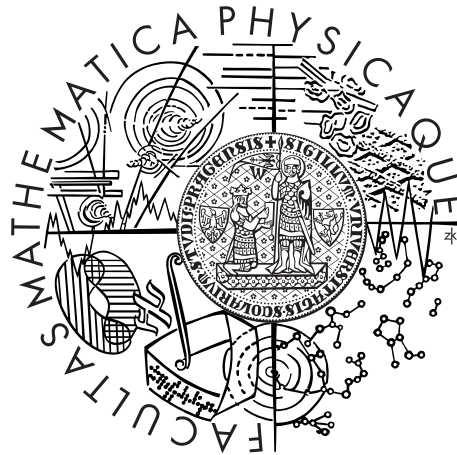


Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Peter Hmíra

Robust feature curve detection in 3D surface models

Department of Software and Computer Science Education

Supervisor of the master thesis: Mgr. Ján Dupej

Study programme: software systems

Specialization: computer graphics

Prague 2015

I would like to thank my supervisor Mgr. Jan Dupej and also RNDr. Josef Pelikán for their helpful guidance. I also dedicate this thesis to family who have supported me throughout my studies.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In date

signature of the author

Název práce: Detekce významných křivek na 3D povrchových modelech

Autor: Peter Hmíra

Katedra: Katedra software a výuky informatiky

Vedoucí diplomové práce: Mgr. Ján Dupej, KSVI, MFF UK

Abstrakt: Väčšina algoritmov zaoberajúcich sa detekciou črtov, nemá dostatočnú robustnosť voči šumu alebo nezvládajú vhodne spoje na povrchu v tvare písmena T. Prináša to nové výzvy navrhnúť taký algoritmus, ktorý nielen nájde črty na zašumených 3D-dátach získaných pomocou 3D-skenera. Dnešné algoritmy aj keď sú dostatočne robustné voči šumu, v okolí spojov v tvare T stratia dôležité informácie o črtoch, nakoľko trojicu čiar alebo kriviek vyhodnotia ako rovinu.

Klíčová slova: povrchový model, detekce křivek, robustní metody, diferenciální geometrie

Title: Robust feature curve detection in 3D surface models

Author: Peter Hmíra

Department: Název katedry či ústavu, kde byla práce oficiálně zadána

Supervisor: Mgr. Ján Dupej, KSVI, MFF UK

Abstract: Most current algorithms typically lack in robustness to noise or do not handle T-shaped curve joining properly. There is a challenge to not only detect features in the noisy 3D-data obtained from the digital scanners. Moreover, most of the algorithms even when they are robust to noise, they lose the feature information near the T-shaped junctions as the triplet of lines “confuses” the algorithm so it treats it as a plane.

Keywords: surface models, curve detection, robust methods, differential geometry

Contents

Introduction	3
1 Feature detection	4
1.1 Related work	5
1.1.1 Normal deviation based methods	5
1.1.2 Covariance analysis based methods	7
1.1.3 Projection procedure based methods	8
1.2 Extracting Feature Curves on Point Sets	9
1.2.1 MLS fitting	9
1.2.2 Curvatures computation	10
1.2.3 Potential feature points detection	12
1.2.4 Smoothing the feature points	14
2 Proposed algorithm	16
2.1 Limits of the previous algorithms	16
2.1.1 T-shaped connections	17
2.1.2 Curves interference	18
2.2 Improvements	19
2.2.1 Neighborhood optimization	19
2.3 The optimized algorithm	19
2.3.1 Principal curvatures calculation	20
2.3.2 Correlation	21
2.3.3 Growing feature curves	23
2.3.4 Feature lines post-processing	24
3 Implementation	27
3.1 Application structure	27
4 Results	29
Run-time tests	29
4.1 Synthetic data	29
4.1.1 Sphere	30
Comparison with the ground truth	31
4.2 Scanned data	36
4.2.1 Bunny	36
4.2.2 Armadillo	39
Run-time tests	40
4.2.3 Palate	41
Conclusion	42
Bibliography	43
List of Tables	45
Appendices	46

A User guide	47
A.1 Usage	47

Introduction

In pattern recognition and 3D-model processing, feature detection starts from an initial set of measured data and builds derived values - *features* intended to be informative, in some cases leading to better human interpretations. After the feature detection, the features are expected to contain the relevant information from the input data so that any task can be performed by using this reduced representation instead of the complete initial data.

Digital scanner technology has become more affordable and increasing its popularity and utility. These scanners collect a dense sampling of points, with mechanical probes or lasers, to generate a virtual representation of a physical form. Consequently, geometric processing of point clouds is becoming increasingly important. The preservation of sharp features is a primary concern for many geometric computations and modeling applications.

Probably the most simple method of feature detection is to implement a threshold test that identifies potential feature edges where normals differ above some tolerance level across adjacent samples. However, the method is notably sensitive to noisy data and the feature-preserving smoothing the normals is the nontrivial problem.

This master thesis is about feature curves extraction directly from a surface point cloud; no topological information or surface reconstruction is needed. In this thesis we analyze several existing algorithms that extract features from 3D-models, classify them into categories according to the approach to curves approximation.

We pick one algorithm claiming the robustness to noise and analyze its behavior in the edge situations that may occur when common inputs are used. We will make a various test over synthetic models and “real data” obtained from the laser scanner, as well.

In next part, we will propose a new algorithm that covers deficiencies found in the other algorithms and analyze the computational cost of the improvement. The analysis will focus on preserving the feature points in noisy models and the post-processing of the feature curves. The post-processing involves the filtering of the outlier features, connecting the features gaps and connecting the T-shaped junctions.

We will then describe the implementation of the proposed algorithm as well as the reference algorithm. We will discuss the motivation of choosing the data structures used in the program. We then analyze the run-time of the algorithms implemented and analyze their performance on the various 3D models. The models used for the testing purposes will test primarily the robustness to noise so we test the algorithm on each model multiple times with different levels of noise. We will then compare the outcomes of the algorithms, in terms of correctness and losing the information because of noisy data.

1. Feature detection

Feature detection is known to be an indispensable tool in mesh processing. Features such as valleys and ridges have to be classified in a stable way to provide the base for the algorithms such as feature-preserving mesh-denoising algorithms, face recognition, etc. The feature detection turns out to be an important ingredient to surface processing applications.

Indeed, some applications are:

- *mesh decimation*: given initial, noisy surface is smoothed, while edges on it are simultaneously preserved or even enhanced
- *surface matching*: surfaces are reduced to a skeleton feature lines to enable a better comparison item *reverse engineering*:

Feature line extraction consists in finding perceptually salient lines over 3D meshes that a human eye will notice. Detection of feature lines on polygonal surfaces has been an area of intensive research

Similar to edge detection in images, feature line extraction on 3D surface meshes is an ill posed problem due to the lack of information on the sampling process, the noise process and the signal geometry. According to the situation at hand, the same geometrical discontinuity might be due to a “true” feature in the object, to insufficient sampling in a region of high curvature, or to noise in vertex positions. Additional difficulties in feature line extraction process are its sensitivity to irregular mesh connectivity when extracted feature lines are composed of mesh edges, the use of a parametric model (e.g snakes) when feature line segments are independent of the mesh connectivity ([1] and [2]), and the selection of significant feature lines at the right scale.

Challenges

3D-models processed by the feature-detection algorithms are not always smooth with obvious features; especially when we have real data obtained from the 3D laser scanners.

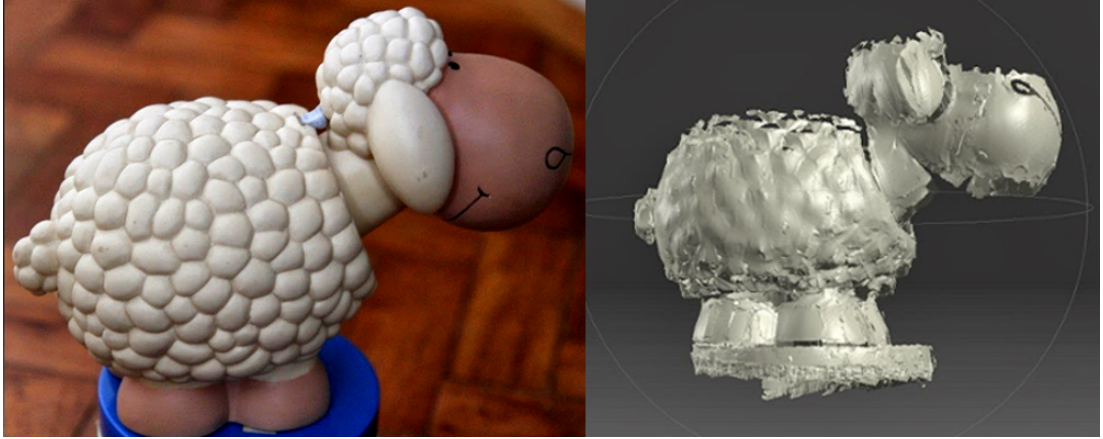


Figure 1.1: On the left, there is an original object photographed. On the right, we can see the 3D-scan of the object rendered in a 3D viewer. Source of the image:[3]

In figure 1.1 we can see the 3D model of the object obtained with a 3D laser scanner. In most of feature-extraction algorithms the noise causes the loss of information which makes such an algorithms unusable for a considerable amount of 3D-scans. As a matter of the fact, laser scanners output data with the remarkable noise, in this thesis we will analyze the feature extraction algorithms that focuses on the preservation of the feature information in noisy 3D-models.

1.1 Related work

The previous work on feature curves extraction can be roughly divided into three categories:

1. normal deviation based
2. covariance analysis based
3. projection procedure based

The algorithm *Extracting Feature Curves on Point Sets* falls into the third category of feature extraction algorithms since it is based on a projection procedure for local fitting.

1.1.1 Normal deviation based methods

The normal deviation based methods estimate the normal vectors and then performs the segmentation over the points cloud based on the variation of the normals. The point cloud is segmented into clusters of points (segments) and the sharp edges detection is performed.

Detection of closed sharp feature lines in point clouds for reverse engineering applications

The algorithm described in the article by *Demarsin et al.*[4] first estimates the normal vectors by PCA analysis of the neighborhood of each point. Once the

normal vectors are calculated, the segmentation is performed on the input point cloud based on the variation of the normals. The segmentation clusters the points to create the graph that is much smaller than the original cloud, thus making it practical for large point clouds.

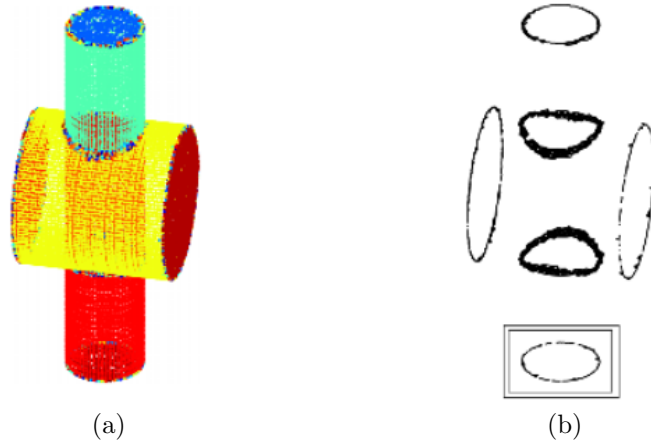


Figure 1.2: Overview of the algorithm[4]. (a) First order segmentation of two intersecting cylinders (b) The graph G_{all} connecting neighboring segments; the area bounded by the rectangle is used to illustrate the following steps of the algorithm in detail in next figure

Once the graph G_{all} is built, the algorithm process the graph in 5 steps:

1. Add edges to G_{all} , indicating a piece of a sharp feature line ($G_{all} \rightarrow G_{extended}$)
2. Build the pruned minimum spanning tree of $G_{extended}$ ($G_{extended} \rightarrow G_{pruned_mst}$)
3. Prune short branches in G_{pruned_mst} ($G_{pruned_mst} \rightarrow G_{pruned_branches}$)
4. Close the sharp feature lines in $G_{pruned_branches}$ ($G_{pruned_branches} \rightarrow G_{closed}$)
5. Smooth the sharp feature lines in G_{closed} ($G_{closed} \rightarrow G_{smooth}$)

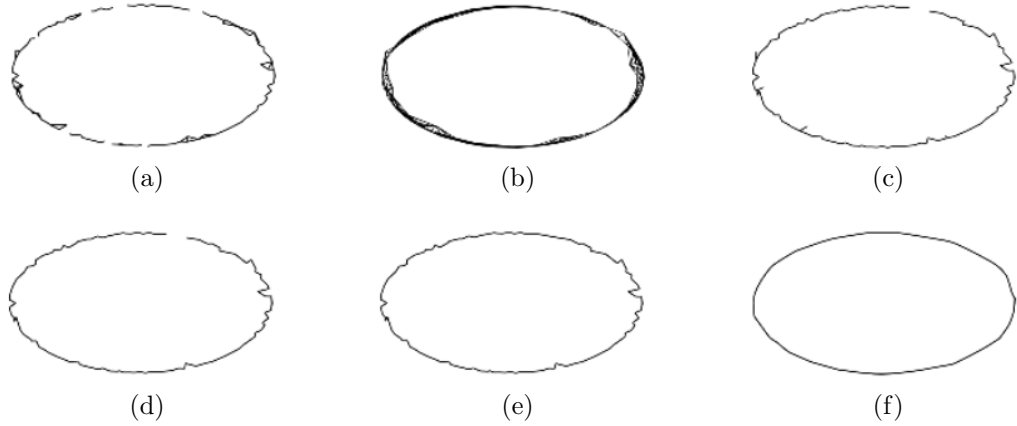


Figure 1.3: The result of each step of the algorithm showed on the detail of the figure (1.2b) bounded by rectangle. (a) Graph connecting neighboring segments (b) Added edges, indicating a piece of a sharp feature line (c) Pruned minimum spanning tree (d) Pruned short branches (e) Closed the sharp feature lines (f) Smoothed the sharp feature lines

1.1.2 Covariance analysis based methods

Covariance analysis based methods classify the points of point cloud according to probability or expectation that they belong to a feature. Based on the principal component analysis on local neighborhood of each point, the methods compute the covariance and classify the point as a potential candidate of the feature. The methods vary in the heuristic used for the point-neighborhood selection.

Feature Extraction from Point Clouds

The algorithm in the article *Feature Extraction from Point Clouds* [5] proposes a method to extract feature curves in two stages. The first stage consists of assigning a penalty weight to each point that indicates the unlikelihood that the point is part of a feature and assigning these penalty weights to the edges of a neighbor graph. Extracting a sub-graph of the neighbor graph that minimizes the edge penalty weights then produces a set of feature patterns.

The second stage is especially useful for noisy data. It recovers feature lines and junctions by fitting wedges to the crease lines and corners to the junctions.

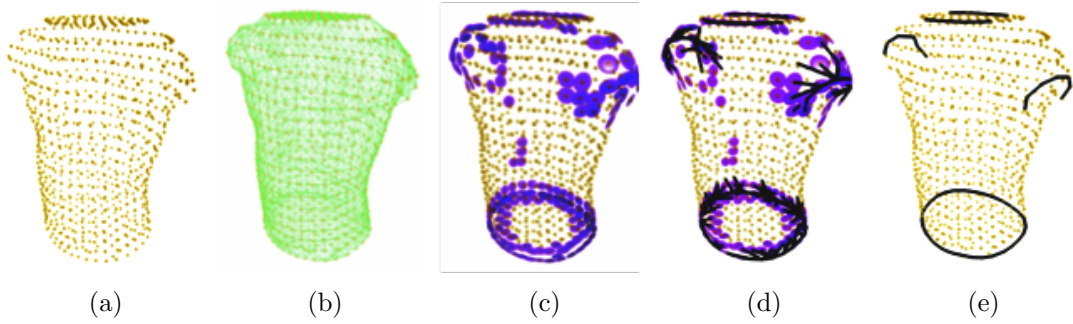


Figure 1.4: Overview of the algorithm[5] pipeline. (a) original point cloud (b) the neighborhood graph (c) point classification (d) crease pattern forming (e) purified crease pattern

1.1.3 Projection procedure based methods

Projection procedure based algorithm first approximates the point set surface. Based upon the framework of Robust Moving Least Squares (RMLS) surfaces [6], it first selects a set of potential feature points that are identified by the RMLS operator to be near to possible features.

Robust Smooth Feature Extraction from Point Clouds

The paper [7] presents an algorithm to define a set of curves that are aligned along the feature edges of a point cloud. It first detects candidate points to be near to possible features. The projection method produces jagged edges that are unable to use without further processing therefore they need to be smoothed. Based on the principal component analysis, followed by a feature growing strategy that constructs a features (represented as polylines).

In order to take into account possible poor sampling quality, the algorithm [7] proposes a technique to connect the multiple extracted polylines across the gaps to create a complete set of feature curves.

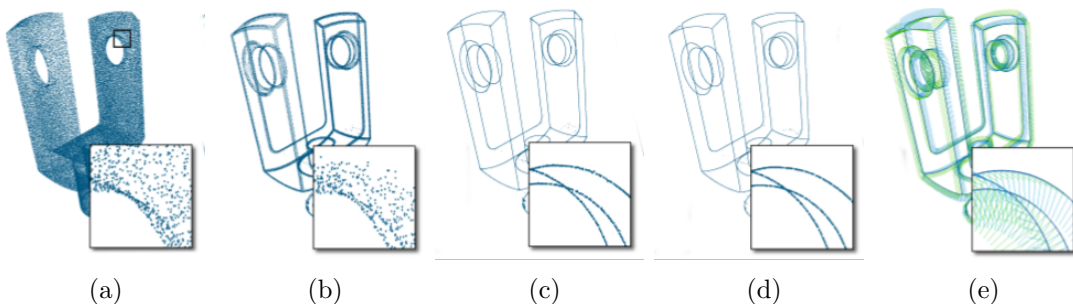


Figure 1.5: Overview of the algorithm[7] pipeline. (a) original point cloud (b) candidate points (c) projected candidate points (d) smoothed projected candidate points (e) reconstructed polylines

1.2 Extracting Feature Curves on Point Sets

Multiple techniques have investigated the identification of feature curves on point set models and polygonal models. In paper [8], there is presented an algorithm for detecting feature curves from raw point cloud. The algorithm produces viable results on irregular point clouds and point data with different level of noise.

The algorithm can be divided in four steps:

1. Approximating the neighborhoods of each point
2. Based on Fundamental forms of fitting surface, the principal directions for each potential point are computed
3. Using cross-correlation coefficient analysis, the projected points are smoothed
4. Create initial sets of the valleys and ridges by growing curves along the along the principal directions of smoothed points

The algorithm works with 4 parameters:

- ω - **MLS radius fitting**

This parameter refers to the size of a radius neighborhood of each point (step 1). In this algorithm there is used $\omega = 2.0 * \kappa$ where κ is the average edges distance.

- τ - **Curvature threshold**

In second step of the algorithm, we calculate the first and the second fundamental form of the fitting surface $g(u, v)$. Once the first and second principal curvature for point p_i is calculated, k_i is the parameter defined as the principal curvature with higher absolute value. This parameter is user-defined.

If $k_i > \tau$ then the point p_i will be added to the ridges point-set.

If $k_i < -\tau$ then the point p_i will be added to the valleys point-set.

- $\sigma_{min}, \delta_{max}$ - **Correlation threshold**

After the ridges and valleys sets are created, the algorithm removes the points that might be outliers or noise. From each point p_i in the valley or ridge, the algorithm calculates a correlation of the 2D-projections into the plane given by the point p_i and the axes x_{pi} and y_{pi} . Where x_{pi} and y_{pi} are the principal axis of δ -neighborhood $NBHD(p_i, \delta)$.

If the cross-correlation coefficient σ is lower than σ_{min} then the radius δ is increased until the σ is not higher than σ_{min} . If the value δ reaches a user-defined value δ_{max} , the point is abandoned.

1.2.1 MLS fitting

The algorithm employs moving least squares (MLS) method to fit a smooth patch for ω radius neighborhood for each point.

For each point p_i and its neighborhood $NBHD(p_i)$, there is approximated a bivariate polynomial g_i which satisfies:

$$\min \sum_{p_j \in NBHD(p_i)} \langle p_j - p_g, n \rangle^2 \quad (1.1)$$

Where p_g is projected point of p_j onto g_i along n_i ; n_i is the unit vector along z -axis of the local-coordinate system of polynomial g_i .

The polynomial g_i is a function in local coordinate frame (o_i, u, v, n_i) , where o_i is a midpoint of neighborhood $NBHD(p_i)$.

$$g(u, v) = a + bu + cv + duv + eu^2 + fv^2 \quad (1.2)$$

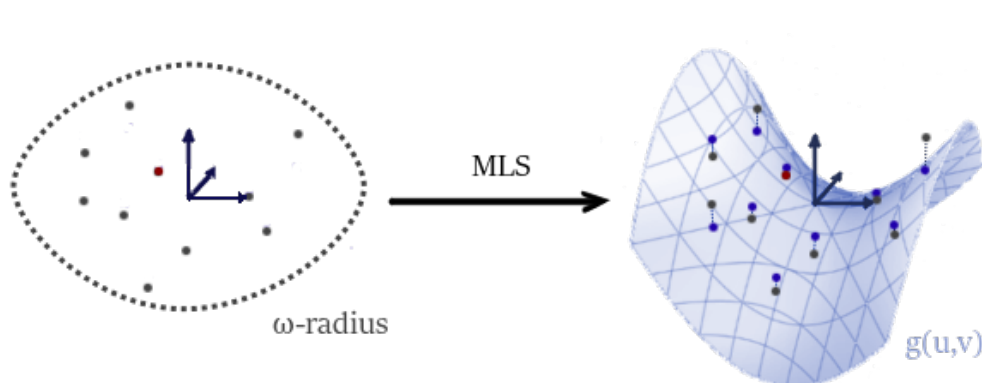


Figure 1.6: Example of polynomial $g(u, v)$ approximated from the point set. Red - desired point p_i . Gray - point included in $NBHD(p_i)$. Blue - points projected onto g .

1.2.2 Curvatures computation

Once the local fitting g_i has been achieved, curvatures of each point p_i on point set surface can be calculated based on computing the first and second fundamental quality of the surface.

Before we describe how the principal curvatures is calculated, we briefly introduce the theoretical background of the surface analysis.

We want to measure how a regular surface \mathcal{M} bends in \mathbb{R}^3 . A good way to do this is to estimate how the surface normal \mathbf{U} changes from point to point. We use a linear operator called the shape operator to calculate the bending of \mathcal{M} .

Definition: Let $\mathcal{M} \subset \mathbb{R}^3$ be a regular surface, and let \mathbf{U} be a surface normal to \mathcal{M} defined in a neighborhood of a point $p \in \mathcal{M}$. For a tangent vector v_p to \mathcal{M} at p we define the **shape operator** S as:

$$S(v_p) = -D_v(U) \quad (1.3)$$

Where $D_v(U)$ is the derivative of \mathbf{U} in the direction v .

Definition: Let \mathcal{M} be a regular surface in \mathbb{R}^3 . The first fundamental form is the bilinear form denoted by \mathbf{I} defined as:

$$\mathbf{I}(v_p, w_p) = v_p \cdot w_p \quad (1.4)$$

The second fundamental form is also the symmetric bilinear form \mathbf{II} on a tangent space \mathcal{M}_p given by:

$$\mathbf{II}(v_p, w_p) = S(v_p) \cdot w_p \quad (1.5)$$

Where v_p and w_p are tangent vectors of \mathcal{M} .

Weingarten equations[9] express the derivatives of the normal vector to a surface using derivatives of the position vector. The shape operator is given in terms of the components of the first and second fundamental forms by the Weingarten equations:

$$S = (EG - F^2)^{-1} * \begin{pmatrix} L * G - M * F & M * G - N * F \\ L * G - M * F & M * G - N * F \end{pmatrix} \quad (1.6)$$

Where E, F, G and L, M, N are the coefficients of the first and second fundamental forms of the surface respectively. They play important roles in many intrinsic properties of a surface. The parameters can be calculated using various methods. For our purposes we show the calculation from the parametric surface $\mathbf{r} = \mathbf{r}(x, y)$.

$$\mathbf{r}(x, y) = \mathbf{a} * x + \mathbf{b} * y + z(x, y) * \mathbf{c} \quad (1.7)$$

The coefficients of the first fundamental form \mathbf{I} are sometimes referred to as the metric coefficients. From the parametric equation for the surface they can be calculated from scalar products of the first partial derivatives of the equation:

$$E = \frac{\partial \mathbf{r}}{\partial x} * \frac{\partial \mathbf{r}}{\partial x}, \quad F = \frac{\partial \mathbf{r}}{\partial x} * \frac{\partial \mathbf{r}}{\partial y}, \quad G = \frac{\partial \mathbf{r}}{\partial y} * \frac{\partial \mathbf{r}}{\partial y} \quad (1.8)$$

The second fundamental form \mathbf{II} and its coefficients quantify how the unit normal vector \mathbf{N} changes orientation on a curved surface. The coefficients are calculated as follows:

$$L = \frac{\partial^2 \mathbf{r}}{\partial^2 x} * \mathbf{N}, \quad M = \frac{\partial^2 \mathbf{r}}{\partial x \partial y} * \mathbf{N}, \quad N = \frac{\partial^2 \mathbf{r}}{\partial^2 y} * \mathbf{N} \quad (1.9)$$

Where \mathbf{N} is the unit normal vector to the surface.

Let v_1, v_2 be unit eigenvectors of S associated to the eigenvalues λ_1, λ_2 . Then the principal curvatures k_1 and k_2 of S are precisely the λ_1 and λ_2 [9] which are also denoted as k_{max} and k_{min} (Assuming $k_{max} > k_{min}$). Since the shape operator S is represented as 2×2 matrix, the eigenvalues can be calculated using [10].

Based on the analysis of experiments and results, the algorithm works with the value k_i which is for point p_i defined as

$$k_i = \begin{cases} k_{min} & |k_{min}| > |k_{max}| \\ k_{max} & |k_{min}| < |k_{max}| \end{cases}$$

Candidate points

Once we have a k_i calculated for each point p_i , the algorithm selects such points those satisfies the condition $|k_i| > \tau$ where τ is an user-defined threshold.

1.2.3 Potential feature points detection

The feature points are smoothed by projecting them onto their principal axis of neighborhoods and smoothed by methods based on statistical analysis of point distributions on approximate plane. The algorithm has adopted the method from the related algorithm [2].

In point of fact this is probably the most important part of the algorithm, moreover the improvements in terms of robustness of noise are made in this stage, we introduce the theoretical background of the principal component analysis.

Principal component analysis

Principal component analysis (PCA) is a multivariate technique that analyzes a data in which observations are described by several inter-correlated quantitative dependent variables. Its goal is to extract the important information from the data, to represent it as a set of new orthogonal variables called principal components, and to display the pattern of similarity [11].

Given a set of points in Euclidean space, the first principal component corresponds to a line that passes through the multidimensional mean and minimizes the sum of squares of the distances of the points from the line. The second principal component corresponds to the same concept after all correlation with the first principal component has been subtracted from the points. [12].

In computational terms, the principal components are found by calculating the eigenvectors and eigenvalues of the data covariance matrix. This process is equivalent to finding the axis system in which the co-variance matrix is diagonal. The eigenvector with the largest eigenvalue is the direction of greatest variation, the one with the second largest eigenvalue is the (orthogonal) direction with the next highest variation. Each eigenvalue is proportional to the portion of the “variance” that is correlated with each eigenvector.

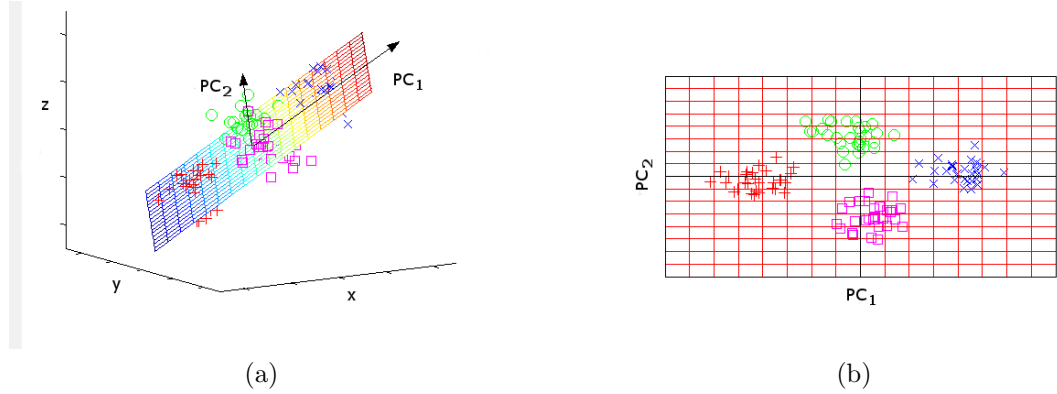


Figure 1.7: PCA identifies the two-dimensional plane that optimally describes the highest variance of the data. (a) Found plane is given by two principal components PC_1 and PC_2 ordered by variance. (b) points projected onto the plane. Source of the image: [13]

In this algorithm, for each point in feature candidates selected in the previous stage, the covariance matrix is calculated of the δ -neighborhood. The δ -neighborhood of a point p is defined as a set of points p_i which satisfies $\|p - p_i\| < \delta$.

We compute the principal components from the covariance matrix [14] of the point set defined as:

$$C = (p_1 - \bar{p}, \dots, p_k - \bar{p}) \cdot (p_1 - \bar{p}, \dots, p_k - \bar{p}) \quad (1.10)$$

Where the point \bar{p} is a midpoint of the neighborhood defined as:

$$\bar{p} = \frac{1}{k} \sum_{j=1}^k p_j \quad (1.11)$$

Once the covariance matrix is calculated, we calculate the eigenvalues $\lambda_0, \lambda_1, \lambda_2$ (with loss of generality, assuming $\lambda_0 \geq \lambda_1 \geq \lambda_2$) and the referring eigenvectors v_0, v_1, v_2 of the covariance matrix C .

As shown in figure 1.7, we can construct a two-dimensional plane α from vectors v_0, v_1 and midpoint \bar{p} . After the neighborhood points are projected to the plane α we translate the points into the local coordinate system.

In order to evaluate the “likelihood” that the point p is a part of the feature curve, we analyze the correlation of the neighborhood points in 2D space. The correlation coefficient denoted as $\sigma_{X,Y}$ is measuring the degree of correlation[15].

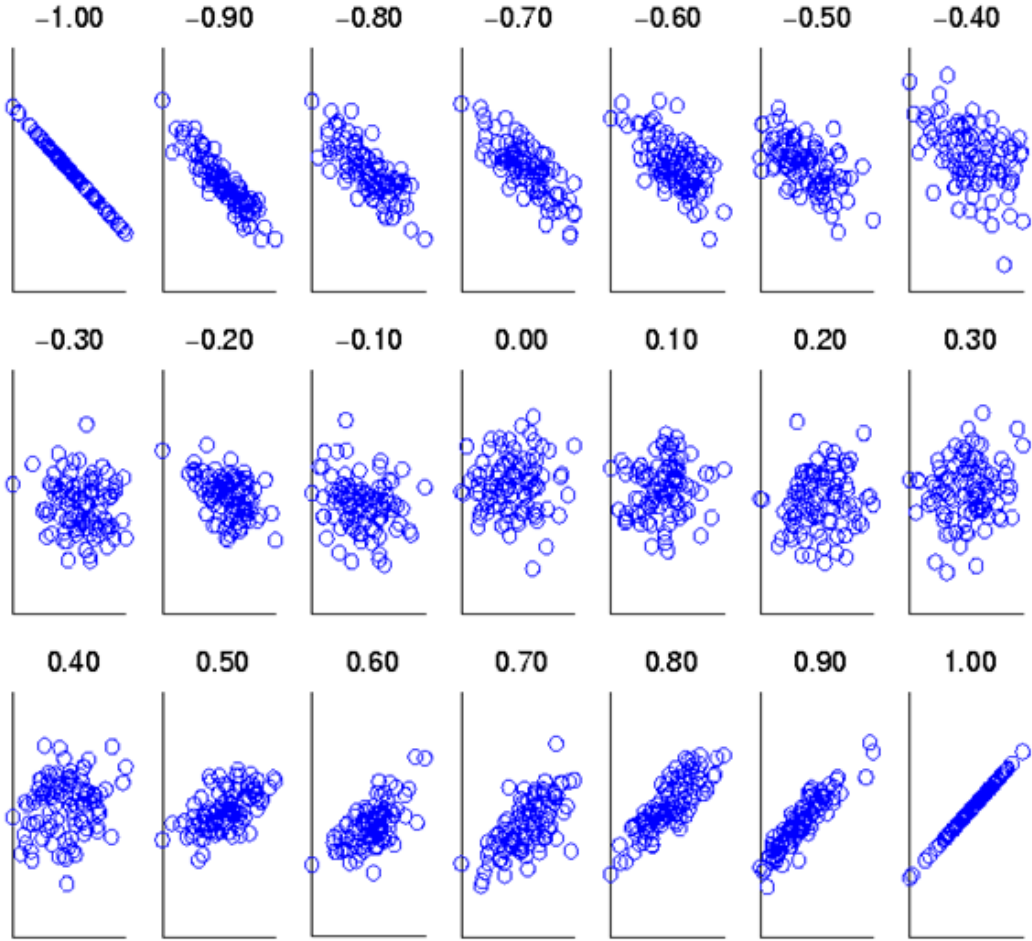


Figure 1.8: Several sets of (x, y) points, with the correlation coefficient of x and y for each set. Source of the image:[16]

The correlation $\sigma_{X,Y}$ is calculated as:

$$\sigma_{X,Y} = \frac{Cov(X,Y)}{\sqrt{D(X)} \cdot \sqrt{D(Y)}} \quad (1.12)$$

As seen from the figure 1.8, the higher absolute value of the $\sigma_{X,Y}$ is calculated, the higher linear dependence of a point set can be observed. In order to select only those points that are part of a feature, the algorithm calculates the $\sigma_{X,Y}$ for each point and checks whether is higher than user-defined constant σ_{min} .

If the $\sigma_{X,Y}$ is lower than σ_{min} , the δ - neighborhood is selected again for increased δ . The procedure is repeated until the $\sigma_{X,Y}$ is not sufficiently large enough or δ reaches δ_{max} . If δ reaches δ_{max} the point is eliminated from the potential feature points and it is marked as outlier.

1.2.4 Smoothing the feature points

Let $R(S)$ and $V(S)$ be ridges and valleys obtained from the previous stage respectively. The sets $R(L)$ and $V(L)$ are point sets that describe the feature polylines.

All points obtained from the previous step $R(S)$ and $V(S)$ are added into two separate priority queues with priority function as the principal curvature ratio. Points with higher curvature ratio have a higher priority in a queue.

The point p_i is popped from a priority queue and added to $R(L)$ ($V(L)$) as l_0 . Then the tracing procedure starts in two different directions $T_{max_{abs}}$ and $-T_{max_{abs}}$. The tracing procedure samples point neighborhood for each l_k , deletes (or mark as flagged) the points of the neighborhood and samples a new point l_{k+1} until stopping conditions are satisfied. We continue creating the features using seed points until the queues are not empty. We describe the sampling process further in our proposed algorithm.

Optimization of feature curves

Once the polylines are created, there may occur some gaps between the polylines. The algorithm check all end points of the features and completes the polyline connecting. It connects all end points that lie in within the aperture cone given by angle μ .

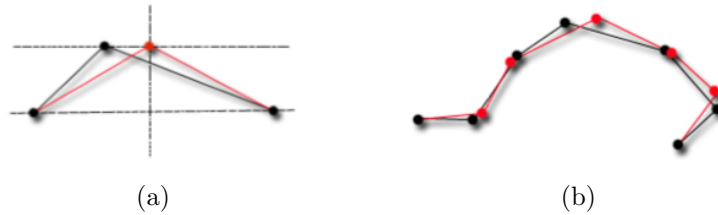


Figure 1.9: Smoothing the feature lines. Source of the image: [8]

Smoothing the feature lines

The polylines are smoothed by applying the uniformity scheme on interior points for 1 or 2 passes. This step will be in our algorithm intentionally omitted since it leads to loss of the information that are later used for the algorithm comparison.

2. Proposed algorithm

In this chapter, we propose a new algorithm that is mostly based on the paper [8]. We first describe the implementation of the base algorithm that will be our reference algorithm.

Once we will have a reference results on our data, we can then clearly see the assets made by our improvements.

Similarly as the previous algorithm, our algorithm works with these parameters:

- ω - **MLS radius fitting**
- τ - **Curvature threshold**
- σ_{min}, δ, c - **Correlation threshold**
Unlike the previous algorithm, the δ -neighborhood is not expanded in the stage 3. We have rather chosen the anisotropic approach for the selection of the δ_c -neighborhood.
- ξ - **Feature post-processing**
Compared to the previous algorithm, the proposed algorithm has one additional stage that process the feature lines. The stage fills gaps between the end points of detected lines and connects the T-shaped junctions.

2.1 Limits of the previous algorithms

Despite to the fact that algorithm provides a robustness to complexity of a 3D-model, there are certain edge cases that the algorithm does not handle correctly.

In the step where the correlation is calculated (subsection 1.2.4) the correlation coefficient is calculated for each neighborhood-pick with increasing parameter δ . Since the neighborhood-search and calculation of a correlation coefficient from point set is rather complex operation, the question then arises, “*Is there a simpler, more efficient and still robust-to-noise method how to pick the proprietary points of the potential feature curve?*”

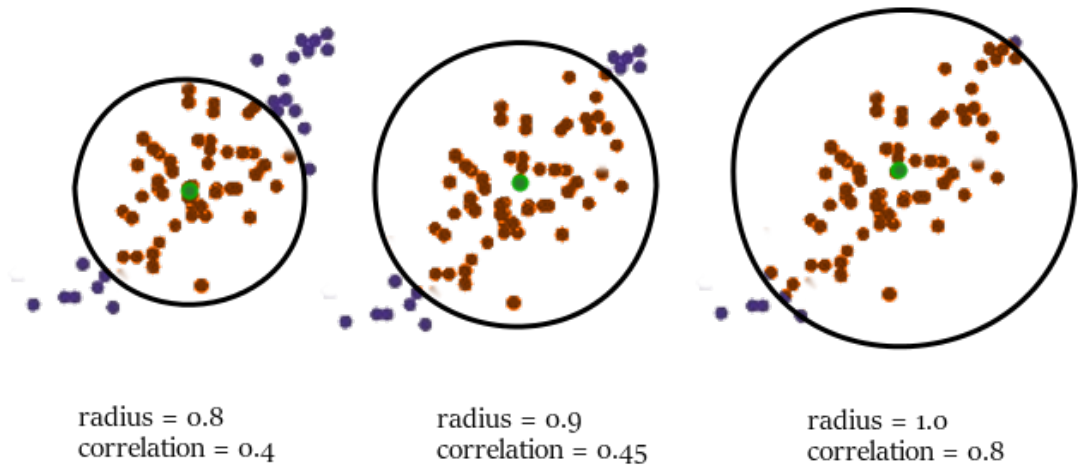


Figure 2.1: The radii of the neighborhood and referring correlation coefficient

One can observe that the searching procedure within the neighborhood does not take into account the valuable information obtained from the previous step - principal directions and principal curvatures; it does pick all points within the radius δ regardless of the shape of the curvature that can be roughly predicted from the principal curvatures and vectors.

In next subsections we will introduce ourselves the edge cases that the previous method could not handle.

2.1.1 T-shaped connections

In some cases, feature curves are connected in T-shaped junction. As seen in the figure 2.2, arbitrary point close to any junction can never be classified as part of a feature curve; the correlation coefficient is too low.

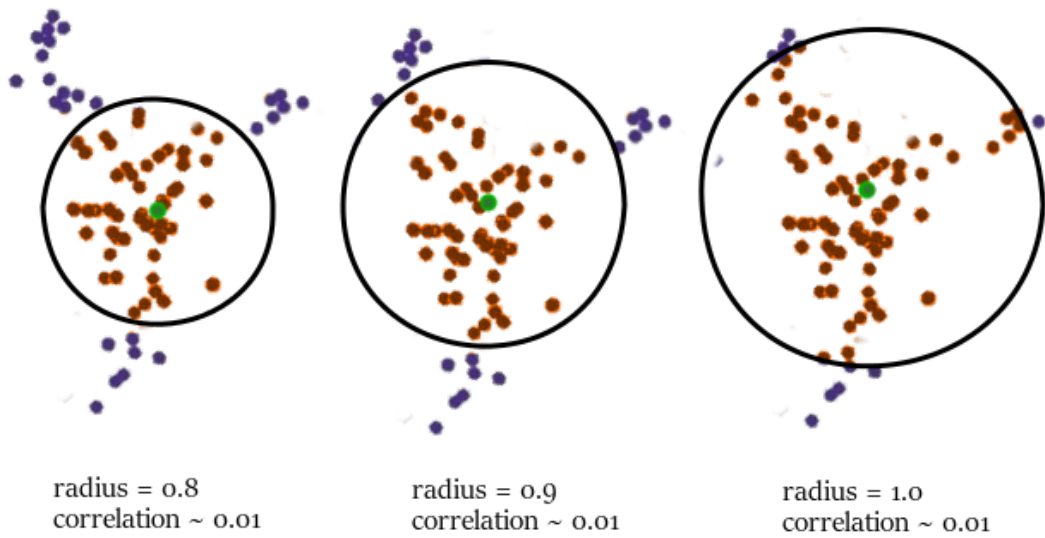


Figure 2.2: Example where arbitrary point is classified as outlier

In complex and sometimes noisy 3D-objects, it may lead to loss of feature curves information.

2.1.2 Curves interference

Two parallel feature curves with relatively low distance from each other makes an interference so the neighborhood of a point from one curve can exceed to another curve.

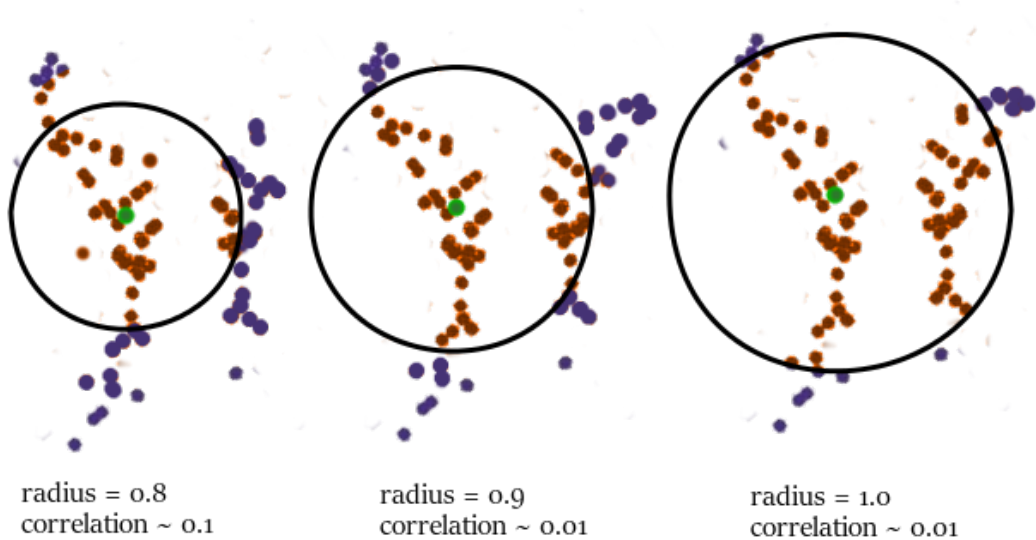


Figure 2.3: Two interfering feature curves

In figure 2.3 there is shown an example situation where the interference lead to the loss of the information. The correlation coefficient is too low to classify the point as a part of the feature.

2.2 Improvements

In the article described in 1.2, in the step where the potential feature points are separated, the algorithm does not take into account the information about the point from the previous step. It can cause certain deficiency(2.1) of the algorithm.

For each point, the neighborhood space that can be searched around the desired point is always sphere with limited radius.

2.2.1 Neighborhood optimization

Instead of the sphere-neighborhood that the algorithm can construct multiple times for one point (until the correlation coefficient is not sufficient) we can construct an ellipsoid-neighborhood only once per point. The orientation of the ellipsoid is given by the principal directions of the shape operator at a point.

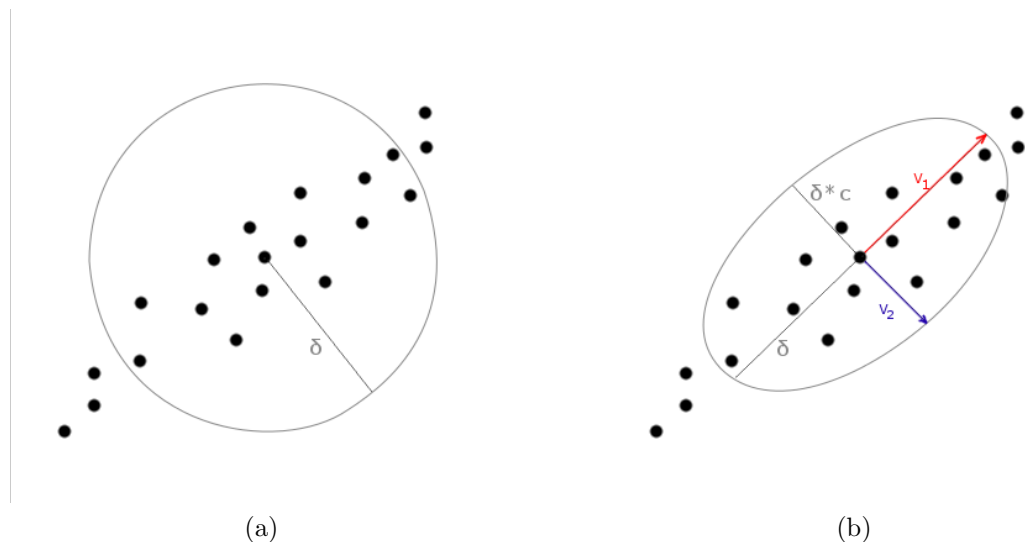


Figure 2.4: The comparison between (a)the circle-neighborhood approach (previous algorithm) and (b)our method. δ , c are user-defined parameters. We used δ as $\delta = \omega * 1.5$. Vectors v_1 and v_2 are principal vectors, where the principal curvature regarding to v_1 must be greater than v_2

2.3 The optimized algorithm

In this section, we design a new, improved feature extraction method primarily based on the algorithm as described in (1.2) that particularly solves the problems in edge cases as we mentioned in (2.1). We will focus on the limits of the algorithms listed in (2.1). Furthermore, we add to the algorithm one extra step that runs a post-processing over the detected features. That will provide more precise

and human-eye-friendly results for visualization of detected features.

Algorithm 1: The improved method

Input: $\omega, \delta, c, \sigma, \tau$

Data: *pointCloud*

Output: labeled points, potential feature points, feature lines

```

1 foreach point  $p \in \text{pointCloud}$  do
    | neighborhood := PickNeighborhood( $p, \omega$ );
    | CalculatePrincipalCurvatures( $p, \text{neighborhood}$ );
    | CalculatePrincipalDirections( $p, \text{neighborhood}$ );
    | if MaxPrincipalCurvature( $p$ ).absoluteValue  $> \tau$  then
    |   | MarkPointAsCandidate( $p$ );
2 foreach point  $p \in \text{candidatePoints}$  do
    | neighborhood := PickNeighborhood( $p, \delta, c$ );
    | if CrossCorrelation( $p$ )  $> \sigma_{min}$  then
    |   | MarkPointAsPotentialFeature( $p$ );
3 foreach point  $p \in \text{potentialFeaturePoints}$  do
    | if  $p \notin \text{any feature}$  then
    |   | featureSeed :=  $p$ ;
    |   | feature := CreateFeature(seed);
    |   | ExpandFeature(feature);
4 ChainFeatures();
5 SmoothFeatures();
Render();

```

The details of the implementation and the data structures used are provided in next chapter.

2.3.1 Principal curvatures calculation

As in the reference algorithm in the subsection 1.2.1, we run the MLS-fitting on the input point cloud. For each point p , we find a approximation of the bivariate polynomial function for the ω - neighborhood of point p .

For each point p_i , we evaluate principal curvatures and vectors as in subsection 1.2.2 and calculate the value k_i which refers to the principal curvature with higher absolute value.

If the value k_i of the point p_i satisfies the condition $|k_i| > \tau$ the point p_i is marked as a candidate point.

2.3.2 Correlation

In this step, we achieve removing the outlier points from the candidate point set; For each candidate point $p \in F_{can}$ we pick the δ -neighborhood $N_{can}(p)$ from the F_{can} .

Once we have the set $N_{can}(p)$ selected, we project the points from the neighborhood on the plane defined as follows: First, we build covariance matrix defined as:

$$C = (p_1 - \bar{p}, \dots, p_k - \bar{p}) \cdot (p_1 - \bar{p}, \dots, p_k - \bar{p})^T \quad (2.1)$$

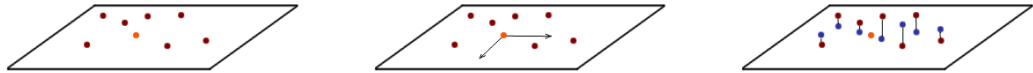
Where \bar{p} is a midpoint of the neighborhood calculated as:

$$\bar{p} = \frac{1}{q} \sum_{p_j \in N} p_j \quad (2.2)$$

Where N is the neighborhood. Once the covariance matrix C is calculated, we calculate the eigenvalues and their corresponding eigenvectors of the matrix C . As we explained in the section (1.2.3), the eigenvectors with the largest eigenvalues correspond to the dimensions that have the strongest correlation in the point set. We will find two linearly independent vectors that define the plane by picking two eigenvectors with the highest corresponding eigenvalues.

Thus we calculate the eigenvalues $\lambda_0, \lambda_1, \lambda_2$ and then their corresponding eigenvectors V_0, V_1, V_2 . (without the loss of generality, assuming $\lambda_0 \leq \lambda_1 \leq \lambda_2$).

We then create the plane α defined by vectors V_0 and V_1 with the origin \bar{p} . Next, the points within the neighborhood are projected onto plane α to treat the points as the set of 2D-vectors within the plane α and origin \bar{p} .



(a)

(b)

(c)

Figure 2.5: Red:points of the picked neighborhood. Orange: the midpoint \bar{p} . Blue: projected points onto plane α . (a) calculated midpoint \bar{p} (b) calculated base vectors of plane α using PCA (c) points projected onto plane α

In order to calculate the cross-correlation coefficient that reflects the correlation of the points within the neighborhood, the correlation is calculated along the x -axis of the plane α . Projected points are then analyzed by correlation σ_{XY}

where the random variables X and Y are given by the local x and y coordinates of the 2D-vectors. The cross-correlation coefficient σ_{XY} is defined as:

$$\sigma_{XY} = \frac{cov(X, Y)}{\sqrt{D(X)}\sqrt{D(Y)}} \quad (2.3)$$

The σ_{XY} has always value between -1 and 1. It represents the degree of linear correlation between X and Y which, in our case, means distribution along a potential feature line.

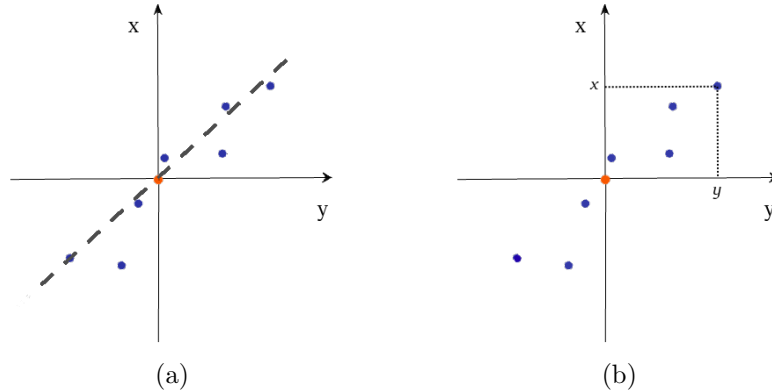


Figure 2.6: As the blue points in Figure 2.5 refer to projected points onto the plane α , this figure shows the 2D-projection of the plane. (a) the dashed line is in the direction of the first principal component of the covariance matrix (b) plane α projection x and y values for an arbitrary point

The resulting σ_{XY} refers to linear correlation where 1 or -1 is a perfect linear function and 0 totally uncorrelated points. As a matter of the fact we do not consider whether the line is in descend For that reason we have a constant σ_{min} that is a minimum to *absolute* value of the correaltion $\sigma_{X,Y}$

Picking a neighborhood

As in the algorithm outline (Algorithm 1), as well in this section we mention δ -neighborhood with no further details. In the algorithm (1.2), the δ -neighborhood refers to a sphere-bounded point set which has initially a radius δ_{min} and expanding continuously until the sphere radius reaches δ .

In section (2.1) we described several limits of this method; there are potential edge cases where the method might eliminate “good” candidate points. Thus we changed the sphere-policy to an ellipsoid-policy points selection.

In section (2.2.1) we describe another approach to selecting the neighborhood points. Instead of constructing a sphere we create an prolate spheroid oriented in direction of V_0 ; where V_0 is an eigenvector obtained from the covariance matrix C . The parameters for the ellipsoid are $\delta_A = \delta$ and $\delta_B = \delta * c$ (as shown in Figure 2.4).

2.3.3 Growing feature curves

The very first step we have to make before we start to growing the feature is to build a priority queue with the curvatures ratio as a weighting function. However, instead of making a simple queue, we created a heap with the same weighting function. In case of queue, the asymptotic complexity of an insertion is $O(n)$ and a removal $O(1)$. In the other hand, the heap provides a better performance; the complexity of an insertion is $O(\log N)$ as well as a removal [17].

Once we have the heap created and filled, we pop out the root element and set is as the seed. From the seed, we obtain the information about the principal directions calculated in the beginning of the algorithm (2.3.1). We set a vector dir_{seed} as a principal direction that refers to principal curvature with higher absolute value.

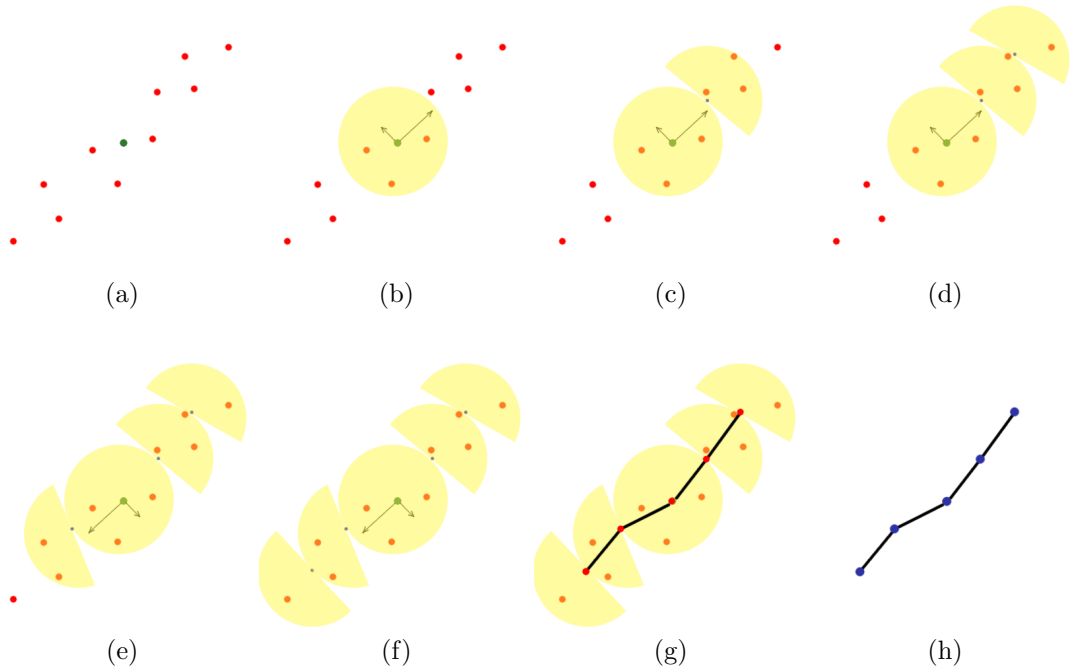


Figure 2.7

From the seed (green on the Figure 2.7), we continue in direction dir_{seed} . There we place another seed and then pick a neighboring points within the radius ω . We filter the points that was not previously picked and calculate the vector dir as the average of all principal vectors of neighboring points. We continue in direction dir , place a new seed and repeat entire operation until the empty neighborhood is reached (2.7d). Then we start again from the initial seed, but in direction of the $-dir_{seed}$. When the iterations are done, we collect the seeds and add a feature.

Note: when calculating the average direction of next step, all vectors must be “swept” in the consistent direction. Which mean there can not be any two vectors V_1, V_2 where $dot(V_1, V_2) < 0$. Before we calculate the mean direction, we check the sign of the vector as shown on figure below.

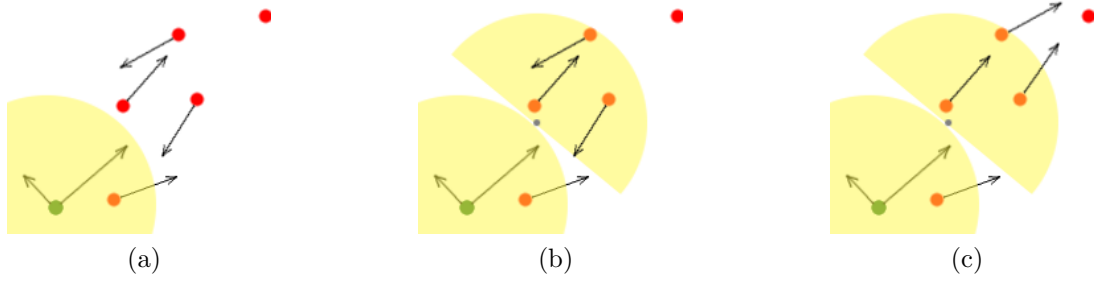


Figure 2.8: Zoomed figures (b) and (c) of 2.7. The green point is the seed. (a) initial state with seed and sphere (b) new seed (small blue point) in the direction of principal vector of seed. The hemisphere covers 3 new points, but 2 of them have principal vectors in opposite direction from seed (towards to seed) (c) normals reverted

2.3.4 Feature lines post-processing

Since we use an anisotropic approach to filtering the potential feature points (as described in section 2.3.2), we can assume that more feature points close to T-shaped junctions are preserved.

The question then arises, “*Once having the filtering method that better handles the T-shaped junctions, are we able to join the polylines with the same approach as we join the end points of two polylines?*”

First, we will show how do we process the gap between the two end points of the feature lines.

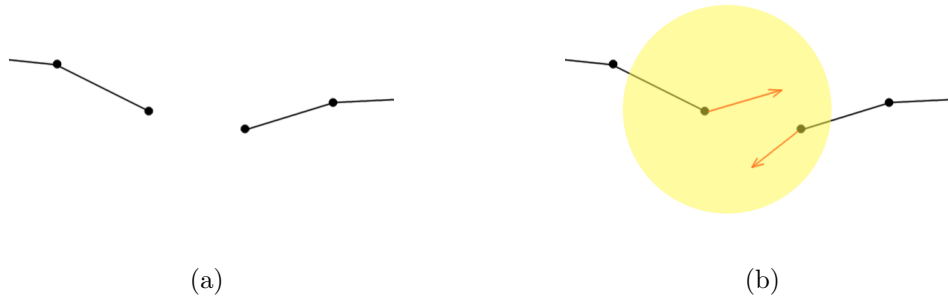


Figure 2.9: (a) The two end points of the feature polyline (b) The ω radius of an end point and the principal vectors of the end points.

In order to to classify two endpoints as considerable for connection, they must fulfill these conditions:

1. The distance between the endpoints must be lower than ω .
2. Let the v_A and v_B be the principal vectors (obtained) from the stage no. 2. The dot product $dot(v_A, v_B)$ must be lower than 0.

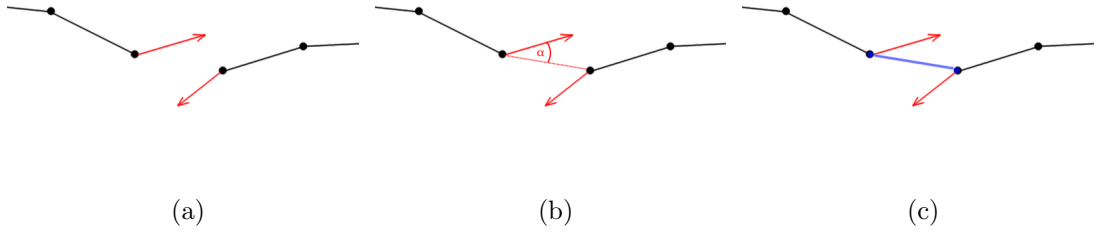


Figure 2.10: (a) The end points of two feature polylines (b) The ω radius of an end point and the principal vectors of the end points.

If the two basic conditions are fulfilled, the algorithm checks whether the principal vectors v_A, v_B satisfy the condition:

$$\text{dot}(v_A, B - A) > \xi \quad (2.4)$$

Where ξ is a user-defined constant that can be loosely understood as the arcsine of the maximum angle between the endpoints of two merged polylines.

T-shaped junctions

As a matter of the fact we have already mentioned the *T-shaped junctions* in terms of feature points, the algorithm so far is not capable to grow a feature lines that splits in two ones. The feature polyline is represented as a sequence of points.

In subsection 2.1.1 we demonstrated how can the reference algorithm lose the potential feature points near to the T-shaped junctions. As we use the anisotropic method of neighborhood selection, our proposed algorithm preserves some points near junctions so there is a higher probability that the endpoint of feature curves near a junction are closer to the junction.

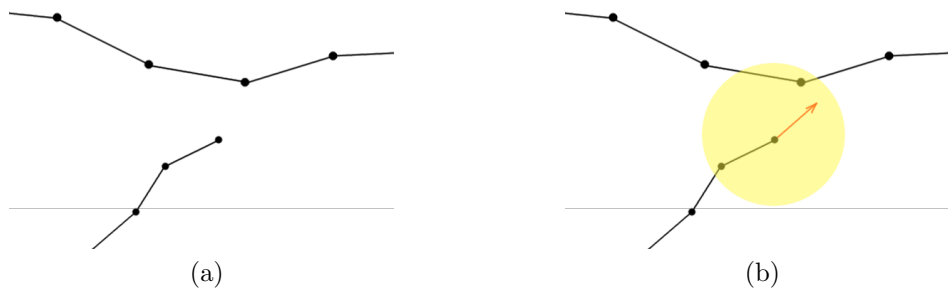


Figure 2.11: (a) The end point of a feature polyline (b) The ω radius of an end point and the principal vectors of the end points.

Similarly to the previous case, the end points of the features will be tested. Unlike the end point connections, there is one end point of a feature picked and finds the closest point from a feature polyline within the ω -radius.

If the point X of a feature is found, the process of connecting is similar as in chaining two feature lines.

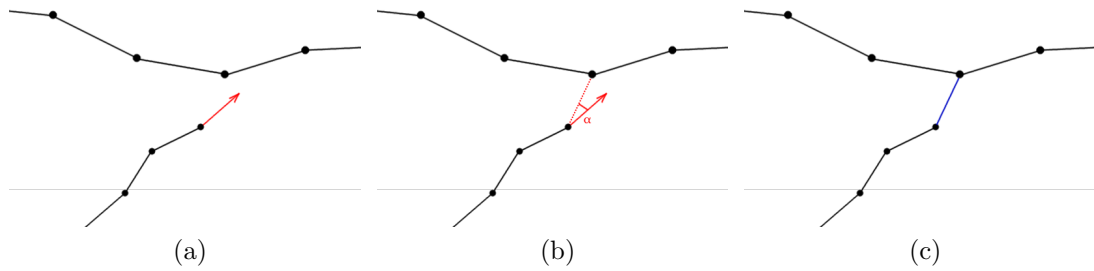


Figure 2.12: (a) (b) The ω radius of an end point and the principal vectors of the end points.

As well as in the previous case, the algorithm calculates the dot product of the vectors v_A and $(X - A)$; where v_A is the principal vector of the end point A and the $X - A$ is the normalized direction vector given by endpoint A and the selected point X . Thus before the T-connection is generated it must satisfy the condition:

$$\text{dot}(v_A, A - X) > \xi \quad (2.5)$$

3. Implementation

The application is written in C# on the platform .NET 4.5 and using the academic framework MorphoMe3cs[18].

3.1 Application structure

On the very top of the application there is program that parses a name of the test and proprietary arguments from command line. In our case, the **Meshload** test is started.

The subprogram **MeshloadTest** requires a parameter representing a full or absolute path to the configuration file for the test. The configuration file is in general simple text file with parameters in form $\langle parameter \rangle = \langle value \rangle$. separated with newlines. Those parameters represents the file with an input 3D-object and parameters for the algorithm used. So the example configuration file looks like follows:

```
#Bunny with Gaussian noise added (value 0.0005)
file= C:\Documents\bunny0005.ply
ground_truth= C:\Documents\sphere_ground_truth.obj
kappa=2.2
lower_bound=-190
upper_bound=190
heuristic=pca
join_tees=true
# end of Bunny
```

Figure 3.1: example configuration file. The lines leading with # are comments so they are not parsed.

The program recognizes following parameters:

- **file** - the absolute path to the file with the object
- **ground_truth** (optional) - the absolute path to the ground truth results.
- **lower_bound, upper_bound** - the parameters that threshold the principal curvatures in first phase of the algorithm. In the algorithm description it is referred by τ .
- **kappa** - the coefficient used for evaluation of ω (size of the neighborhood)
- **heuristic** - the user specifies the approach used in the second stage of the algorithm. There can be used our algorithm described in section 2.3 or the reference algorithm described in section 1.2.
 - incremental - the approach presented in 1.2. The δ -neighborhood points are picked from the sphere with increasing parameter δ . Thus this method we named *incremental*

- PCA - the approach presented in our algorithm. Instead of “inflating sphere” there is used an ellipsoid based on the PCA analysis.
- **join_tees** (optional) - (default = *true*). This parameter enables or disables the part of feature polylines post-processing that connects the T-shaped junctions.

The algorithm is placed in a separate package `MeshProcessing.FeatureDetection` within the project *MorphoMe3cs*. There are also various helper classes placed outside the package e.g. data structures or geometric meta-data. Those classes will be introduced in a separate subsection.

The *Meshload* test works as a certain initializer and starter of an entire algorithm. It validates the parameters, load model, analyzes it and finally, prepares the algorithm.

The package `MeshProcessing.FeatureDetection` contains:

- class FeatureDetection
The class that implements the algorithm. It contains two most important methods: `Initialize()` and `ProcessCloud()`.

The `Initialize()` method sets all parameters needed for the algorithm e.g. ω , δ , τ and prepares for the selected heuristic.

The `ProcessCloud()` method works in 4 main steps:

1. extracts the feature candidates using the method `featureCandidates()`
2. process the candidates using the method `potentialFeaturePoints()` which is called from the `IHeuristic`
3. generates the feature lines using the method `raisePolyLines`
4. post-process the features using the method `processFeatures`

- class Feature
Our own representation of the feature polyline. Besides the linked list of the points, it contains the end points of the feature that are used in feature post-processing.
- interface IHeuristic
The interface that has a declaration of the method `potentialFeaturePoints()` that selects a potential feature points from the feature candidates. In the algorithm 1, the method refers to stage 2.
- class IncrementalHeuristic
The implementation of interface `IHeuristic` according the description in section 1.2
- class PCAHeuristic
The implementation of interface `IHeuristic` according the description in section 2.3

4. Results

In this chapter we will examine the results given by our algorithm (2) and compare it with the results of the algorithm (1.2). We will test the performance and precision of the extracted curves on two 3D-models with varying amount of added noise.

The noise added is a Gaussian noise using the open-source 3D-editing tool Blender [19]. The Blender environment provides an options to add the noise using the *randomize* function under the package *mesh tools*. User can define size of the amplitudes and the random seed.

Run-time tests

Similarly as in the algorithm description in pseudocode 1, we divide the run-time of the algorithm into 4 phases.

- **phase1** - In this phase, the MLS-fitting method evaluates a local function for each point and calculates the principal curvatures. Candidate points from the point cloud are then extracted according maximal absolute value of the principal curvatures.
- **phase2** - Based on principal component analysis, from the candidate points are filtered out points those are classified as outliers.
- **phase3** - After the potential feature points are extracted, the algorithm creates a poly-line representation of features, which is a sequence of short lines.
- **phase4** - Feature lines are post-processed. The gaps between two curves are under certain conditions filled and the T-shaped junctions are connected.

All those 4 phases are tested separately and measured the run-time is written into table with results

The algorithm was tested on a notebook Dell Inspiron 15Z with Intel©Core i7-3537U 3.1 GHz processor and 6GB memory.

4.1 Synthetic data

Before we measure the accuracy of the algorithms, we need to have a model where the results of a perfect accurate algorithm can be predicted. This result is commonly known as the ground truth[20].

From here we see, how the noise does affect the results and the robustness to noise.

4.1.1 Sphere

For the testing purposes, we created a synthetic data with various levels of Gaussian noise. We generated a simple sphere with a triplet of lines raised out of the surface. These lines are connected in one point.

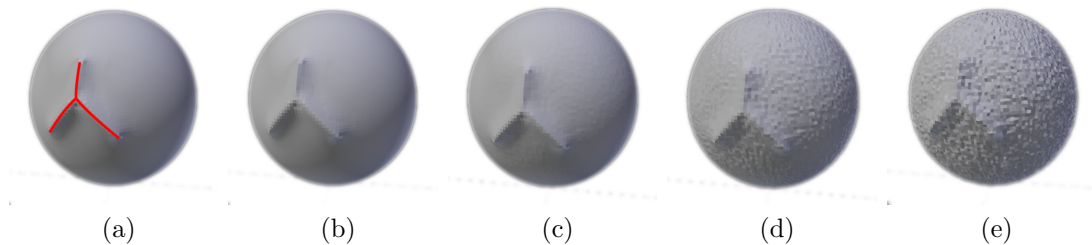


Figure 4.1: Our synthetic sphere model with various level of Gaussian noise added. (a) No noise added with the expected feature curve highlighted (b) No noise added (c) Model with Gaussian noise of 0.001 (d) Model with Gaussian noise of 0.005 (e) Model with Gaussian noise of 0.01

Since the three feature curves on our sphere are connected in one point. The deficiency of the original algorithm[8] occurs as explained in 2.1. In the other hand, our algorithm shows a preservation of curves information in the joint of the triplet.

Incremental heuristic tests

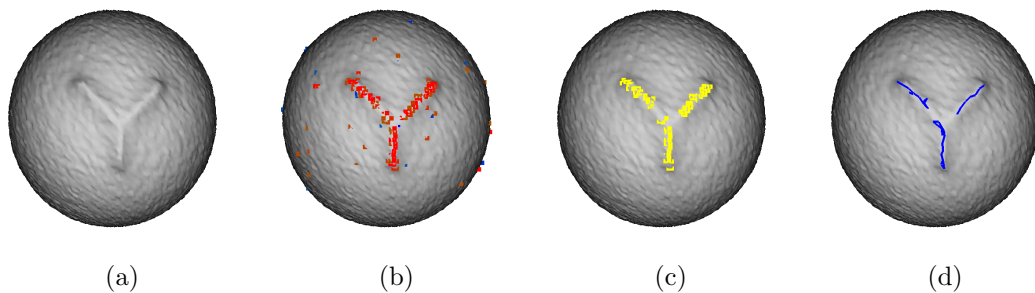


Figure 4.2

From the figure 4.2d we can see that the added noise has deformed the feature so the algorithm was unable to recovery the information.

PCA heuristic tests

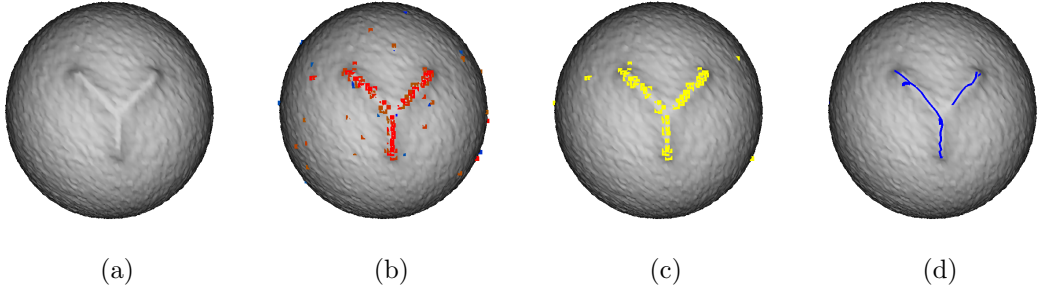


Figure 4.3

Unlike the previous method, the PCA heuristic has preserved at least one of two connections.

Table 4.1: The run-time performance of the proposed algorithm over the sphere

model	phase 1	phase 2	phase 3	phase 4
sphere (noise 0.01)	8.378s	0.106s	0.021s	0.004s
sphere (noise 0.005)	8.211s	0.014s	0.021s	0.005s
sphere (noise 0.001)	8.282s	0.012s	0.022s	0.005s
sphere (no noise added)	8.170s	0.011s	0.023	0.005s

The table above shows that the added noise cause a higher computational cost of the algorithm. The **phase 2** is the stage of the algorithm that filters out the outlier points from the feature candidates. Since the added noise causes the higher principal curvatures in certain points, the step before evaluates more candidates.

Comparison with the ground truth

The main advantage of this model is the clear and transparent predictability of the ground truth. Since the model has been done manually using the 3D-editor, we created a separate file that contains the points that should be placed along the detected feature.

In order to evaluate the correctness of the algorithm, we need to define the error that says how close is the extracted features to the ground truth. We define the function $Err(GT, L)$ which takes a line set L and the ground truth points GT and then calculates the distances between each point $p \in GT$ and the lines $l \in L$.

The Err function can be defined as:

$$Err(GT, L) = \text{Var}(X) \quad (4.1)$$

Where X is a set of numbers representing a distance from a point to a feature line. Thus the equation can be defined as follows:

$$Err(GT, L) = \sum_{p \in GT, l \in L} (w(p, l) - \overline{w(p, l)})^2 \quad (4.2)$$

Where $w(p, l)$ is our custom weighting function defined as:

$$w(p, l) = \exp(\text{distance}(p, l)) \quad (4.3)$$

We have chosen the \exp function in order to penalize abandoned points that are not close to any feature line.

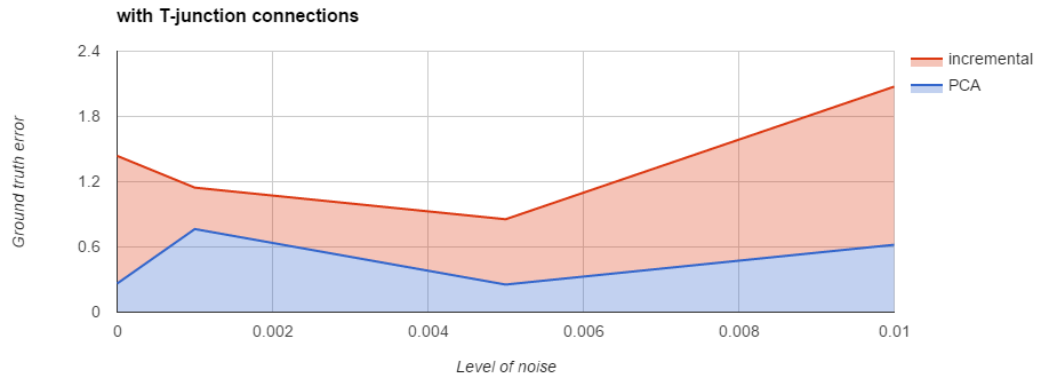


Figure 4.4: This graph describes the Err value at various level of noise. The feature lines was post-processed and the T-shaped junctions was connected

From the graph in the figure 4.4, there can be seen that our method using the PCA-heuristic for feature detection is much closer to the ground truth compared to the incremental-heuristic.

Moreover, there can be seen a paradox on the graph at noise of value 0.005. The noise causes that the error is slightly lower than the error for the noise of value 0.001. This phenomenon is caused by the random seed that generated such a noise that lowered the distance between the features and the ground truth points.

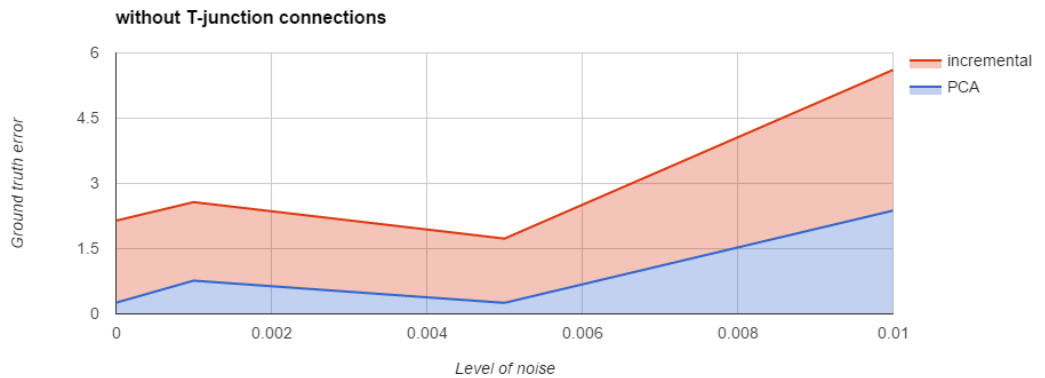


Figure 4.5: This graph describes the Err value at various level of noise The feature lines was post-processed, but the operation that connects the T-shaped junctions was omitted.

Another phenomenon can be seen when we compare the figures 4.4 and 4.5. As seen on the graphs at value of noise 0, at the incremental method curve (red), the error is relatively high on the graph above (where the T-junctions were connected). It is caused by the edge case described in the section 2.1. The straight straight lines in the middle of the triplet are interfering thus the points close to the joint are filtered out in the PCA-filtering. Then the features can not be connected without a notable bias.

Moreover, there can be also observed that the gap between the errors of the incremental and PCA heuristic increases with the noise added to the model. The feature detection method that does not connect the T-junctions in the post processing reveals the higher robustness to the noise of the algorithm.

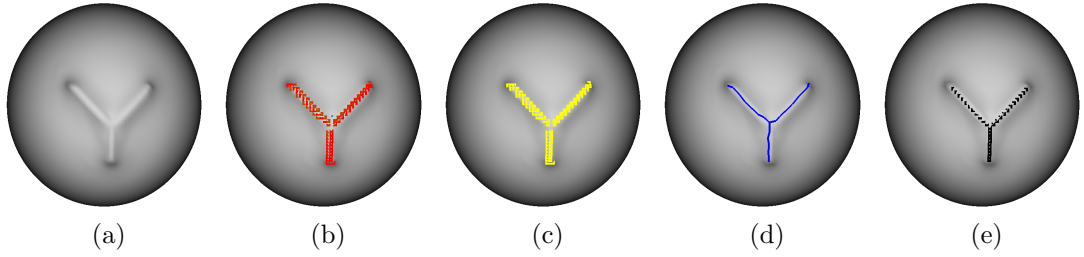


Figure 4.6: No noise added (a) clean model (b) candidate points (c) potential feature points (d) detected feature curves (e) ground truth

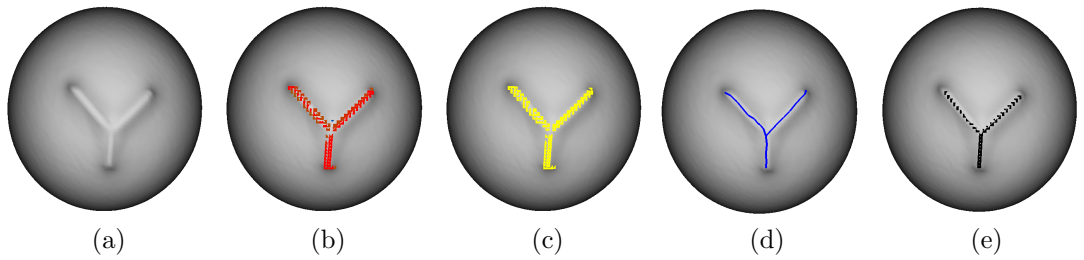


Figure 4.7: Added noise of 0.001 (a) clean model (b) candidate points (c) potential feature points (d) detected feature curves (e) ground truth

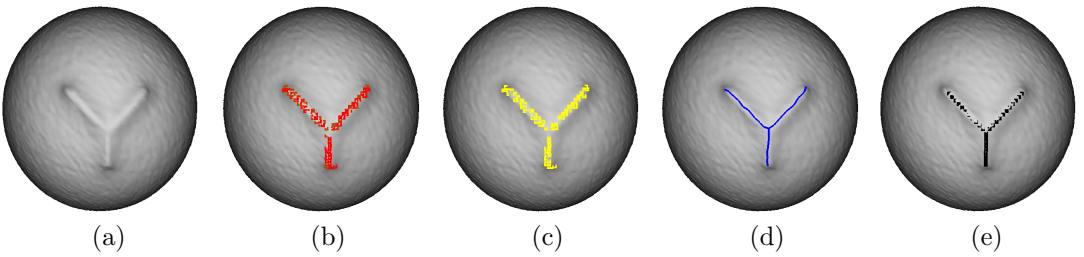


Figure 4.8: Added noise of 0.005 (a) clean model (b) candidate points (c) potential feature points (d) detected feature curves (e) ground truth

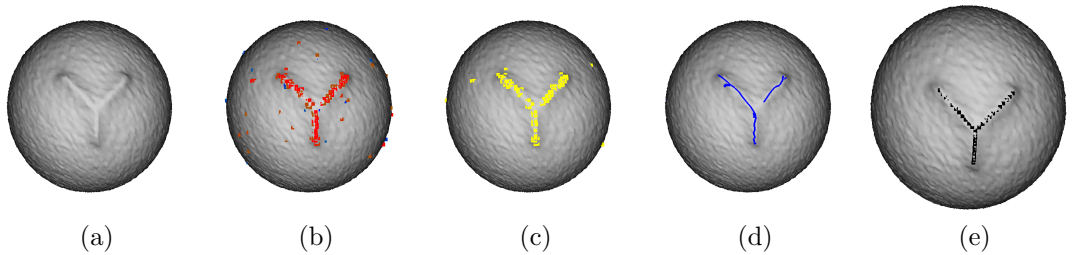


Figure 4.9: Added noise of 0.01 (a) clean model (b) candidate points (c) potential feature points (d) detected feature curves (e) ground truth

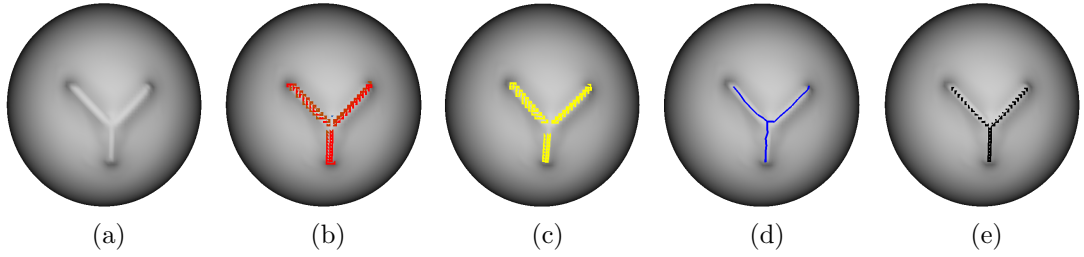


Figure 4.10: No noise added (a) clean model (b) candidate points (c) potential feature points (d) detected feature curves (e) ground truth

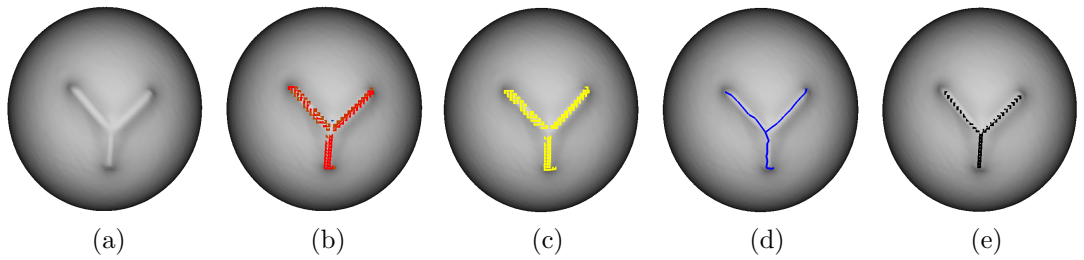


Figure 4.11: Added noise of 0.001 (a) clean model (b) candidate points (c) potential feature points (d) detected feature curves (e) ground truth

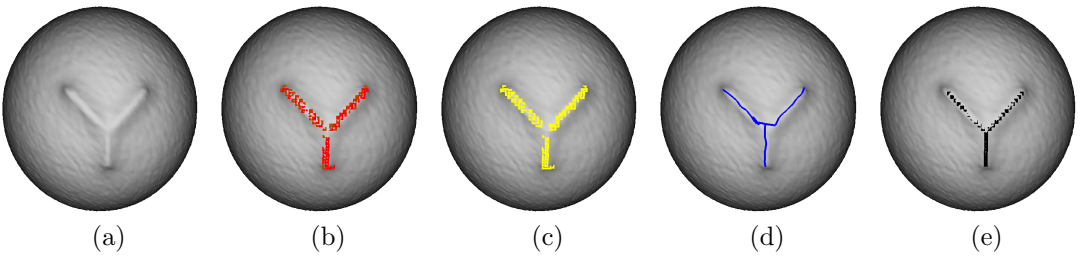


Figure 4.12: Added noise of 0.005 (a) clean model (b) candidate points (c) potential feature points (d) detected feature curves (e) ground truth

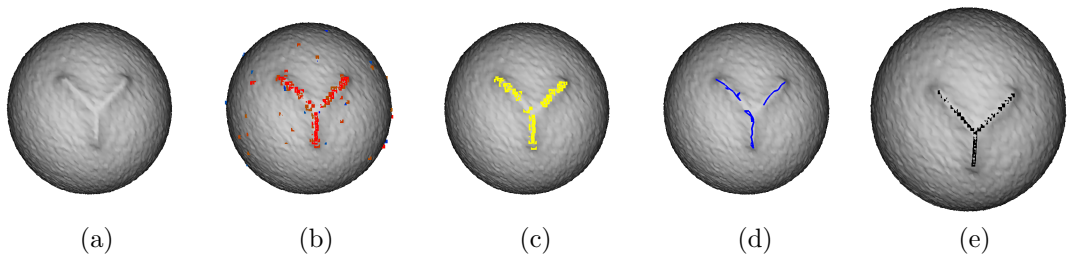


Figure 4.13: Added noise of 0.01 (a) clean model (b) candidate points (c) potential feature points (d) detected feature curves (e) ground truth

4.2 Scanned data

The algorithm shows a good results on the synthetic data, however now we test the algorithm on the “real data”.

4.2.1 Bunny

As the algorithm was tested on the well-known 3D-model Stanford bunny [21], we test the algorithm on the same model to see the improvements brought by our algorithm.

Incremental heuristic tests

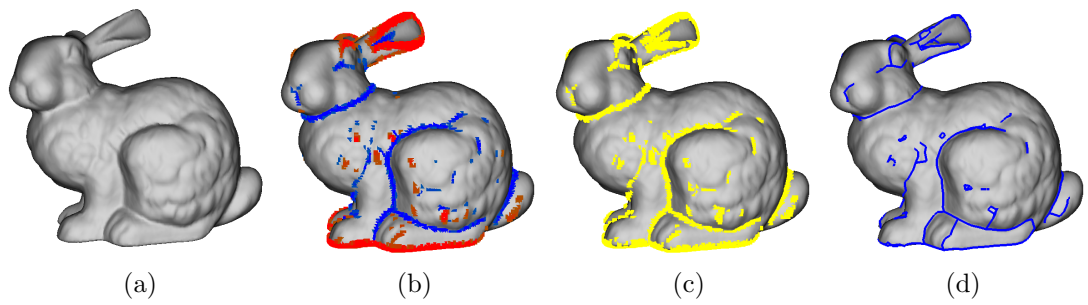


Figure 4.14

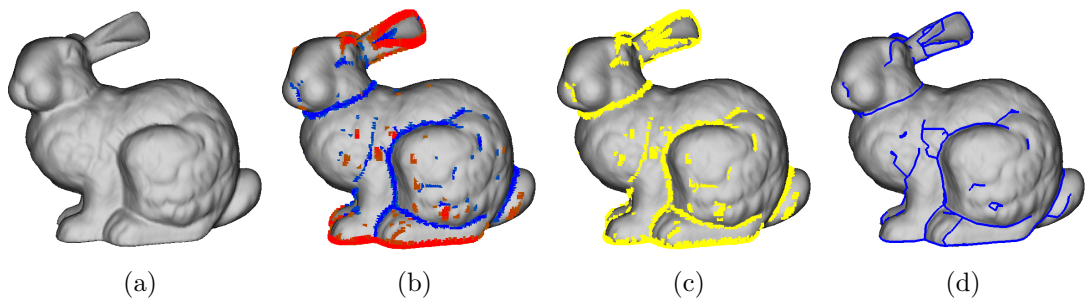


Figure 4.15

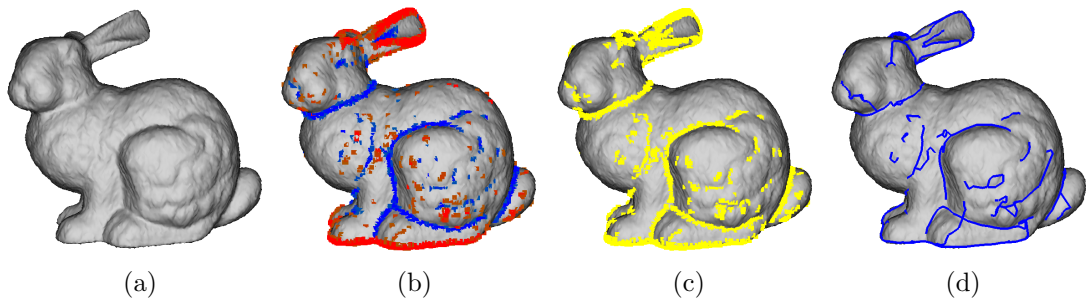


Figure 4.16

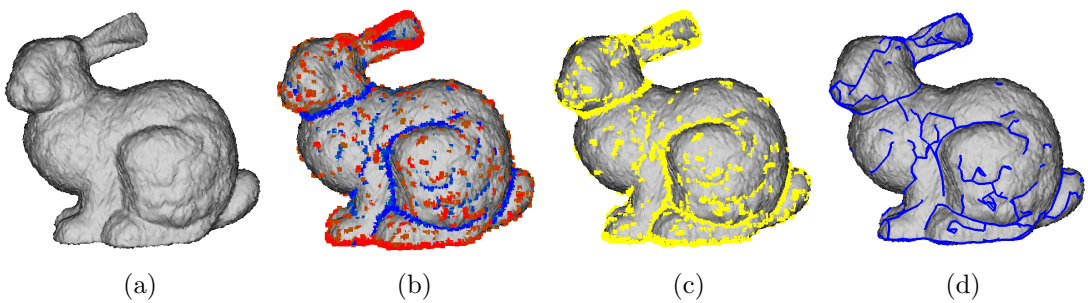


Figure 4.17

PCA heuristic tests

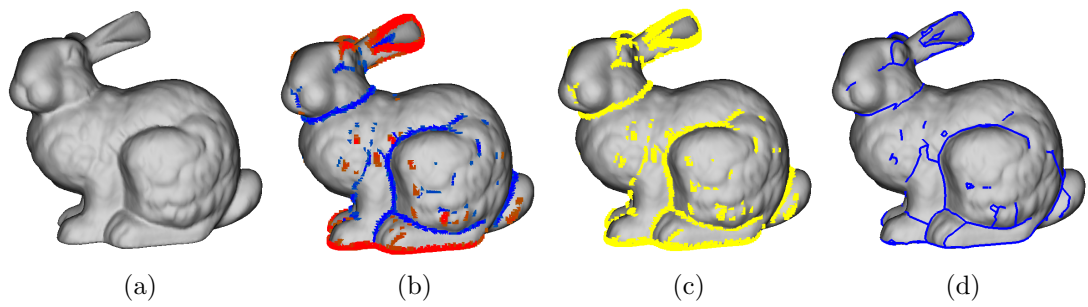


Figure 4.18

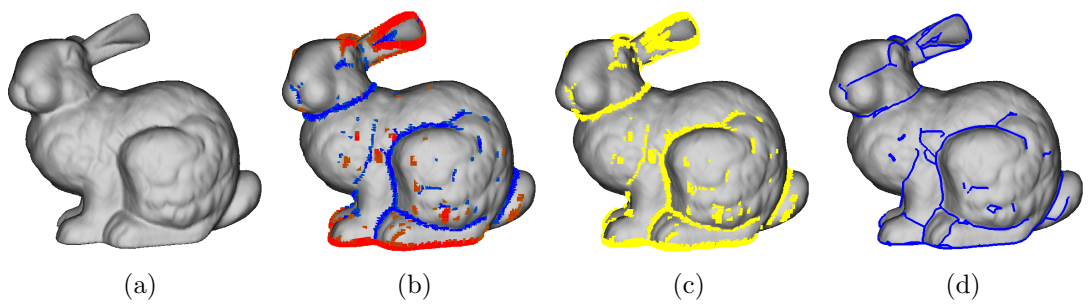


Figure 4.19

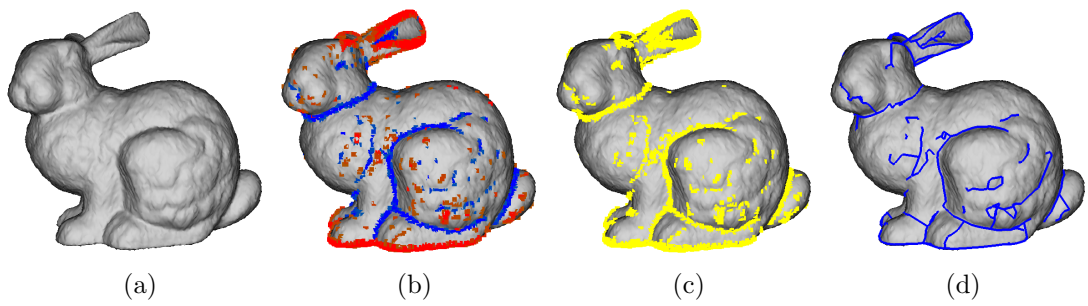


Figure 4.20

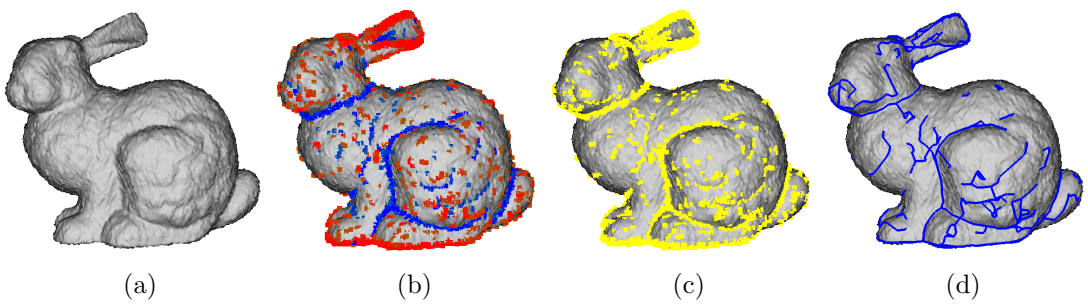


Figure 4.21

4.2.2 Armadillo

In order to test the performance of the algorithm and the computational costs, we need to test the algorithm on the 3D model that have at least 100,000 vertices. On the higher amount of analyzed points, we can better compare the PCA-heuristic to the incremental-heuristic.

For our case, we choose the well-known model Armadillo[22], also from the Stanford 3D scanning repository. The original model has 172,974 vertices and 345,994 faces, which is enough for our testing purposes.

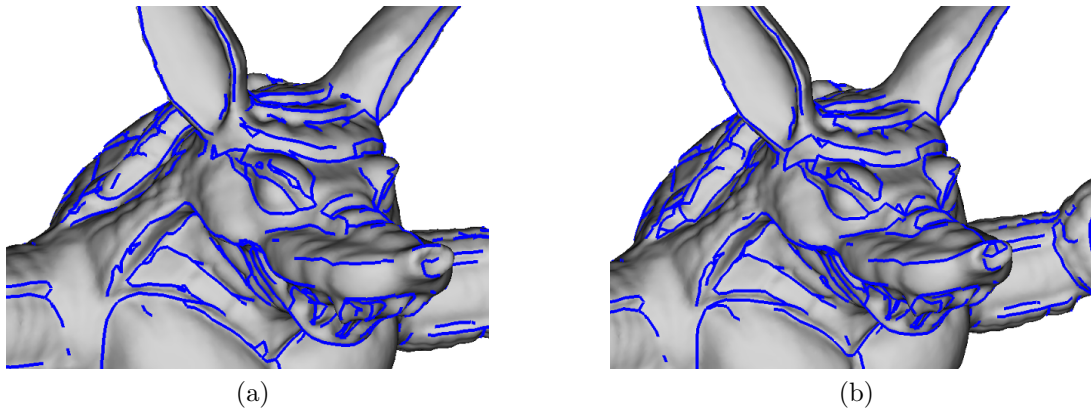


Figure 4.22: (a) PCA-heuristic (b) Incremental-heuristic

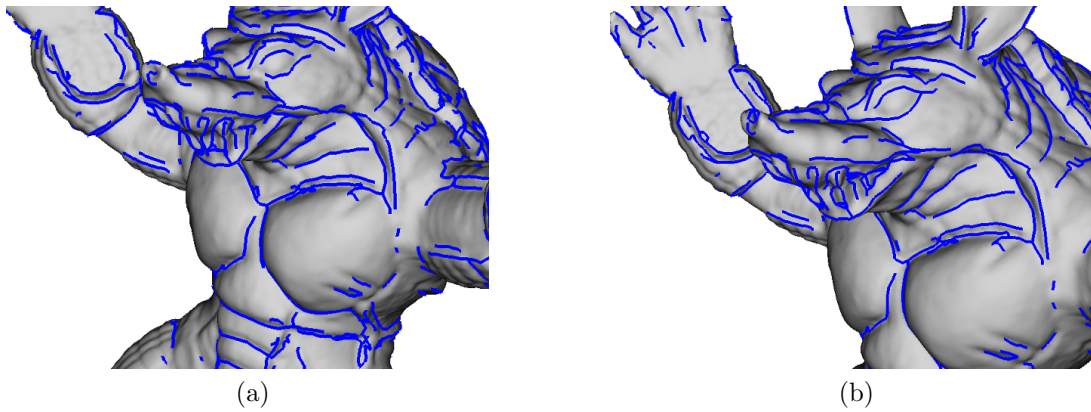


Figure 4.23: (a) PCA-heuristic (b) Incremental-heuristic

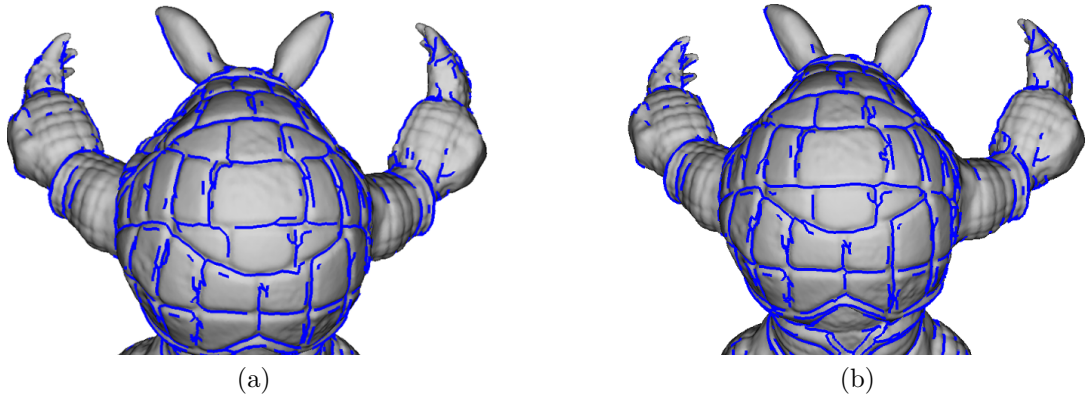


Figure 4.24: (a) PCA-heuristic (b) Incremental-heuristic

Run-time tests

From the figures 4.22b, 4.23b and 4.24b there can be seen that both heuristics produces a reasonable outputs. However, the choice of the heuristic makes a significant difference in terms of run time performance.

heuristic	phase 1	phase 2	phase 3	phase 4
incremental heuristic	72.244s	17.603s	0.074s	1.741s
PCA heuristic	72.689s	1.044s	0.075s	1.835s

Table 4.2: The run-time performance on Armadillo model. The values in the cells refers to the duration of the phase at the specified heuristic.

In the table 4.2 there can be observed that the PCA heuristic is incomparably more effective at the **phase 2**.

4.2.3 Palate

The 3D model introduced in this section is a real 3D scan obtained with the digital laser scanner Roland Picza LPX-1200 3D.

The model is a scan of human palate. Which is a mesh of 46499 vertices and 92090 faces. On the scan there can be observed errors that arise naturally during 3D scanning.

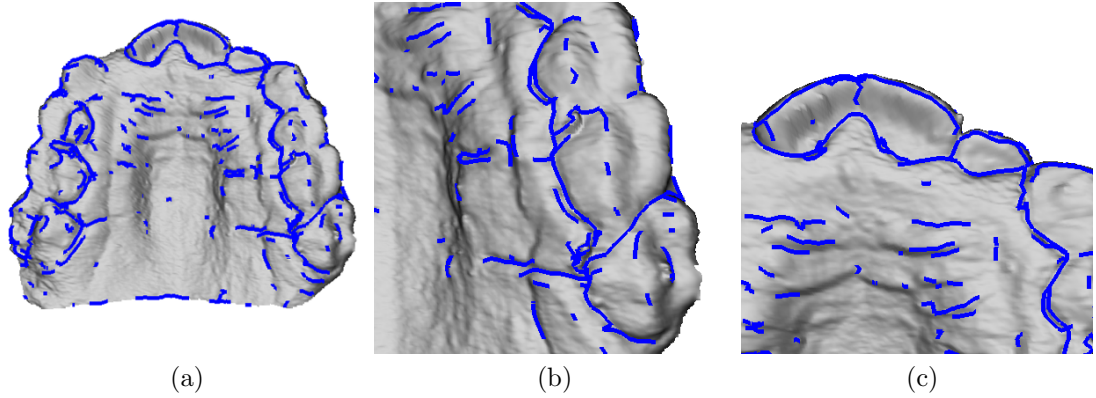


Figure 4.25: The features extracted from molar using the PCA-heuristic (a) the top view (b) detailed view of incisor (c) detailed view of molars

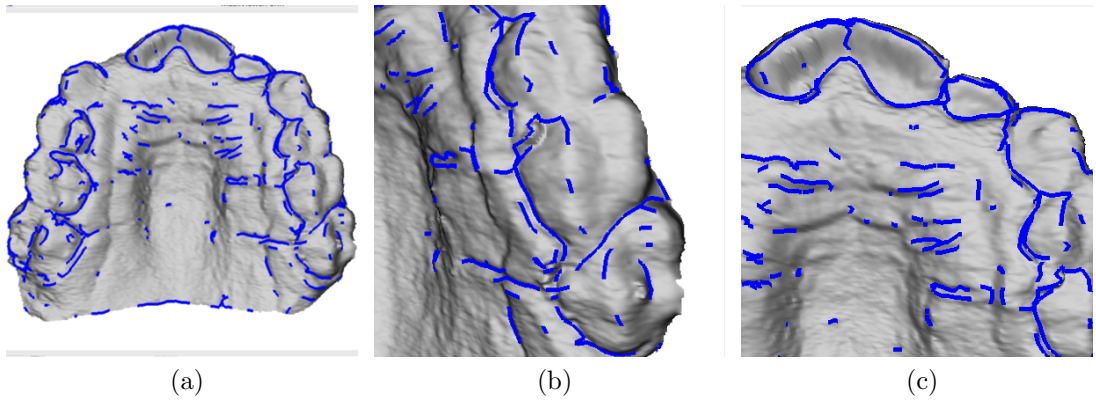


Figure 4.26: The features extracted from molar using the incremental heuristic (a) the top view (b) detailed view of incisor (c) detailed view of molars

heuristic	phase 1	phase 2	phase 3	phase 4
incremental heuristic	13.653s	1.693s	0.029s	0.108s
PCA heuristic	13.109s	0.154s	0.030s	0.127s

Table 4.3: The run-time performance on a palate model. The values in the cells refers to the duration of the phase at the specified heuristic.

As well as in the previous section, in the table 4.3 there can be observed that the PCA heuristic is incomparably more effective at the **phase 2**.

Conclusion

The thesis provides an overview of the feature detection methods used for the purposes of reverse engineering, filtering, simplification, non-photo realism, reconstruction and other geometric processing applications. Since there is no universal or exact definition of what constitutes a feature, its exact definition depends on the particular problem, for purposes of this thesis, we defined the features as the valleys and ridges of the 3-dimensional objects.

The goal of this thesis was to propose an algorithm that detects the feature curves from the point clouds. The algorithm implementation is focused on the robustness to noise, the correctness of the results and the computational costs. The algorithm was tested on edge cases that are occurred on the real 3D-models obtained from digital scanners.

This master thesis presents a new algorithm that is particularly based on the algorithm *Extracting Feature Curves on Point sets* [8] and extracts the valley-curves and ridges-curves from raw point cloud. The thesis propose an algorithm that changes the approach of the algorithm to isotropic and test it over various 3D models. Our method leverages the valuable information extracted in principal component analysis (PCA) of the reconstructed surface and changes the approach of the feature curves approximation. In addition the new method opened a door to improved gap-connections between the detected feature polylines and the t-shaped junction connections which could not be handled by the referring algorithm.

We have properly tested both algorithms as well on synthetic data as on the “real data” obtained from digital scanner. The testing focused on the robustness to noise, computational complexity and the correctness of the result. Since the correct results on the “real data” is hard to measure, the correctness was primarily tested on the synthetic models that have got pre-defined expected results.

The test showed that the robustness to noise has been preserved even though the computational costs was reduced. The second stage of the algorithm shows the significant processing speed improvements. The proposed method constructs ellipse only once and calculates the principal components based on the analysis of the points within the ellipse. The reference algorithm, in the other hand calculates the principal components repeatedly. When we used our feature lines post-processing, the results of the reference algorithm was evaluated as biased.

Despite to the fact the results on synthetic models as well as on “real models” are encouraging, there is still room to improvements in areas of feature curves post-processing. In future work, we aim at changing the results to parametric curves (e.g. Bézier curves) instead of the polylines. The curve weights can be calculated from the curvatures or the correlation of the feature points. It could, for instance, serve as pre-process for the feature curves post-processing; our variation considers only the direction vector at the end point of a feature.

Bibliography

- [1] Y. Lee and S. Lee, “Geometric snakes for triangular meshes,” in *Computer Graphics Forum*, vol. 21, pp. 229–238, Wiley Online Library, 2002.
- [2] M. Pauly, R. Keiser, and M. Gross, “Multi-scale feature extraction on point-sampled surfaces,” in *Computer graphics forum*, vol. 22, pp. 281–289, Wiley Online Library, 2003.
- [3] B. “gettingmyfeetwet”, “3d scanning methods,” 2015. [Online; accessed 20-July-2015].
- [4] K. Demarsin, D. Vanderstraeten, T. Volodine, and D. Roose, “Detection of closed sharp feature lines in point clouds for reverse engineering applications,” in *Geometric Modeling and Processing-GMP 2006*, pp. 571–577, Springer, 2006.
- [5] S. Gumhold, X. Wang, and R. MacLeod, “Feature extraction from point clouds,” in *Proceedings of 10th international meshing roundtable*, vol. 2001, Citeseer, 2001.
- [6] S. Fleishman, D. Cohen-Or, and C. T. Silva, “Robust moving least-squares fitting with sharp features,” in *ACM Transactions on Graphics (TOG)*, vol. 24, pp. 544–552, ACM, 2005.
- [7] J. Daniels, L. K. Ha, T. Ochotta, and C. T. Silva, “Robust smooth feature extraction from point clouds,” in *Shape Modeling and Applications, 2007. SMI’07. IEEE International Conference on*, pp. 123–136, IEEE, 2007.
- [8] X. Pang and M. Pang, “Extracting feature curves on point sets,” *International Journal of Information Engineering and Electronic Business (IJIEEB)*, vol. 3, no. 3, p. 1, 2011.
- [9] E. Abbena, S. Salamon, and A. Gray, *Modern differential geometry of curves and surfaces with Mathematica*. CRC press, 2006.
- [10] O. Knill, “Eigenvalues and eigenvectors of 2x2 matrices,” 2015. [Online; accessed 20-July-2015].
- [11] H. Abdi and L. J. Williams, “Principal component analysis,” *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 4, pp. 433–459, 2010.
- [12] I. T. Jolliffe, “Principal component analysis and factor analysis,” *Principal component analysis*, pp. 150–166, 2002.
- [13] M. Scholz, “Pca - principal component analysis,” 2015. [Online; accessed 20-July-2015].
- [14] L. Wasserman, *All of statistics: a concise course in statistical inference*. Springer Science & Business Media, 2013.

- [15] J. Lee Rodgers and W. A. Nicewander, “Thirteen ways to look at the correlation coefficient,” *The American Statistician*, vol. 42, no. 1, pp. 59–66, 1988.
- [16] P. A. Forum, “Correlation coefficients,” 2015. [Online; accessed 20-July-2015].
- [17] T. H. Cormen, *Introduction to algorithms*. MIT press, 2009.
- [18] CGG MFF UK, *Morphome3cs II*. Charles University in Prague, Czech Republic, 2015.
- [19] Blender Online Community, *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam,
- [20] Wikipedia, “Ground truth Wikipedia, the free encyclopedia,” 2004. [Online; accessed 20-July-2015].
- [21] G. Turk and M. Levoy, “The stanford bunny,” 2005.
- [22] G. Turk and M. Levoy, “The stanford armadillo,” 1996.

List of Tables

- 4.1 The run-time performance of the proposed algorithm over the sphere 31
- 4.2 The run-time performance on Armadillo model. The values in the cells refers to the duration of the phase at the specified heuristic. . 40
- 4.3 The run-time performance on a palate model. The values in the cells refers to the duration of the phase at the specified heuristic. . 41


Appendices

A. User guide

User Interface (UI) design focuses on anticipating what users might need to do and ensuring that the interface has elements that are easy to access, understand, and use to facilitate those actions.

A.1 Usage

The application work as a standard 3D viewer. Before it starts, the command-line environment with the algorithm running is shown. There is shown the logs made by application. The green ones are written by the MeshLoader and the algorithm itself writes a logs in yellow.

A screenshot of a terminal window with a black background and white text. The text shows the execution of a program named 'Morphome3cs Tests'. The logs include: 'Finding tests... 2 found', 'Starting meshload... right after TRY parameter parsing', 'Creating instance.. Created instance..', a green log '[0.012] Loading mesh...' with sub-logs '46499 vertices, 92090 faces' and 'mean edges length: 0.298061', a green log '[0.422] Converting to pointcloud ...', a green log '[0.637] Adding a noise ...', a green log '[0.727] Detecting the features in the mesh ...', a yellow log '[0.000] (<+[0.000]) Grids built', a yellow log '[13.653] (<+[13.653]) Candidates processed', a yellow log '[15.346] (<+[1.693]) Outliers eliminated', a yellow log '[15.375] (<+[0.029]) Feature curves raised', a yellow log '[15.483] (<+[0.108]) Feature curves merged', a yellow log 'counter: 351', a green log '[16.411] Mesh processed ...', and a yellow log 'number of lines=1284'.

```
Morphome3cs Tests
=====
Finding tests... 2 found
Starting meshload...
right after TRY
parameter parsing
Creating instance..
Created instance..
[ 0.012] Loading mesh...
  46499 vertices, 92090 faces
mean edges length: 0.298061
[ 0.422] Converting to pointcloud ...
[ 0.637] Adding a noise ...
[ 0.727] Detecting the features in the mesh ...
[ 0.000] (<+[ 0.000]) Grids built
[ 13.653] (<+[ 13.653]) Candidates processed
[ 15.346] (<+[ 1.693]) Outliers eliminated
[ 15.375] (<+[ 0.029]) Feature curves raised
[ 15.483] (<+[ 0.108]) Feature curves merged
counter: 351
[ 16.411] Mesh processed ...
number of lines=1284
```

Figure A.1: The command-line environment

On initialization of the algorithm, new logger is started thus it also starts a new timeline. On the very left we can see the total time in seconds from the beginning and right after there is a time spent within the algorithm stage.

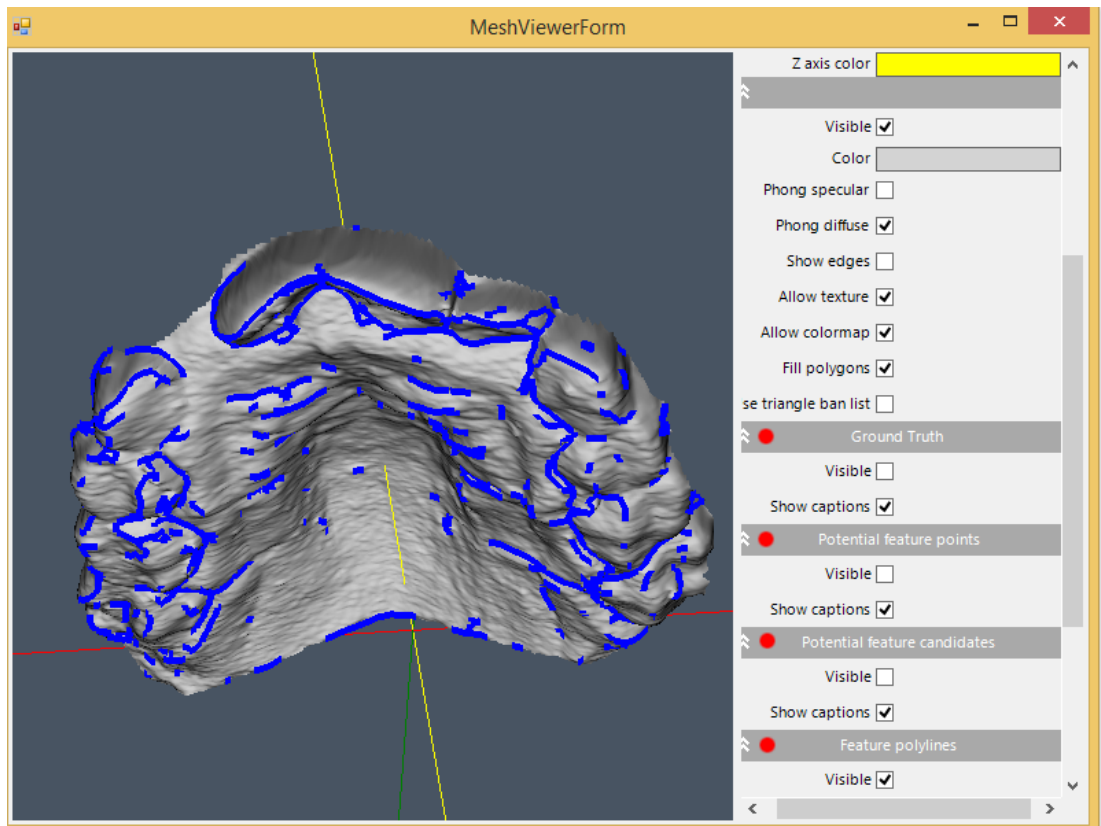


Figure A.2: The algorithm outputs can be shown/hidden using the panel marked with red dot

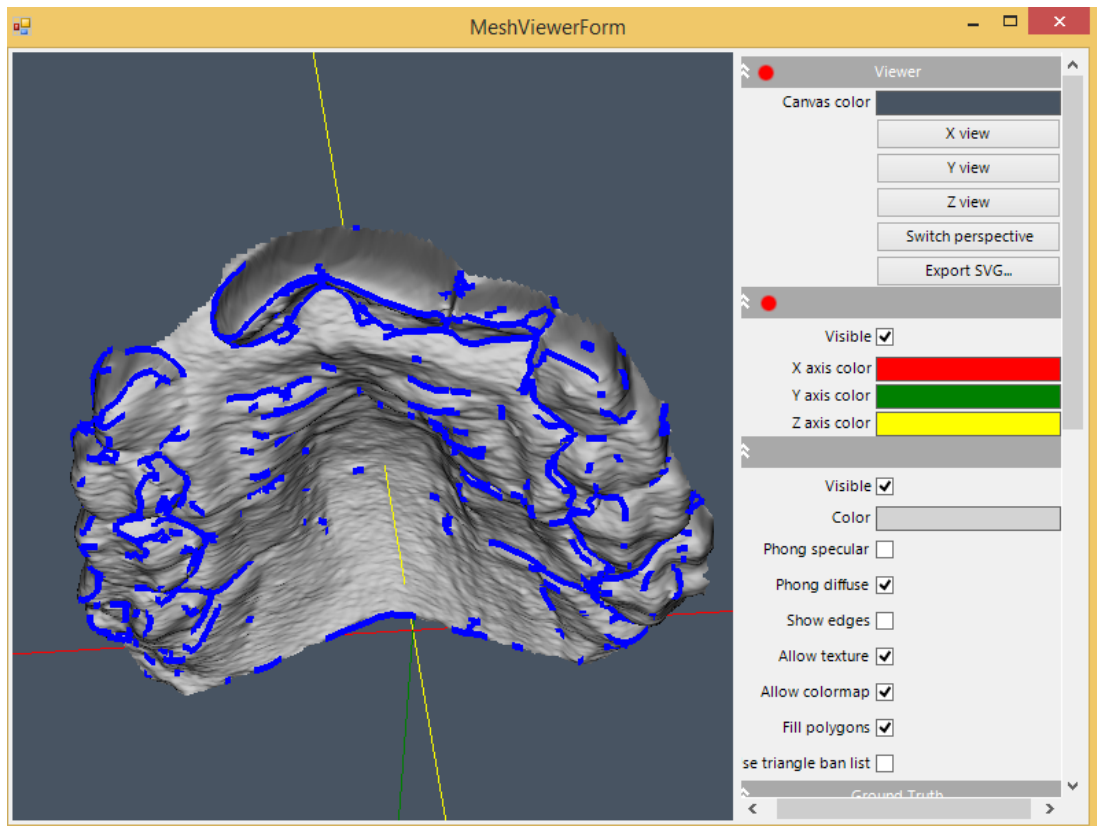


Figure A.3: The environment options as x,y,z axes or the background color can be controlled with the panel on the right top

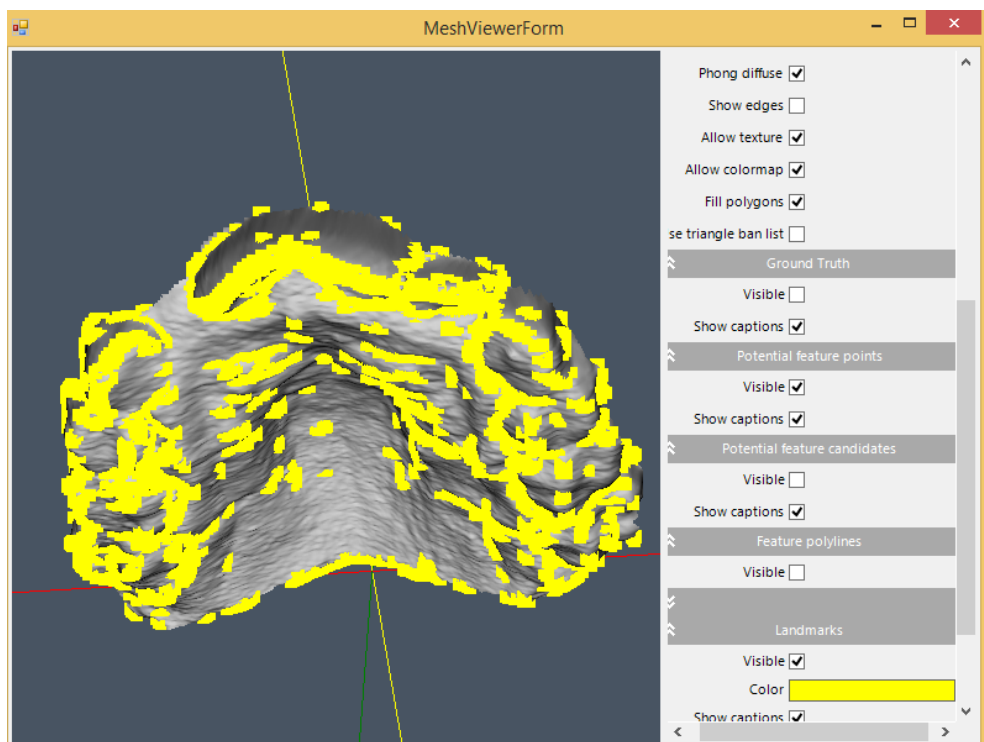


Figure A.4: The user can control the visibility of the results within the stages of the algorithm. On the viewer there are shown the result points after the stage 2

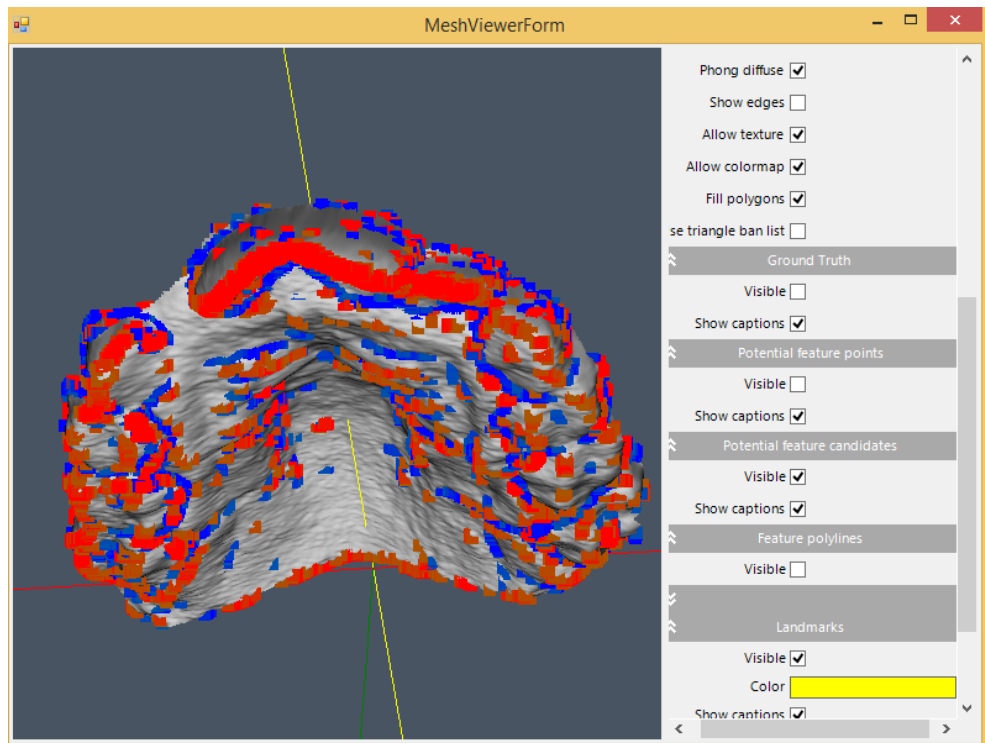


Figure A.5: Ridges and valleys points after the stage 1. There can be seen the presence of the noise since the outlier points has not been yet filtered