reconstruction is correct.

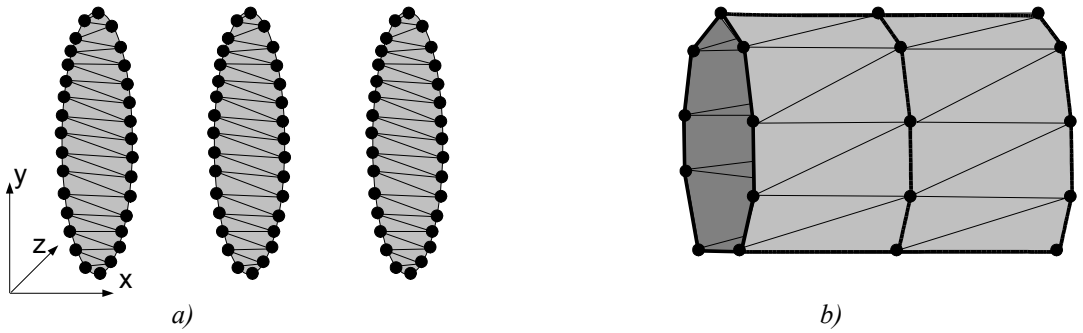


Fig. 6.3: a) An example of the object sampled in one direction more precisely than in others, b) uniformly sampled object (in both directions almost the same resolution).

Fig. 6.4a) shows real object, which is sampled more precisely in one direction in the part of neck and tail. The distribution of the point of this problematic parts is visible in Fig. 6.4b) and Fig. 6.4 d). The reconstruction fails in these parts because the Voronoi cells here have bad direction even though the shapes are thin and long (Fig. 6.4c). The estimated normal vectors direct parallel with the surface. Fig. 6.4e) 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.

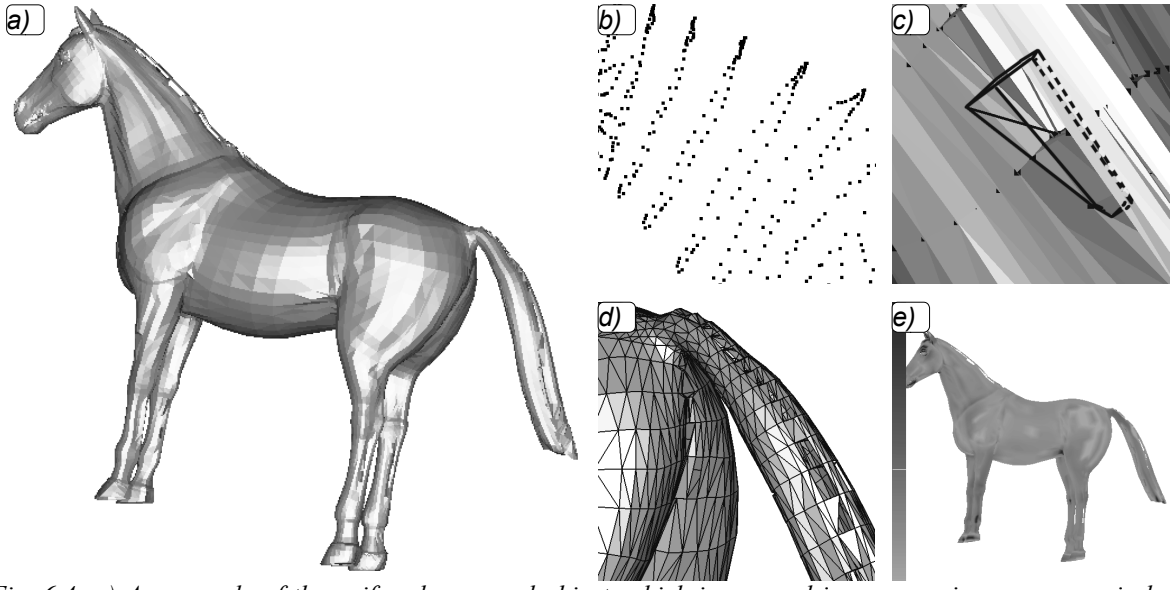


Fig. 6.4: a) An example of the uniformly scanned object, which is scanned in some regions more precisely in one direction, b) the distribution of points in the neck part, c) the Voronoi cell of some point in the neck part (the estimated normal is not orthogonal to the surface), d) the tail part of the object with the triangle mesh, e) the ε-sampling.

When the data are nonuniformly sampled, or uniformly but with some 2D noise, than the reconstruction success depends on the local points configuration, some reconstructions work perfectly and other not. Although the algorithms are not very sensitive to the changes in sampling density, the Voronoi cells can have bad shapes, the normal vector estimation is not very good on same places and holes or bad triangle configurations appear. An example of a nonuniformly sampled object is in Fig. 6.5a), Fig. 6.5b) shows the detail to the reconstruction. In Fig. 6.5c) the overlapped triangles arise from the local undersampling equally as in Fig.

6.5d) in the region of the ears. Fig. 6.5e) is the ε-sampling, the dark gray parts are the parts with bad ε-sampling and in these parts the reconstruction is not guaranteed.

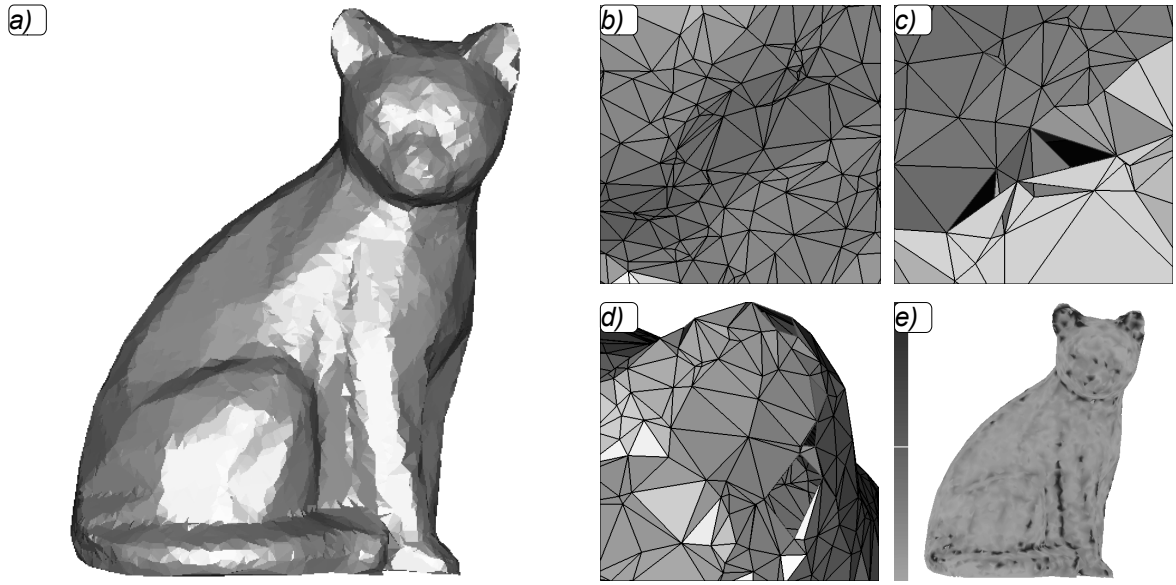


Fig. 6.5: a) An example of a nonuniformly scanned object, b) a detail of the reconstructed triangle mesh, c) the detail of the place with bad triangle configuration, d) the region with a sharp edge and with holes due to a local undersampling (cat's ears), e) the ε-sampling, red regions have worse ε-sampling.

The algorithms completely fail on the data which contain 3D noise. An example of this data is presented in Fig. 6.6a,b). It is a sampled terrain with a lot of noise, the reconstruction failed and the triangle mesh (Fig. 6.6d) is nearly unusable. In Fig. 6.6b), the reason is shown, the Voronoi cell shapes of many points doe not fulfill the algorithms requirements. Fig. 6.6e) shows the ε-sampling, well reconstructed parts of the surface have ε bellow 0.4 (bright parts in Fig. 6.6e).

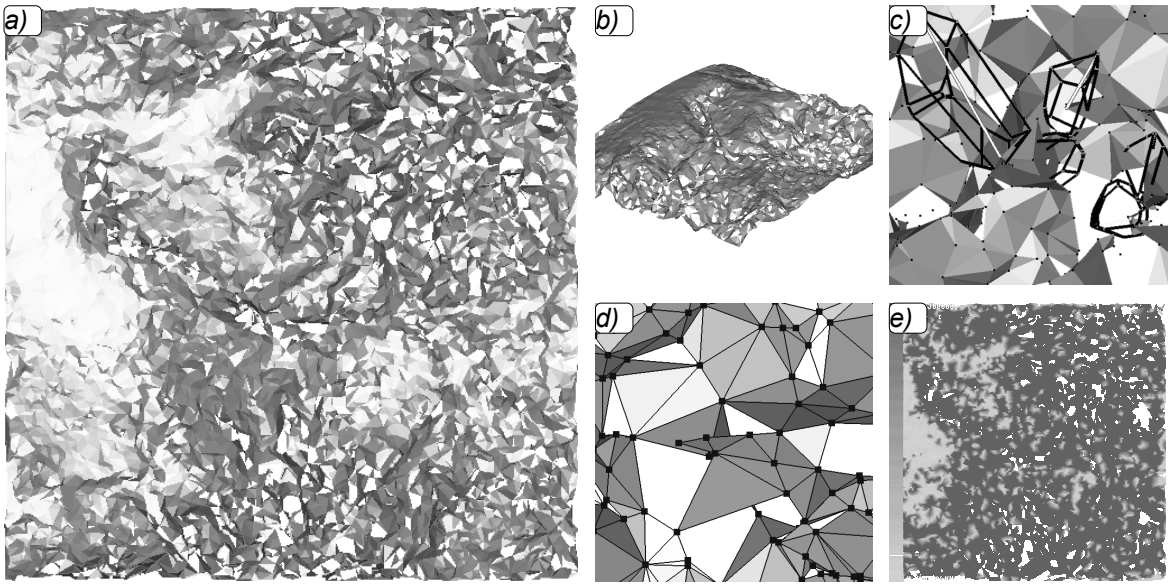
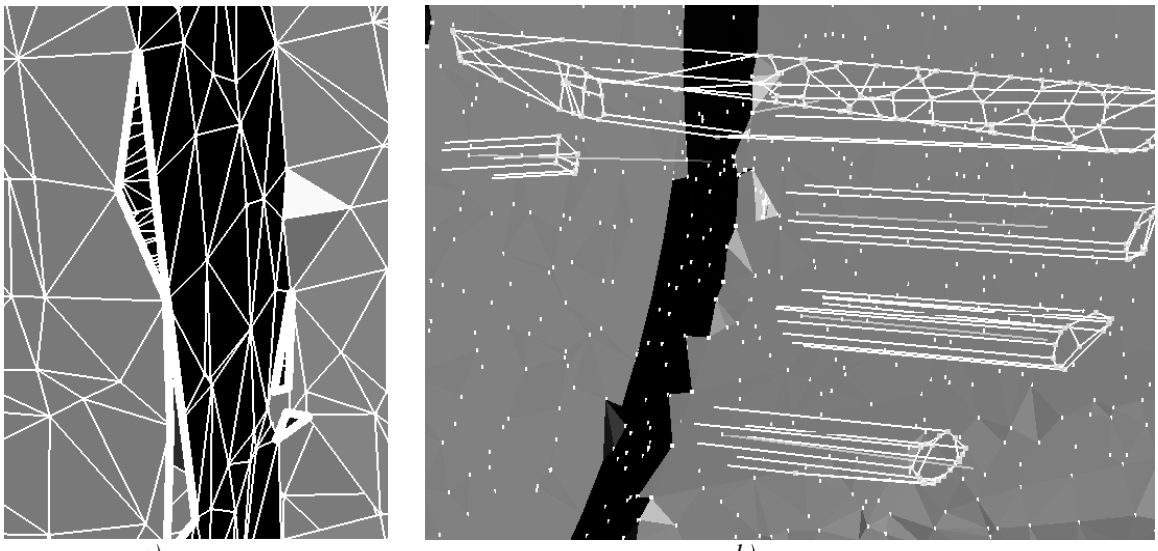


Fig. 6.6: a) An example of the surface with noise, b) another view of the surface, c) the Voronoi cells of some points in the surface, d) the detail of the reconstructed triangle mesh,e) the ε-sampling of the surface.

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 in the reconstructed surface some triangles arise with incorrect positions due to it.

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 are connecting the outliers with the other points making the reconstruction more difficult.

Edges are problematic, too, 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. 6.7 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. 6.7a) is the detail of the edge while Fig. 6.7b) presents the larger neighborhood 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.



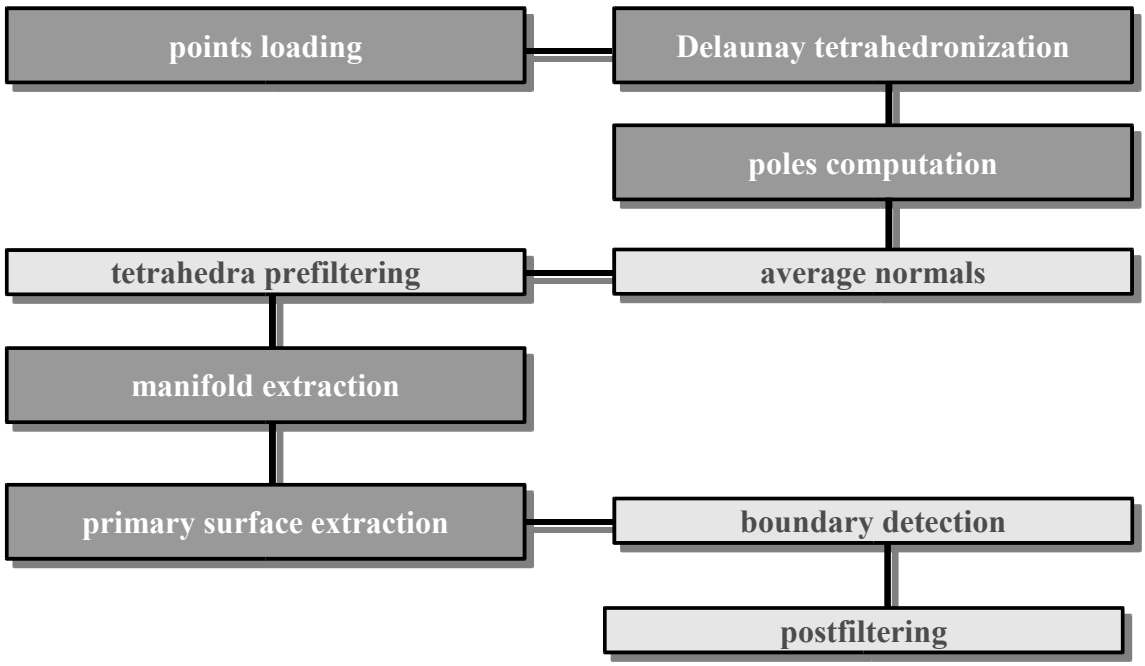
a) *b)*

Fig. 6.7: 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).

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 kind of data we have tried some improvement. They will be discussed in the next chapter.

7. implementation and improvements

The 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 onepass algorithm due to the amount of data limitation in the twopass algorithm although many presented improvements (mainly the steps of manifold extraction and the triangle mesh improvements) can be used in the twopass version too. The result of our twopass algorithm implementation can be found in [MVa02]. The work of the onepass algorithm can be divided into the following stages:



The stages printed in dark gray are main steps of the algorithm, the light gray stages are other improvements developed during the work on the algorithm. All steps will be described in the following paragraphs. The work on the algorithm is intensive, we can say that in current implementation only the idea of the poles computation and surface triangles marking is used from the original algorithm.

7.1. points loading

As mentioned in the first chapter, only E^3 positions of the vertices scanned from some object without any additional information are the input to the algorithm, so the structure of the input file can be very simple. We use a plain ASCII file, the first line of the file has to be the number of input points. This disposal helps in the phase of testing, e.g., when we want to reconstruct large data, we can sort the points according to some coordinate and reconstruct only the part of the surface.

Next lines of the file contain three numbers in floating point format separated by spaces or tabs. The end of the line can contain commas and the beginning the char “p”, we have found the files with this format on internet. Lines with other format are jumped over. An example of this file can be seen in Fig. 7.1.

```
11444
p 0 3.78683 0
d 0.159566 0.159566 0.159566
p 0.051427 3.75726 0
d 0.162481 0.162481 0.162481
p 0.099351 3.72645 0
d 0.165852 0.165852 0.165852
p 0.144067 3.69456 0
.....
```

Fig. 7.1: An example of the input file.

7.2. Delaunay tetrahedronization

For the tetrahedronization the code implemented by the supervisor of this thesis is used. The code is fast and robust (it uses the numerically stable geometric predicates library), it uses the method of incremental insertion for the creation of the tetrahedronization. The first phase of the algorithm is adding of 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 tetrahedron (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 all points are included to the tetrahedronization, the last step is then the deletion of the initial tetrahedron, its four points and tetrahedra incident to them.

The problematic part of the algorithm is the tetrahedron location, to which the inserted point belongs. If it is implemented by brutal force, all created tetrahedra has to be processed and the one correct 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 $O(N \log N) \div O(N)$ 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 230K points on 1GB of memory. More details about the stability were mentioned in chapter 6. The details about out implementation can be found in [IKo02, JKo03].

7.3. poles computation

After the Delaunay tetrahedronization computing we can obtain by dualization the Voronoi diagram. For each point p we take all the tetrahedra which are incident with this point p and compute the centers of their circumscribed spheres. These centers form the Voronoi vertices of the Voronoi diagram and they are used directly for the poles computation.

The vertex with the maximum distance 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 formed by the point *p* and the normal vector *n*. The schema of the poles computations is in Fig. 7.2.

```
for each point p
      T = ∅
      for each tetrahedron t incident with the point p
            T = T ∪ t
      end for

      max = 0
      for all t ∈ T
                  c = center of the circumscribed sphere of t
                  if max < |c - p|
                        p+ = p
                        max = |c - p|
                        n = c - p
                  end if
      end for

      max = 0
      for all t ∈ T
                  c = center of circumscribed sphere of t
                  if ((c – p) n < 0) AND (max < |c - p|)
                        p- = p
                        max = |c - p|
                  end if
      end for

end for
```

Fig. 7.2: The schema of the poles computation.

7.4. average normals

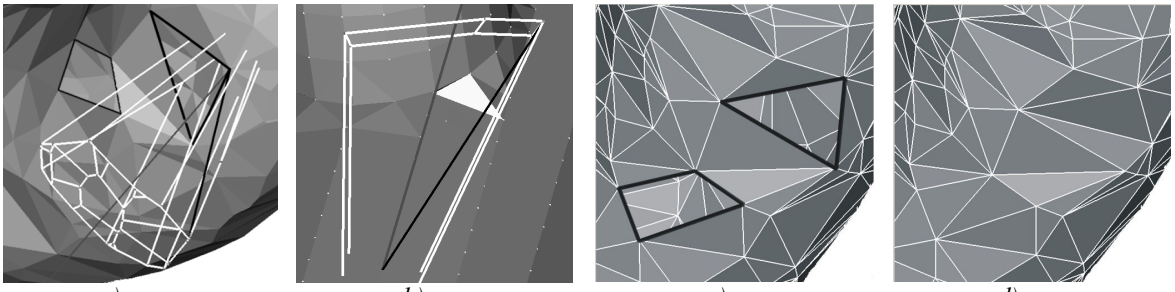
During the algorithm testing, we made a following observation. For sufficiently dense sampling the Voronoi cell is thin and long, the pole is nearly orthogonal to the surface. But in the cases, when the surface is not well sampled or has boundaries, the error between the estimated normal vector (vector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

So a simple improvement was made. Instead of computing only the farthest vertex (the positive pole *p*+) and taking 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. 7.3 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).

```
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 = the same half plane as p+
                  average = average + (v - p)
end for
```

Fig. 7.3: 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. 7.4 the examples are shown, Fig. 7.4a) and Fig. 7.4b) show a part of the reconstructed surface with the Voronoi cell and computed average normal. The direction of the average normal is more precise than the normal vector from point $p+$ to p. Fig. 7.4c) is a part of a reconstructed surface with the original normal vectors computation while Fig. 7.4d) presents the same part reconstructed using average normals where no holes appear.



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

Fig. 7.4: 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.

7.5. primary surface extraction

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 tetrahedronization the surface triangles using the test described in 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 the triangle twice. So all tetrahedra have a flag indicating whether it was processed or not.

We take one tetrahedron after another and for all four faces, triangles, we compare whether the opposite tetrahedron sharing the same face was processed. If it was, than we can continue with the other face otherwise the surface test for this face is computed. After the computation we set the tetrahedron flag as processed.

The surface test for a triangle f is computed as follows: because we know which two tetrahedra share the triangle, we can easily compute the dual Voronoi edge e of this triangle. It is the edge between the centers of the tetrahedra's circumscribed spheres (recall Fig. 4.6). Then for each triangle vertex p (an input sample point) the angles α, β between two vertices w_1, w_2 of the edge w and the normal n are computed. If the angle interval $<\alpha, \beta>$ intersects the

interval $<\pi/2 - \theta, \pi/2 + \theta>$ (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 Fig. 7.5 for details).

In the case that the triangle f is on the convex hull, no second 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 $\pi/2 + \theta$.

```
for each tetrahedron t
      for each triangle f of a tetrahedron t
                  top = tetrahedron neighboring to t over f
                  if (top <> NULL AND top.flag = TRUE)
                        continue
                  end if

                  correct = 0
                  w1 = center of circumscribed sphere of t

                  if top <> NULL
                        w2 = the center of the circumscribed sphere of top
                  end if

                  for each vertex p of triangle f
                        α = ∠(w1 - p, n)

                        if top <> NULL
                              β = ∠(w2 - p, n)
                              if (α, β) intersects (π/2 - θ,π/2 + θ)
                                    correct++
                              end if
                        else  if α < π/2 + θ
                                    correct++
                              end if
                        end if
                  end for

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

      end for

      t.flag = TRUE
end for
```

Fig. 7.5: The schema of the primary surface triangles computation.

7.6. manifold extraction

The triangles, which pass the conditions of Voronoi filtering (the conditions described in paragraph 7.6), 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 miss on the places of local discontinuity. For example, very flat tetrahedra in a smooth part of the surface (Fig. 7.6a) or tetrahedra on the surface edge (Fig. 7.6b) may have all faces marked as surface triangles. The number of overlapped 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.

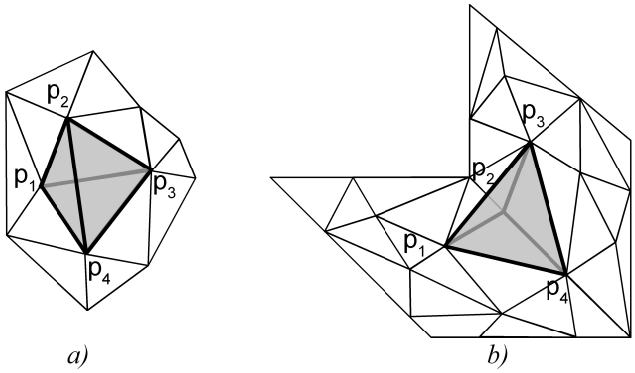


Fig. 7.6: Two examples where overlapping triangles may appear, a) flat part of the surface, b) a part with a sharp edge. One pair of triangles is $p_1p_2p_3$ and $p_1p_3p_4$, the second pair is $p_1p_2p_4$ and $p_2p_3p_4$.

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. The reason was that the manifold extraction methods were explained very briefly in the papers, however, this step is important. Our approach uses breadth-first search for appending triangles on free edges and it has has a linear time complexity. It was presented in [MVa03].

The preprocessing step of the extraction is creation of two structures which help in the phase of triangle neighbors searching. The first structure is the list of all triangles incident to each point, see an example in Fig. 7.7. The algorithmic complexity of the structure creation is $O(T)$, where T is a number of all triangles in the primary surface.

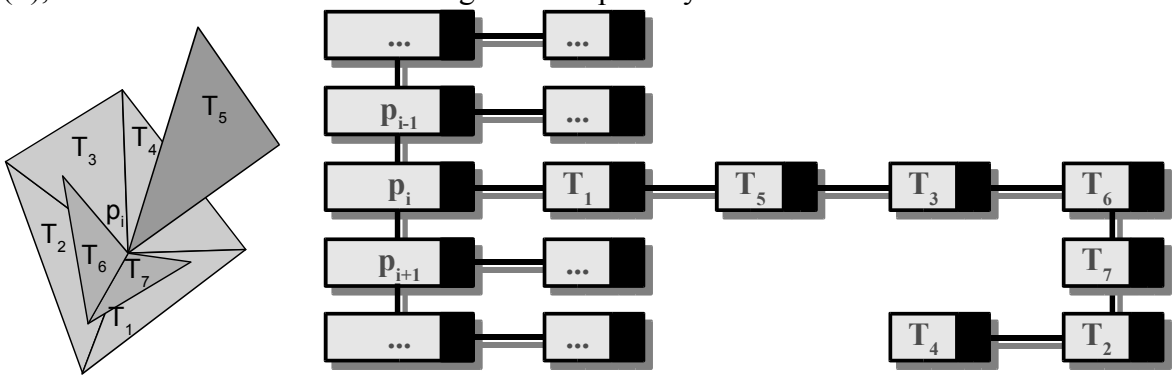


Fig. 7.7: An example of the data structure where each point p_i contains a list of its incident triangles T_i.

The second structure is created using the previously described structure and is called multiple neighbors list. As the name prompts, it is a list of triangles, where each triangle contains 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 incident, see example in Fig. 7.8. These two structures are used for fast triangle location.

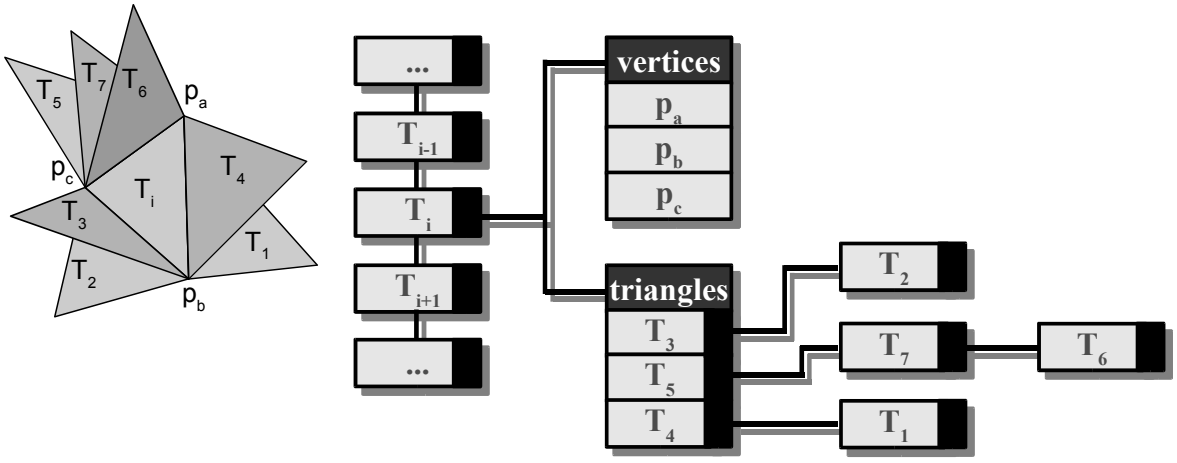


Fig. 7.8: An example of the structure which contains for each triangle the list of all possible triangle neighbors on its edges.

The direct 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 neighbors 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 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.

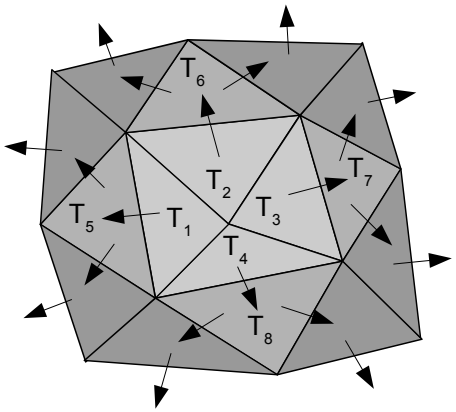


Fig. 7.9: The starting triangle configuration of the manifold is created by triangles T_1, T_2, T_3, T_4. They form the first level of the extraction tree. Next level of the tree is formed by triangles T_5, T_6, T_7 and T_8.

We can start now with the extraction. First we have to find starting triangles which will form root of the searching tree. We will find them using "umbrellas". We say that the point p has an "umbrella" if there exists a set of triangles incident to point which form a topological disc and no edge of the neighboring triangles is sharp. Other definition is that these triangles do not overlap when we project them to the plane defined by the point and normal (the estimated average normal). When we find the starting triangles around some point (Fig. 7.9), we add these triangles to the list which represents the first level of tree. Other levels of the extraction tree are created from previous ones by appending triangles on the non-processed

edges. Fig. 7.10 show an example of the tree created from the triangles in Fig. 7.9. The triangles T_1 - T_4 were used for the root of the tree and the triangles T_5 - T_8 were extracted in the second level of the tree. From this level other triangles are extracted and the first level is not farther needed, so we need for the tree to store only two levels of the tree, the current level and the newly created level. It is marked in Fig. 7.10 by the rectangle around the two bottom levels.

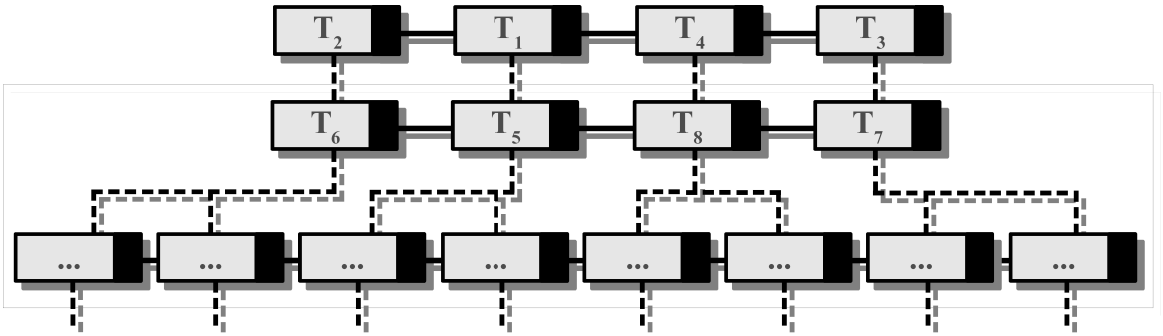


Fig. 7.10: First three levels of the tree created from Fig. 7.9. The rectangle presents which of the tree levels are actually necessary to store in memory.

When we create a next level of the tree, it is necessary to choose one triangle of the set of triangles on a currently non-processed edge. Fig. 7.11 shows the situation when some triangles (in dark gray) have been extracted and we have to extract at one edge of the triangle T_1 other triangle where the surface (manifold) continues. Three triangles T_2, T_3 and T_4 exist there (are marked as belonging to the surface) and we have to choose which of them is the correct one. For the recognition the direction of the triangle normal is important. The direction is inherited from the previously extracted triangle, the direction of the starting triangles in the tree root is estimated using the normal vector of the point around the starting triangles lie.

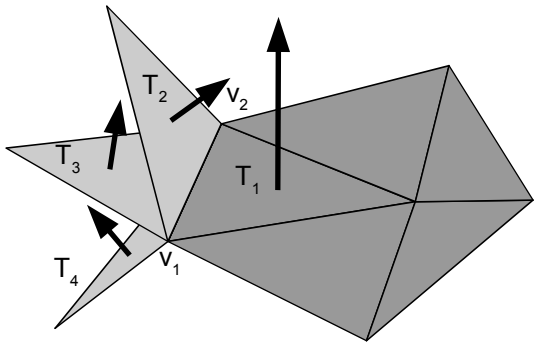


Fig. 7.11: The dark gray triangles are already extracted, the light gray triangles T_2, T_3, T_4 incident on one edge to the triangle T_1 and one of them has to be chosen.

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 [TKD01d]. We assume that the triangle T_1 is already accepted to the manifold and is correct. First we have to orientate the triangles T_2, T_3, T_4, see

Fig. 7.11. As we need to accept on the edge v_1v_2 the triangle nearest to the correct surface, 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. 7.12). Because we know tetrahedron neighbors, we can walk through these neighbors 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 neighbors and we choose the first marked triangle we find.

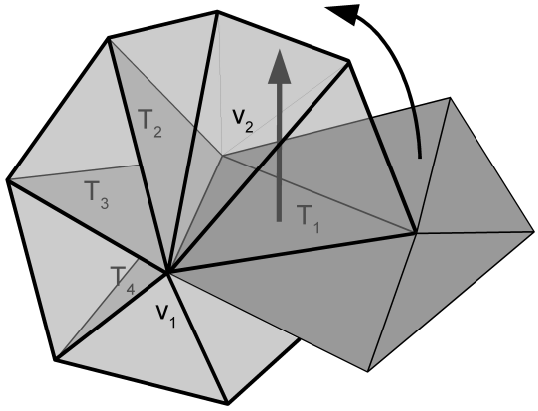


Fig. 7.12: The set of tetrahedra incident with the edge v_1v_2. The arrow shows the direction of walk.

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. Although it is a heuristic, we did not find any problem with it.

We continue recursively in adding new triangles to the edges of triangles in the current level tree, we form the next level tree and recursively continue. All points which were used in extracted triangles are marked with a flag. When we are not able to continue 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. 7.13.

```
for all points p
      set p.flag to FALSE
end for

for all points p
      if p has umbrella and p.flag = FALSE
            tree.current = umbrella triangles around p
            tree.next = NULL

            while tree.current <> NULL
                  for all triangles t ∈ tree.current
                        for all unprocessed edges e ∈ t
                              t_new = get correct triangle at e
                              if t_max <> NULL
                                    tree.next = tree.next ∪ t_max
                                    for all points s ∈ t_max
                                          s.flag = TRUE
                                    end for
                              end if
                        end for
                  end for

                  tree.current = tree.next
                  tree.next = ∅
            end while
      end if
end for
```

Fig. 7.13: The schema of the primary surface triangles computation.

The advantage of this manifold extraction method is that it has not big memory requirements and is very fast, however, we cannot compare with the time of the original algorithm due to different platforms. The example of extraction is in Fig. 7.14.

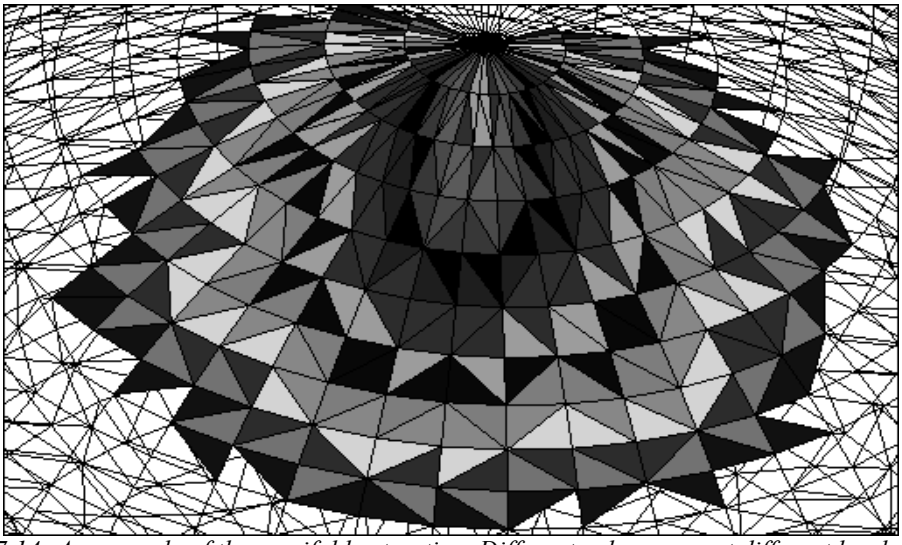


Fig. 7.14: An example of the manifold extraction. Different colors present different levels of the tree.

7.7. 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 are marked as surface triangles. See Fig. 7.15a) for an example: the dark gray triangles are already extracted and we are looking for triangle neighbor 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 the triangle hole occurs (Fig. 7.15b). Fig. 7.15c) shows a correct configuration.

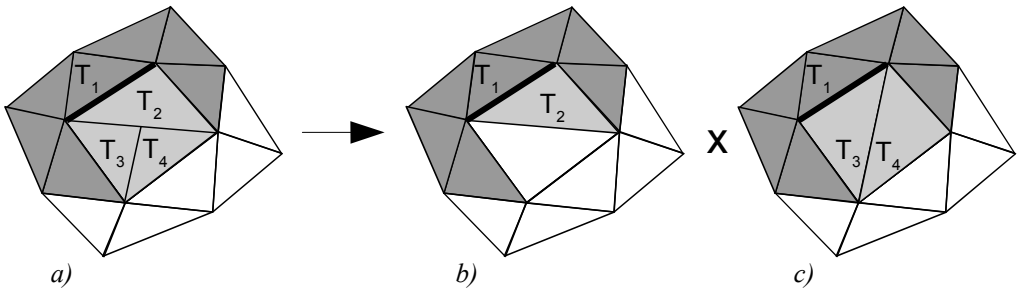


Fig. 7.15: Two configurations in the manifold extraction of the tetrahedron with three marked surface triangles, a) the initial status, on one edge of the triangle T_1 there are two connected triangles belonging to one tetrahedron, b) the wrong choice, c) the correct choice.

In order to avoid such situations it is necessary before the manifold extraction step to detect the tetrahedra, which have three marked faces, and remove one overlapped 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. 7.16). The best configuration is in Fig. 7.16d), the angle between triangle normals incident to the edge is the smallest (the dot product of the normals is close to one, in Fig. 7.16b) and Fig. 7.16c) is close to minus one).

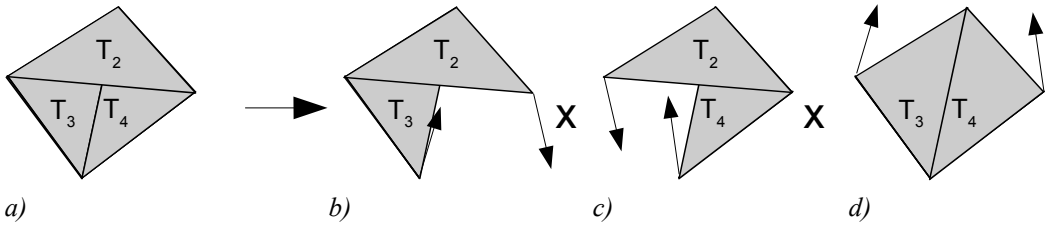
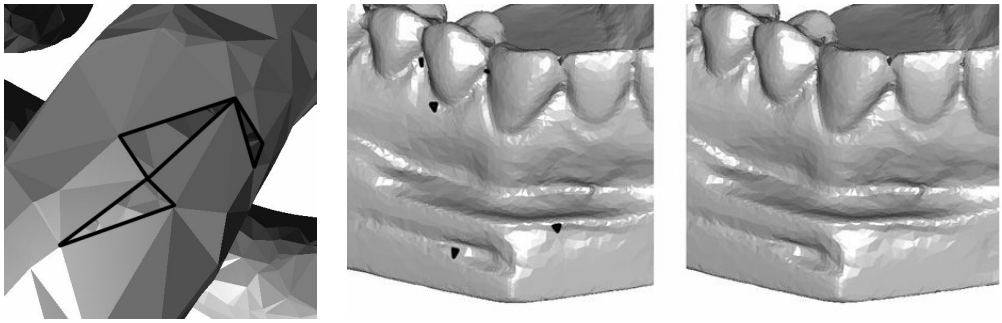


Fig. 7.16: a) The tetrahedron with three marked faces T_2, T_3 and T_4 and three possibilities b), c) and d) which two triangles to choose. Arrows correspond to the triangle normals.

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 good 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 and the triangle holes appeared there, too (Fig. 7.17a). After applying the prefiletring (Fig. 7.17b) and Fig. 7.17c), the situation changed and our algorithm was able to reconstruct the surface with much less triangles holes. Sometime a triangle hole still appears but the cause is different, the missing triangles were not chosen to be surface.



a) b) c)

Fig. 7.17: a) The detail to the surface reconstructed by Dey's COCONE, black are highlighted triangle holes in the surface, b) shows the reconstruction by our algorithm without the help of prefiltering (missing triangles are black) and c) with the help of prefiltering.

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. 7.1, the row “points” is the number of points in the tested dataset) and the number of redundant triangles measured, which it is necessary to remove from the triangulation. 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.

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

Tab. 7.1: 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").

7.8. postfiltering

When we have the data, which are 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. 7.18 shows an example. This detail is taken from a dataset which is not uniformly sampled and contains some noise. The highlighted part presents the erroneous place after the manifold extraction, missing and overlapping triangles.

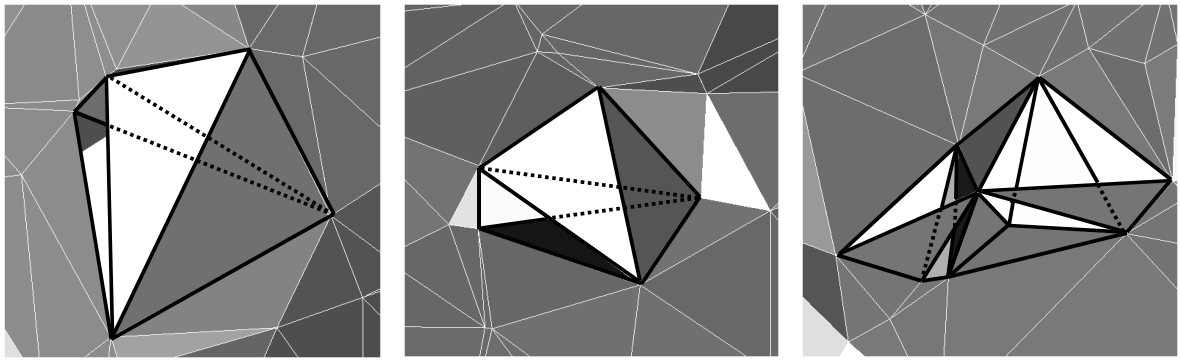


Fig. 7.18: A part of the reconstructed surface with an error after manifold extraction.

Missing and overlapping triangles appear there due to bad normal vectors arisen from the incorrect shape of Voronoi cells. We have analyzed 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 neighboring triangles (see Fig. 7.19).

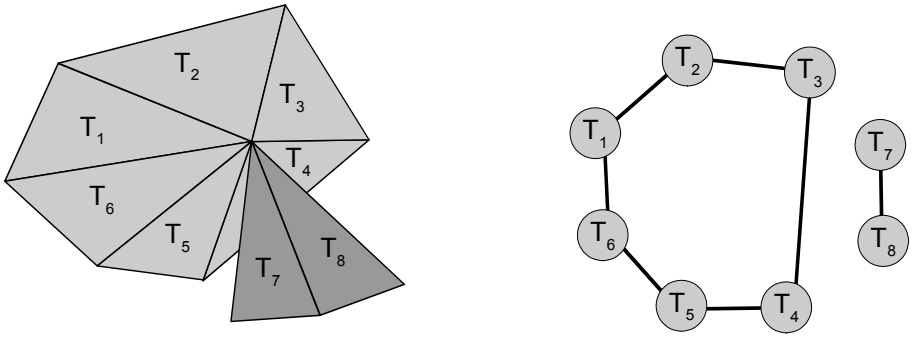


Fig. 7.19: An example of the fan configuration and a graph corresponding to the fan.

There exist two acceptable configurations of the triangle fan. Fig. 7.20a) presents a full fan around a point. It can be detected as the graph cycle which contains all nodes. Fig. 7.20b) 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.

Other configurations are incorrect and some triangles have to be deleted. When we are able to find one cycle in a graph, we can delete all triangles whose graph nodes are not included in the cycle. The most common configuration is shown in Fig. 7.21a), one full triangle fan with one separated triangle. The Fig. 7.21b) 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.

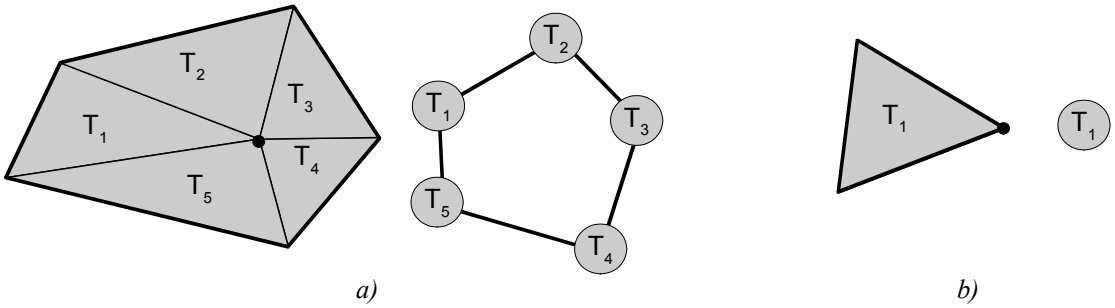


Fig. 7.20: a) Full fan and one graph cycle corresponding to the fan, b) one triangle with its graph.

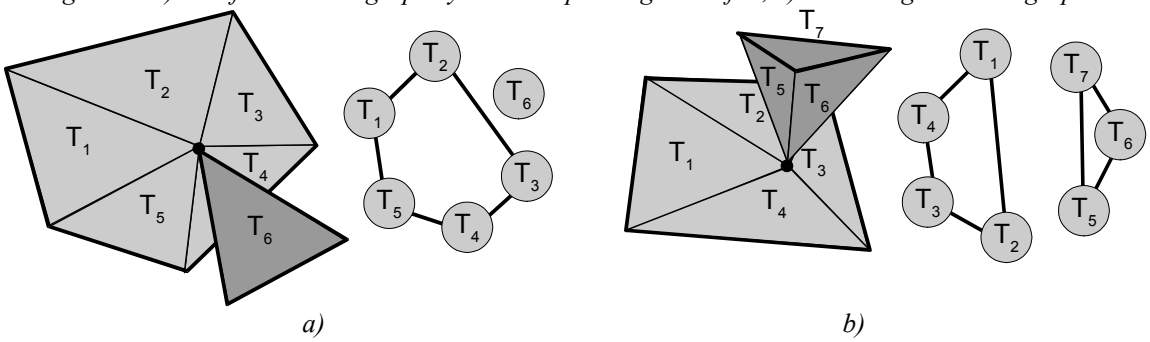


Fig. 7.21: a) One full fan with another separated triangle, b) more full fans.

The configurations presented in Fig. 7.22 are more problematic. When there are only subfans (we denote the fan as subfan if it does not form a cycle), the finding of the good fan configuration is not so simple and it will be explained in the following text. Here we cannot avoid the use of the normal vectors (we are testing these configurations in the projected plane), and it can bring problems. The normal vectors have good estimation only on the smooth parts of the surface, but the places, where these problematic configurations of the fans appear, are on the places where the sampling is not correct.

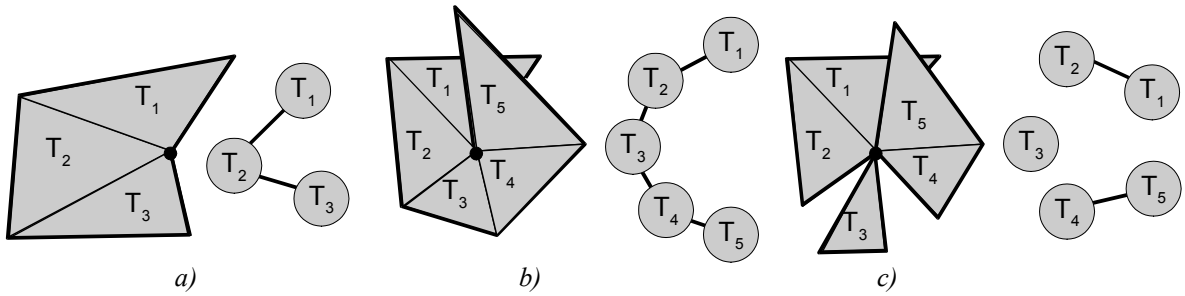


Fig. 7.22: Some fan configuration formed only by subfans, a) one subfan without overlapping triangles in a projection, b) one subfan with overlapping triangles in a projection, c) more subfans.

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 simpler for the configuration in Fig. 7.22a) and Fig. 7.22b) than Fig. 7.22c) 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 (Fig. 7.22a), the

configuration is accepted and no changes in the triangle mesh is done. When it is more (Fig. 7.22b), we delete triangles from one end of the subfan until the angle is less than 2π. We have implemented now only the removing from one end but better is to remove these triangles in more sophisticated way. Fig. 7.22c) 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. 7.18 three examples of bad triangle fan configuration were shown. Following Fig. 7.23 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 the triangles. Current implementation of manifold extraction is prepared for the probable next step of the extraction-the holes filling, but is limited now to fill only the triangle holes.

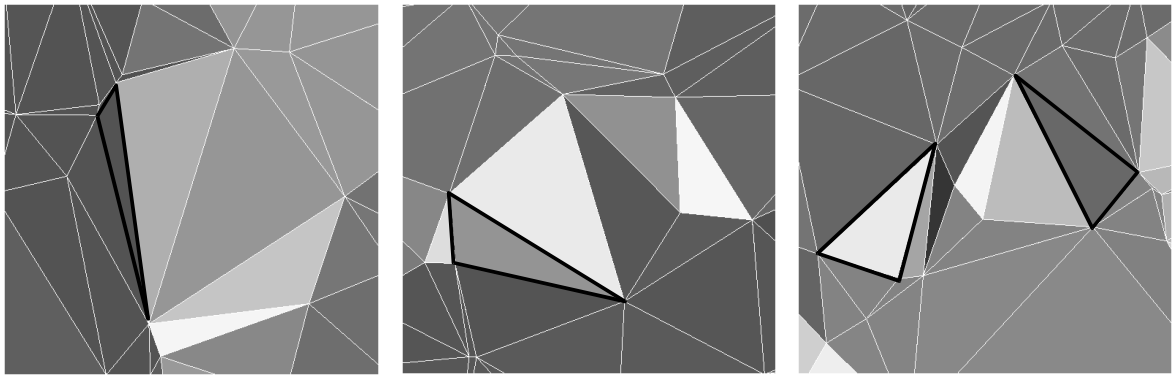


Fig. 7.23: The same parts of the surface as in Fig. 7.18 after applying prefiltering. The black triangles present the triangle holes formed after postfiltering which were filled with triangles.

The problem with overlapping triangles appear in the COCONE algorithm too, we found some bad fan configurations on the reconstructed surface (Fig. 7.24). In this case it was not possible to reproduce Fig. 7.24 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.

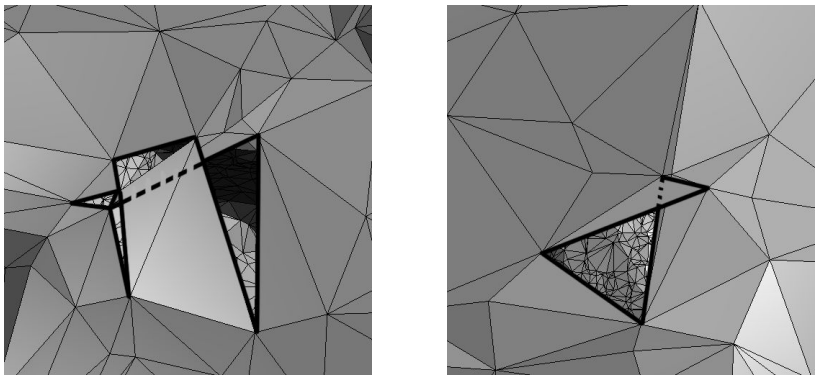
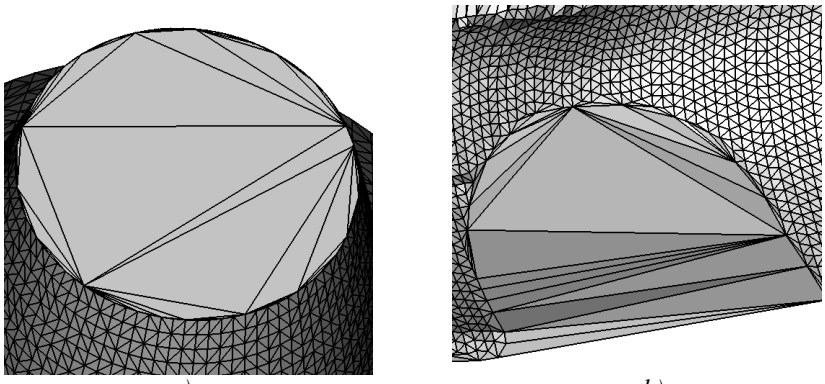


Fig. 7.24: The overlapping triangles in the surface reconstructed using COCONE algorithm.

7.9. 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 presents a boundary or just a local undersampling and we are left to heuristics. Dey presented a heuristic algorithm, which was mentioned in chapter 6, for a recognition if a point lies on a boundary or not.

We present now other heuristic approach based on the observation of the boundary triangles, see Fig. 7.25. There are two examples of a surface, where boundary triangles appear incorrectly. Fig. 7.25a) presents the case when the boundary triangles are perpendicular to the surface triangles while Fig. 7.25b) 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 point and incident triangle normals.



a) b)

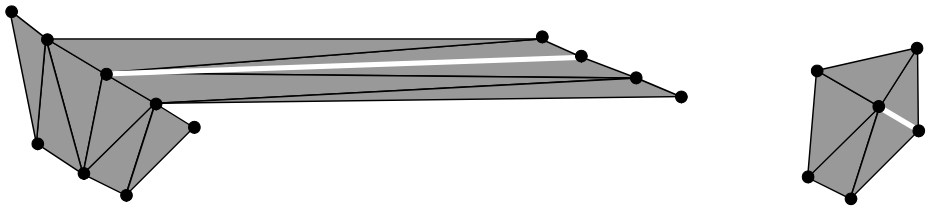
Fig. 7.25: Two examples of badly triangulated boundary, a) the boundary triangles perpendicular to the surface triangles, b) the boundary triangles parallel to the surface triangles.

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 :

$$m = c_{be} + c_{se} |n_p \cdot n_t|$$

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. 7.26a) and the boundary triangles were not filtered. Then we have tried to take the smallest edge in the set *E* (Fig. 7.26b). But some datasets have due to errors in the scanning process some points very close together, so more triangles were filtered than we wanted. Now 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. The schematic description of the algorithm is shown in Fig. 7.27.



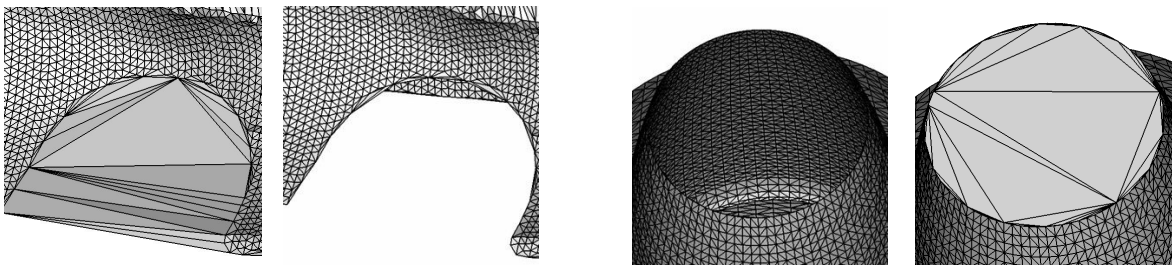
a) *b)*

Fig. 7.26: a) The referential edge is taken as the median of all edge lengths (the white edge), b) the referential edge is the shortest edge.

```
C_be = 2.0
C_se = 5.0

for each point p
      n_p = estimated point p normal

      E = all edges incident with p
      e = choose one referential edge from E
      l_e = |e|

      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 · n_t|

            if (m|v_1 - p| > l_e) AND (m|v_2 - p| > l_e)
                  delete t
            end if
      end for
end for
```

Fig. 7.27: The schema of the boundary filtering .

In Fig. 7.25 the reconstruction of some surface with a boundary was shown. After applying the boundary filtering, better reconstruction was obtained, see Fig. 7.28. Both cases, when the boundary triangles are perpendicular and parallel with the surface, were correctly reconstructed.

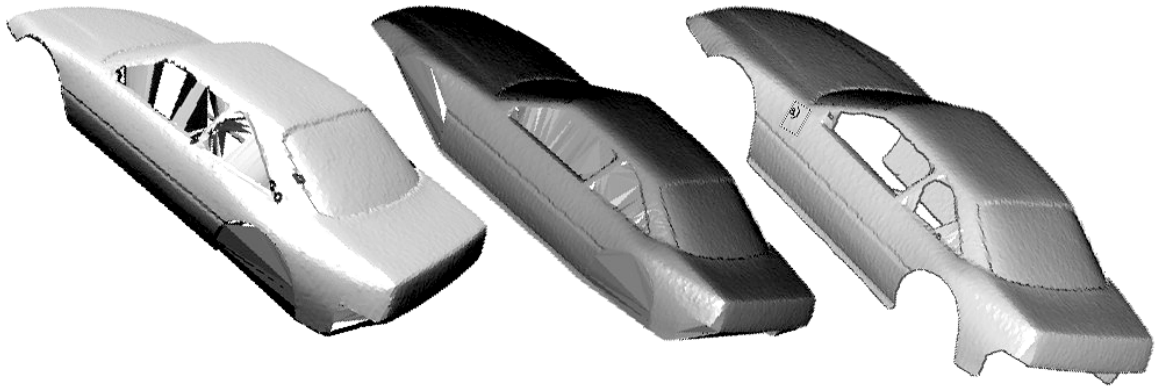


a) b)

Fig. 7.28: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. 7.28b) and the reconstructed models were almost the same. The reconstruction of the object as in Fig. 7.28a) fails and many bad boundary triangles appear there. Fig. 7.29 shows the whole reconstruction of the object from Fig. 7.28a). Fig. 7.29a) 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. 7.29b), 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. 7.29c). 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.

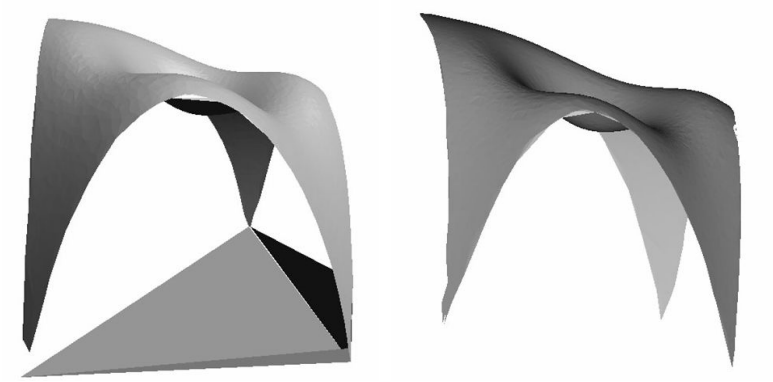


a) b) c)

Fig. 7.29: 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.

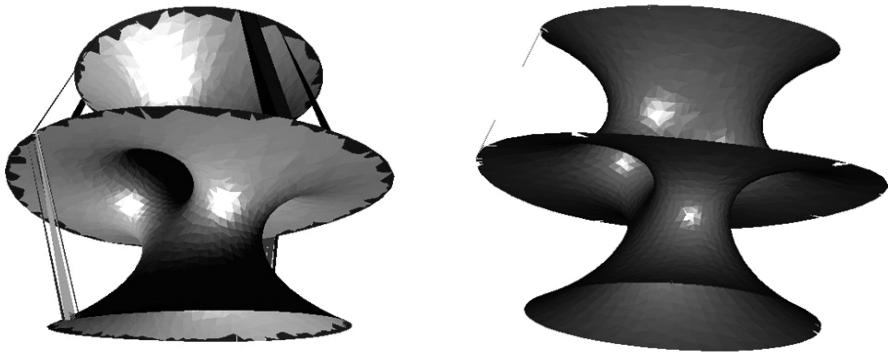
The car model is uniformly sampled, other test data, nonuniformly sampled, can be seen in Fig. 7.30, Fig. 7.31 and Fig. 7.32. The hypersheet model was a little problematic for both programs (Fig. 7.31). The data are not uniformly sampled, sometimes several points are mutually very close and some surface 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 were successfully reconstructed by both programs.



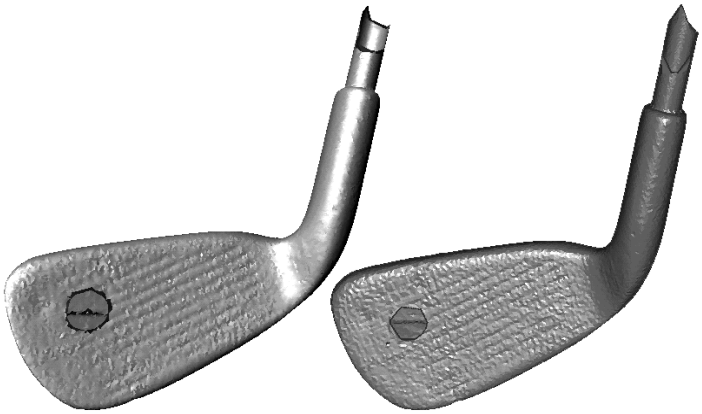
a) *b)*

Fig. 7.30: a) The function x^2y^2 reconstructed using COCONE, the surface is incorrectly connected with the boundary triangles, b) the reconstruction by out approach.



a) *b)*

Fig. 7.31: 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.



a) *b)*

Fig. 7.32: 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.

8. future work

As we have tested in several reconstruction tests, the CRUST algorithm works well for almost all data which pass the sampling criterion built on *LFS*. The only problem is with the data which are sampled more precisely in one direction because the sampling criterion does not count with this kind of data (recall Fig. 6.4). and the surface is not reconstructed correctly. This case was not mentioned in the Dey's or Amenta's articles.

When some regions of the data do not fulfill this criterion then there is no guaranty that they will be reconstructed correctly and the success depends on the local point configuration. Local configuration of points is critical, many ideas, which seem working fantasticly , were tested to improve the quality of algorithm but due the locality they failed. We have presented in this report some methods which really improve the quality of the reconstructed surface – the triangle mesh and there is still a lot of work on.

8.1. holes filling

First the holes filling has to be implemented. After the manifold extraction (and other manifold improvements such as boundary filtering or postfiltering) we have a set of triangles which satisfy the basic mesh condition – the number of triangles connected to one edge is less or equal to 2. When the edge is associated only with one triangle, it is a boundary edge and by recursive walking on neighboring edges with this property we can simply identify the hole. The care has to be taken on the points (edge vertices) where there are more than two associated hole edges (Fig. 8.1). In this case we still do not know what to do, because we want to have simple hole shape without duplicity points (one point on the hole is shared by more than two edges).

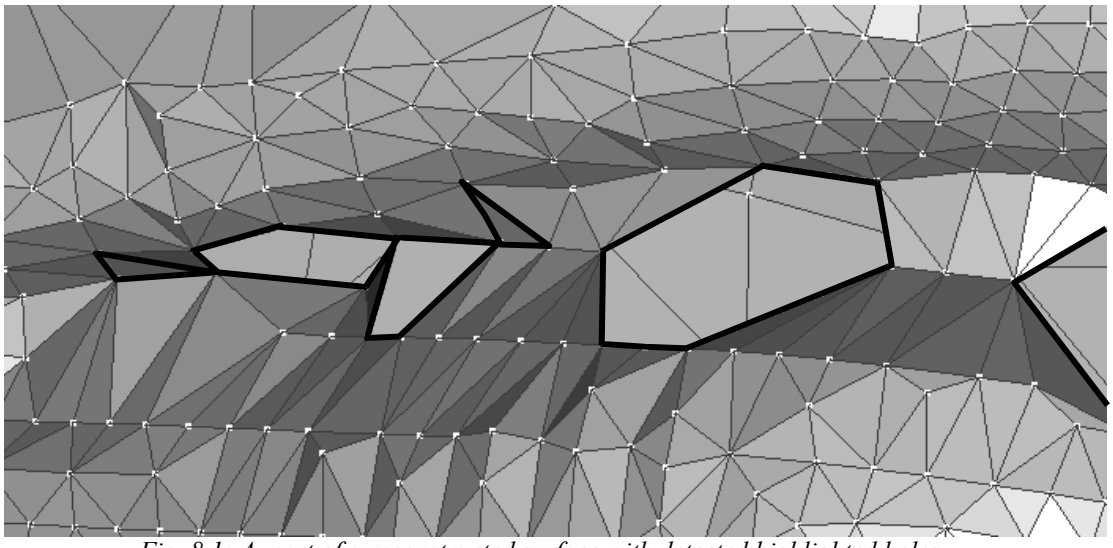


Fig. 8.1: A part of a reconstructed surface with detected highlighted holes.

For the hole filling we can use some existing algorithm for triangulation, the cooperation coming into the consideration is, e.g., with Alexander Jeměljanov, whose algorithm for holes filling is very robust [AJe03]. The other possibility is to develop some new (which uses existing data structures) because the robustness is not the crucial this time because the holes are small. The first idea was to develop a recursive hole triangulation, where two incident hole edges e_1, e_2 with the smallest angle will be connected by an edge (with the third edge e_{12}). Then these two edges will be replaced by the edge e_{12}, the number of hole edges decreases by one and we can recursively continue. Because we do not know if the hole is a boundary or just an undersampled place, some heuristic has to be used for recognition whether to retriangulate or not.

8.2. normal vectors comparison

Interesting idea is comparison of the normal vectors estimation. The normal estimation belongs to the critical part of many surface reconstruction algorithm so some comparison could be helpful. For a such comparison we should take data which have correctly computed normal vectors, maybe the method of implicit surface triangulation, which has been developing by our colleagues [MCe04], can give some data with analytical computed normal. We would like to compare the analytic normal vectors with the Hoppe's approach (recall section 3.1), CRUST approach and our average normal method.

8.3. points smoothing

Some real data we have are far away from the requirements of all reconstruction algorithms. The example of this data is in Fig. 8.2. It is a front part of some building whose photo pictures were taken and by photogrametric method the E^3 data were acquired.

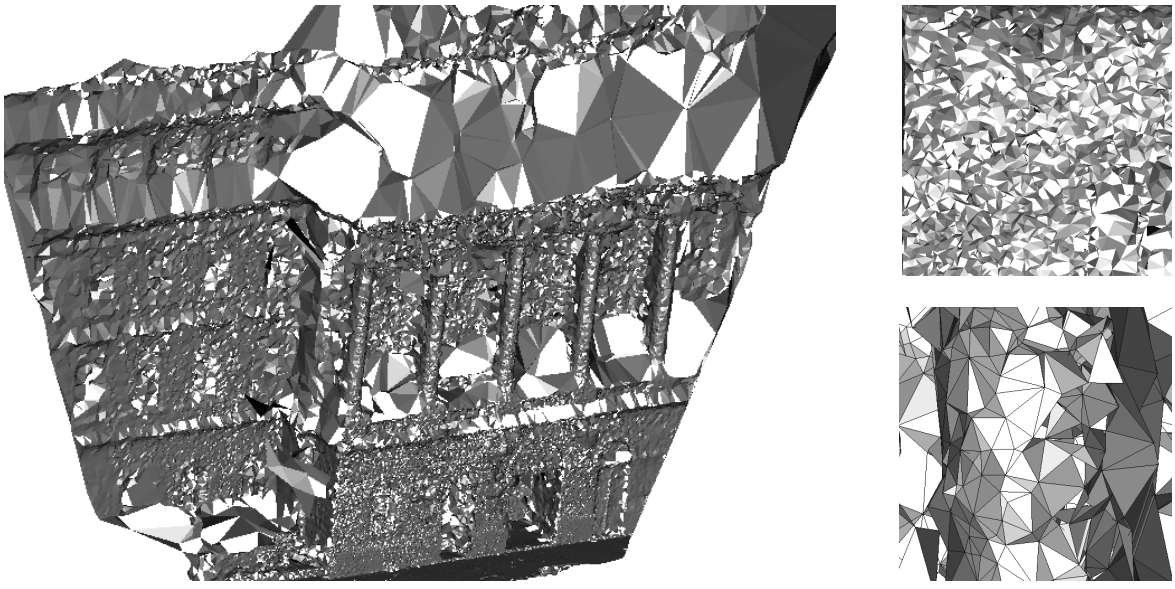


Fig. 8.2: The reconstruction of the front part of the real building, right are two details of the surface (data from TU Graz, Austria).

The result of reconstruction is awful, due to many reasons. The point cloud is very noisy (as seen in details in Fig. 8.2), many regions, which were not visible from the camera, are undersampled and many outliers are present. Unfortunately, it is not possible to use some mesh smoothing for the final triangle mesh, because it is too much erroneous. But it seems that is is able to use some points smoothing before the reconstruction steps for the noise reduction.

Now the smoothing is very rough and it is necessary to make a lot of tests and developments to improve it. The first results are shown in Fig. 8.3. The algorithm at present is very simple. The regular voxel grid is created for fast neighbor searching and for each point its several neighbors are used for a calculation of its average position (in Fig. 8.3 were used 20 neighbors for bunny smoothing). But the first tests of a noisy data smoothing tell that the smoothing depends on the surface position, the points has to be moved in the process of smoothing more to the presumptive direction of the surface than to others. Here we come back to the question of normal estimation, the estimated normals (by poles) are very bad in noisy places, maybe some other normal estimation technique will produce better directions.

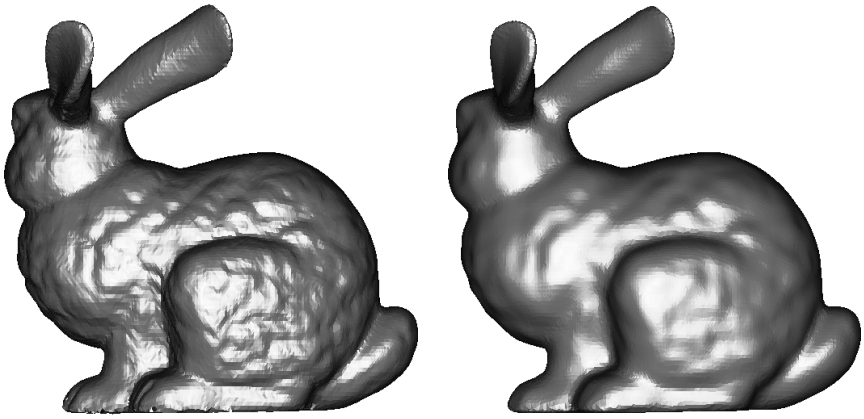


Fig. 8.3: The reconstructed Stanford bunny without and with the points smoothing (20 neighbors).

8.4. points decimation

The algorithms based on the global Delaunay tetrahedronization have a big disadvantage: they are not able to handle large number of points. Dey presented his SuperCOCONE algorithm (chapter 6) which divides the space to octree, reconstructs the lists of the tree and adds the obtained meshes together.

Our Delaunay implementation tetrahedronizes about 250K points on a system with 1GB memory, but some of our data are larger. So we have tried an idea: if there exists two points p_1 and p_2 which are the nearest neighbors each other, we will compute their average position p_{avg} and we replace the points p_1, p_2 with the new point p_{avg}. This approach we can use till we get the acquired number of points in the points set. Our observation shows that the geometry of the surface is not too much destroyed after this approach. Of course, in the future decimation it is necessary to develop some criteria such as, that the decimated edges length cannot exceed some value or that the edges lying on the surface corners has not to be decimated.

In Fig. 8.4 this recursive algorithm is tested on the dataset, which is nonuniformly sampled. The original dataset (in Fig. 8.4a) containing 10000 points was decimated to 5000 points (Fig. 8.4b), 2500 points (Fig. 8.4c), 1250 points (Fig. 8.4d), 600 points (Fig. 8.4e) and 300 points (Fig. 8.4f). It is shown that the curvature and the geometry of the object is not too much destroyed after the decimation, of course, when the decimation is too large than small features are missing.

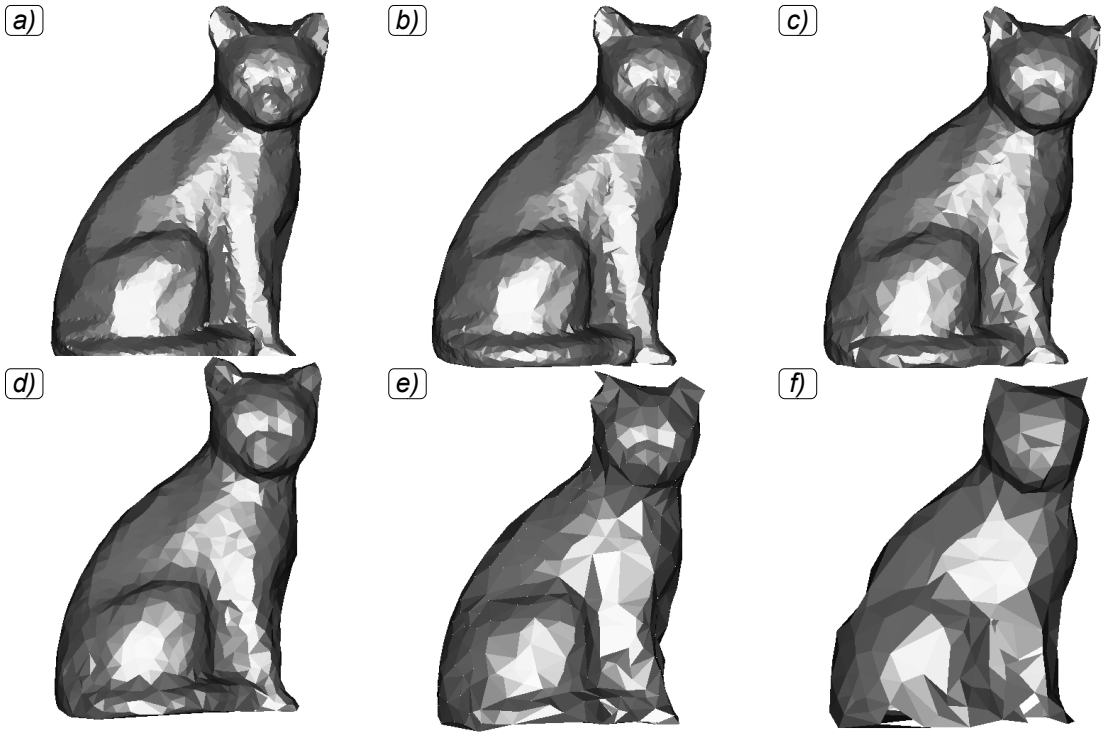


Fig. 8.4: The test of the points decimation before the reconstruction, a) the reconstruction of the original point cloud with 10000 points, b) 5000 points, c) 2500 points, d) 1250 points, e) 600 points, f) 300 points.

This decimation approach was also tested on big datasets that we were not able to reconstruct due to memory limitation. In Fig. 8.5 the results are presented, the datasets have from 400K points to 900K approximately. Results were surprising, all the data were decimated to 200k points and the reconstruction was almost perfect with the exception of some missing triangles due to the same problems as in small models. The features of the objects were also reconstructed correctly, even in small details we were not able to detect any differences between the models. For the future work it is necessary to do some measurement of the resulting mesh for this kind of decimation, because, e.g., many decimation mesh algorithms shrink the object and this problem may occur here, too.

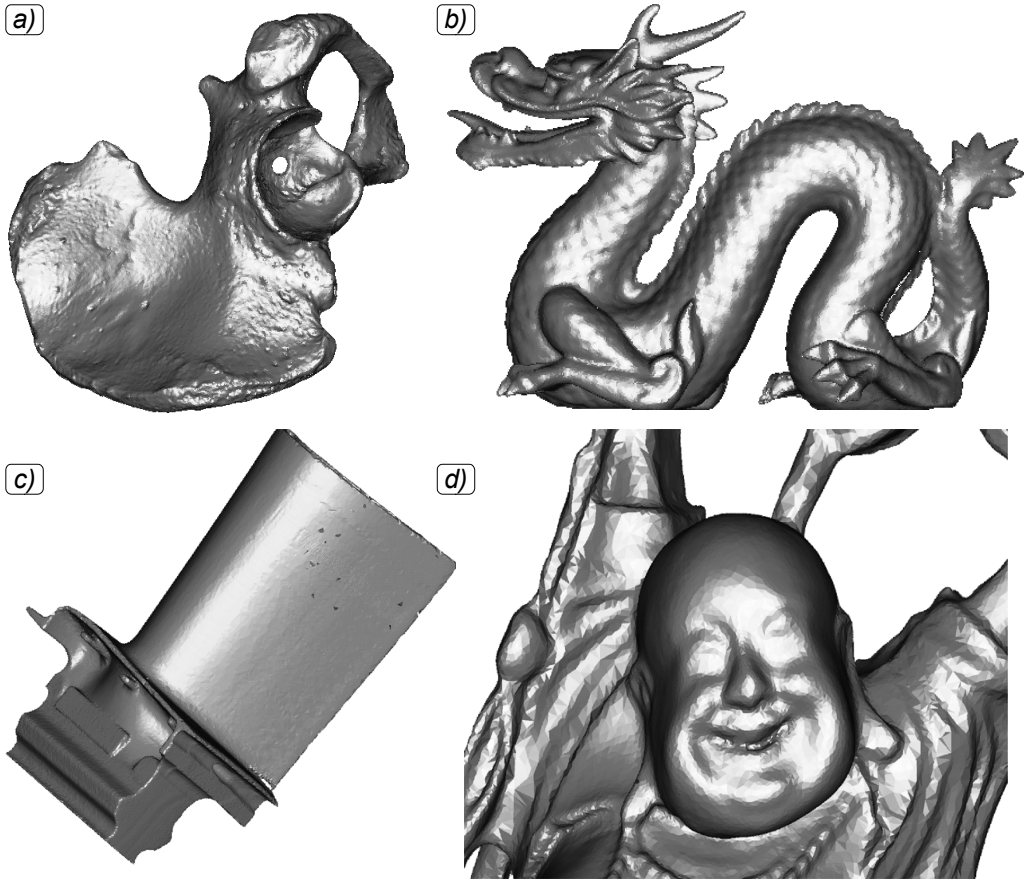


Fig. 8.5: The tests of the points decimation for large data, datasets were decimated to 200k points, a) the "hip" dataset with 530K points, b) the "dragon" dataset with 437K points, c) the "blade" dataset with 882K points, d) the detail of the "happy budha" dataset with 543K points.

8.5. parallel or distributed computation and large data

The next work on the algorithm assumes the connection with a work of our colleague Josef Kohout who is developing an efficient algorithm for parallelization and distribution of Delaunay tetrahedronization. This approach can of course speed the work of the algorithm but the main aim is to give us the possibility to process large data without any data decimation. We do not know yet how to connect these two works together, because the distributed E^3 Delaunay computation is still in the phase of development.

8.6. other improvements

We are also thinking about other improvements to improve the quality of the reconstruction process. The main difficulty is that the CRUST algorithm locally depends on the points configuration, small perturbations in data can drastically change the result, so the development of some improvement is not to easy, when it looks that something should fantastically improve some algorithm steps, than almost always some case is found where it does not work.

We have some ideas how to use the mathematical morphology, see Fig. 8.6. There is some part of the surface and its voxelization. The voxelization computation and points

assignment to each voxel is very fast and it can tell where the surface approximately lies. In the case of a hole, undersampling or when the voxels are too small and there appear holes between the voxels, we can try to use morphology operators such as dilatation or erosion.

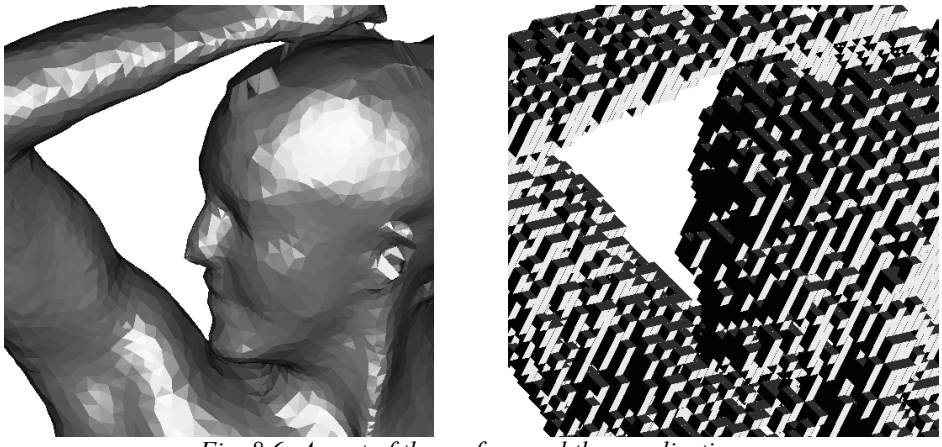


Fig. 8.6: A part of the surface and the voxelization.

Good idea seems to be also the tetrahedronization updating. When there is a hole in the surface we can try to add new points to the tetrahedronization in the plane of the expected surface with respect to the *LFS*, ore when there are too much points, we can try to remove some points from the tetrahedronization, similar approach uses Dey (section 6).

9. conclusions and acknowledgments

In this work the state of the art in the problem of surface reconstruction was presented, aimed mainly at the description of two algorithms, CRUST and COCONE, and some improvements were shown. In the last chapter some ideas to future work were described.

The author of this thesis wants to thank to the supervisor doc. dr. ing. Ivana Kolingerová for big support and patience, also to the head of the graphics group prof. ing. Václav Skala, CSc. for providing good conditions under which the work has been possible. This work was supported by the Ministry of Education of Czech Republic, project MSM 235200005, by the project AKTION 36p9 and by the project FRVŠ 1349/2004.

references

[ABa93] 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

[ABa94] A. Baader, G. Hirzinger. *A self organizing algorithm for multisensory surface reconstruction.* Intern. Conf. on Robot. and Intellig. systems IROS, 1994, pp. 81 - 88

[AJe03] A. Jeměljanov. *Surface reconstruction from problem point clouds*. Technical report, University of West Bohemia, Pilsen, 2003

[BDe34] B. Delaunay. *Sur la sphére vide*. Izvestia, Akademii Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk 7, 1934, pp. 793 - 800

[DAt97] D. Attali. *R-regular shape reconstruction from unorganized points.* Proc. of 13th ACM Sympos. of Discr. Algorithms, 1997, pp. 248 - 253

[DTe88] D. Terzopoulos, A. Witkin, M. Kass. *Constrains on deformable models, Recovering 3d shape and nongrid motion*. Art. Intelligence, 1988, pp. 91 - 123

[DTe91a] 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

[EBi95] E. Bittar, N. Tsingos, M. P. Gascuel. *Automatic reconstruction of unstructured data : Combining medial axis and implicit surfaces.* EUROGRAPHICS 1995, pp. 457 - 468

[EPM93] E. P. Mücke. *Shapes and implementations in three-dimensional geometry.* PhD. thesis, DCS University of Illinois at Urbana-Champaign, Urbana, Illinois, 1993

[FBe00] F. Bernardini, H. Rushmeier. *The 3d model acquisition pipeline.* State of the art report, EUROGRAPHICS 2000, pp. 41-62

[FBe97] F. Bernardini, C. Bajaj. *A triangulation based sampling and reconstruction manifolds using α-shapes.* 9th Canad. Conf. on Comput. Geometry, 1997, pp. 193 - 198

[GVo07] 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

[HEd92] H. Edelsbrunner. *Weighted alpha shapes.* Technical report UIUCDCS-R92-1760 DCS University of Illinois at Urbana-Champaign, Urbana, Illinois, 1992

[HEd94] H. Edelsbrunner, E. P. Mücke. *Three-dimensional alpha shapes.* ACM Trans. Graphics 13, 1994, pp. 43 - 72

[HHo92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle. *Surface reconstruction from unorganized points.* Computer Graphics 26 (2), 1992, pp. 71 - 78

[HHo93] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle. *Mesh optimization.* SIGGRAPH, 1993, pp. 16 - 26

[HHo94] 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

[JDB84] J. D. Boissonat. *Geometric structures for three-dimensional shape representation.* ACM Trans. Graphics 3, 1984, pp. 266 - 286

[IKo02] 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

[JKo03] J. Kohout, I. Kolingerová. *Parallel Delaunay triangulation in E3: make it simple*. The Visual Computer 2003, 19(7&8), pp. 532 - 548

[JRS96] J. R. Schewchuck. *Robust adaptive floating-point geometric predicates.* Proc. of 12th ACM, 1996, pp. 141 - 150

[JVM91] 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

[MCe04] M. Čermák, V. Skala. *Adaptive edge spinning algorithm for polygonization of implicit surfaces*. Accepted for Computer Graphics International, CGI 2004, Crete, Greece, 2004

[MEA96] M. E. Algorri, F. Schmitt. *Surface reconstruction from unstructured 3D data.* Computer Graphic Forum, 1996, pp. 47 - 60

[MVa02] M. Varnuška. *Rekontrukce povrchů geometrických objektů z roztroušených bodů*. Diplomová práce, University of West Bohemia, Pilsen, 2002

[MVa03] M. Varnuška, I. Kolingerová. *Improvements to surface reconstruction by CRUST algorithm*. SCCG 2003 Budmerice, Slovakia, pp. 101-109

[NAm00] N. Amenta, S. Choi, T. K. Dey, N. Leekha. *A simple algorithm for homeomorphic surface reconstruction.* 16th. Sympos. Comput. Geometry, 2000, pp. 125 - 141

[NAm98a] N. Amenta, M. Bern, D. Eppstein. *The CRUST and β-skeleton: combinatorical surface reconstruction.* Graph. Models and Image Processing, 1998, pp. 125 - 135

[NAm98b] N. Amenta, M. Bern, M. Kamvysselis. *A new Voronoi-based surface reconstruction algorithm*. SIGGRAPH, 1998, pp. 415 - 421

[NAm99] N. Amenta, M. Bern. *Surface reconstruction by Voronoi filtering.* Discr. and Comput. Geometry, 22 (4), 1999, pp. 481 - 504

[NAm01] N. Amenta, S. Choi, R. Kolluri. *The Power Crust.* Proc. of 6th ACM Sympos. on Solid Modeling, 2001, pp. 127 - 153

[RMe95] R. Mencl. *A graph based approach to surface reconstruction.* Comp. Graph. forum 14(3), EUROGRAPHICS 1995, pp. 445 - 456

[RMe98a] R. Mencl, H. Müller. *Interpolation and approximation of surfaces from three-dimensional scattered data points.* EUROGRAPHICS 1998, pp. 223 - 233

[RMe98b] R. Mencl, H. Müller. *Graph based surface reconstruction using structures in scattered point sets*. Proc. CGI, 1998, pp. 298 - 312

[RSz92] R. Szeliski, D. Tonnesen. *Surface modeling with oriented particle systems.*Comp. Graphics 26, 1992, pp. 185 - 194

[SMu91] S. Muraki. *Volumetric shape description of range data using "Blobby model".* Comp. Graphics, 1991, pp. 217 - 226

[TKD00] T. K. Dey, K. Mehlhorn, E. A. Ramos. *Curve reconstruction: connecting dots with good reason.* Comput. Geom. Theory Appliacation, 2000, pp. 222 - 244

[TKD01a] T. K. Dey, J. Giesen. *Detecting undersampling in surface reconstruction.* Proc. of 17th ACM Sympos. Comput. Geometry, 2001, pp. 257 - 263

[TKD01b] 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

[TKD01c] 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

[TKD01d] T. K. Dey, J. Giesen, W. Zhao. *Robustness issues in surface reconstruction.* Proc. Intl. Conf. Comput. Science, San Francisco, California, 2001

[TKD99] T. K. Dey, P. Kumar. *A simple provable algorithm for curve reconstruction.* Proc. ACM-SIAM Sympos. Discr. Algorithms, 1999, pp. 893 - 894

[TKD03] T. K. Dey, S. Goswami. *Tight Cocone: A water-tight surface reconstructor*. Proc. 8th ACM Sympos. Solid Modeling application (2003), pp. 127-134 [27]

[WEL87] W. E. Lorensen, H. E. Cline. *Marching Cubes: A high resolution 3d surface reconstruction algorithm.* Comp. Graphics 21 (4), 1987, pp. 163 - 169

publications

- M. Varnuška, I. Kolingerová. *Improvements to surface reconstruction by CRUST algorithm*. SCCG 2003 Budmerice, Slovakia, Comenius University Bratislava, pp. 101-109
  - this paper won the Springer 2nd Best Paper Award
  - proceedings also published under ACM (ISBN 1-58113-861-X)
- M. Varnuška, I. Kolingerová. *Manifold extraction in surface reconstruction*. ICCS 2004, Krakow, Poland, accepted as full paper
- M. Varnuška, I. Kolingerová. *Boundary filtering in surface reconstruction*. ICCSA 2004, Assissi, Italy, accepted as full paper

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, 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

stays abroad

- TU Graz, Austria, October – November 2003
- Aristotle University of Thessaloniki (Αριστοτελιο Πανεπιστιμιο Τηεσσαλονικι,) Greece, February – June 2001

University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitni 8
30614 Pilsen
Czech Republic

Surface Reconstruction Of Geometrical Objects From Scattered Point Data

The State Of The Art And Concept Of PhD. Thesis

Michal Varnuška

Technical Report No. DCSE/TR-2004-04
January 2004

Distribution Public

Technical Report No. DCSE/TR-2004-04
January 2004

Surface Reconstruction Of Geometrical Objects From Scattered Point Data

Michal Varnuška

Abstract

The surface reconstruction is a common problem in a modern computer graphics, there are many applications which need to work with a piecewise linear approximation of the 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. We use for the reconstruction 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 is to develop some improvements of this algorithm or a new algorithm which will be able to handle many kinds of data and properly reconstruct the surface.

The presented report contains the state of the art in the given computer graphics area, it aims to the description of two important algorithms CRUST and COCONE and it shows the common problems with the problematic datasets, published improvements and presumptive future work.

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

Copies of this report are available on
http://www.kiv.zcu.cz/publications
http://herakles.zcu.cz/publications.php
of 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