

Feature Curves Detection and Segmentation of Point Clouds

Alexander König[1(✉)], Ivana Kolingerová[1], and Tomáš Bayer[2]

[1] Department of Computer Science and Engineering, University of West Bohemia, Pilsen, Czech Republic
{akonig,kolinger}@kiv.zcu.cz

[2] Department of Applied Geoinformatics and Cartography, Charles University, Prague, Czech Republic
bayertom@natur.cuni.cz

Abstract. The data type commonly used to represent geodata are point clouds. They hold most of the important information and require less memory than meshes, however, they are sometimes harder to process. One of the keys to reducing the computational cost of working with them is detecting their most important properties and dividing them into sub-clouds that can be further processed separately. To further work with them efficiently it can be beneficial to estimate the borders of the sub-clouds. They can be formed by feature curves, curves that trace the important features in the data. In this paper we propose a method for segmentation and feature curve detection. Our aim is to work directly with the point cloud, without reconstructing the underlying surface, as surface reconstruction can be computationally expensive, especially for terrain point clouds. The segmentation of the input point cloud into nearly planar sub-clouds is done using Variational shape approximation algorithm, and the feature curves are defined using alpha shape computation and poly-line interpolation. The detected feature curves form borders between the sub-clouds. The proposed method was tested for point clouds representing 3D objects as well as for terrain data.

Keywords: Point Cloud · Point Cloud Segmentation · Feature Curves · Feature Curve Detection · Variational Shape Approximation · Alpha Shapes · Poly-Line Interpolation

1 Introduction

Point clouds are the most common data type used when working namely with geodata. They hold most of the important information about the digitized data, are more compact, and require less memory in comparison to 3D meshes. However, due to their unstructured nature, the analysis and further processing of these data sets are in some cases more challenging than for meshes. Namely, terrain point clouds acquired by scanning real sceneries are typically large, often include non-trivial terrain formations (such as overhangs or cavities), and may

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027
O. Gervasi et al. (Eds.): ICCSA 2026, LNCS 16768, pp. 684–702, 2027.
https://doi.org/10.1007/978-3-032-30488-9_42

contain imperfections, including noise or holes. Processing such data is an important and challenging problem. Often it is needed to subsample them, filter out the noise or fill in the holes. To do this efficiently, it can be useful to divide the point cloud into sub-clouds that can be processed separately, faster or easier, than if we worked with the whole point cloud.

There are many approaches for segmentation of meshes and point clouds, some based on feature curves, curves tracing the important features (such as edges and creases) present in the data. These methods mostly focus on working exclusively with 3D meshes and either require a set of feature curves as input or their first step is feature curve detection. Other approaches are based on detecting geometric objects, such as basic geometric primitives, planes, or surfaces in the 3D data. These methods either do not detect feature curves or the quality of detected feature curves depends on the size of the created sub-clouds.

Segmentation with the condition of planarity divides the input point cloud into a set of distinct almost planar sub-clouds. Each satisfies the condition of being able to be projected onto its best fitting plane (which can be found for example using the principal component analysis approach [13]) without the loss of important information. This simplifies the problem of working with 3D data, as one would work with multiple sets of data in 2D. To correctly work with the projected sub-clouds it is needed to estimate the border of its domain. This can be done by projecting the feature curves that neighbor the sub-cloud onto the same plane. It is beneficial to create as accurate borders as possible. This is a non-trivial task for point clouds, as they do not have firmly set neighborhoods.

This paper proposes a point cloud segmentation method preserving the shape of the point cloud as much as possible and subdividing it to simplify further processing. It creates nearly planar sub-clouds and detects the most important feature curves in the data, each curve represented by a set of vertices. The feature curves are connected and form the borders of the sub-clouds.

Novelty of the proposed method lies in the unique combination of the following properties:

- the algorithm works with the input point cloud without reconstructing the underlying surface, as surface reconstruction can be computationally expensive,
- the feature curves are determined from the created sub-clouds, no preprocessing step is required,
- the algorithm output allows to simplify further processing of the 3D data by projecting it into 2D.

In the following section is the discussion of other approaches to segmentation. In Sect. 3 the details of the proposed algorithm and its individual steps are presented. The results for both point clouds representing 3D objects and terrain point clouds acquired by LIDAR scanning are discussed in Sect. 4.

2 Related Work

As previously mentioned, data in 3D is often represented by meshes or point clouds. To further process both point clouds and meshes, it can be advantageous to divide them into distinct parts or regions. This approach, called segmentation, groups together areas with similar properties. Segmentation methods can be either semantic or geometric.

Semantic segmentation divides the input into regions based on the semantic content of the object's parts. For example, a model of a person is divided into head, torso, arms, and legs. It is often used to find object boundaries in 3D space, which is useful in robotics, autonomous driving, or augmented reality. A review of point cloud semantic segmentation was done by Y. Xie et al. [20] and one example of a method for 3D scans with large point counts is Scancomplete [6]. There are methods focusing on geodata, such as LIDAR point clouds, due to their specific properties, such as the method by Y. Wu et al. [19].

Geometric segmentation is based on dividing the input into regions according to their geometric properties. Geometric segmentation of 3D meshes can be based on feature lines, which are then connected to form patches in the input data. One option is to extract complete feature curve networks using a quadric surface fitting technique and then find cycles in them to generate the segmentation, as proposed by Z. Lu et al. [12]. A different approach, proposed by Y. Cao et al. [4], detects and encloses feature loops on surfaces, the created patches are then subdivided, so they are nearly planar. The approach by H. Baharami et al. [2] subdivides the input mesh into developable ruled surfaces, however, it does not seem to be suitable for detailed data. There are also methods to detect feature curves and edges in point clouds, such as the approach by K.W. Lee and P. Bo [10] or Edc-net [3], which can serve as a starting point of segmentation algorithms.

Geometric segmentation for both point clouds and meshes can also be based on detecting geometric objects, such as basic geometric primitives, planes, or surfaces, in the 3D model. One possible approach is the Variational Shape Approximation (VSA), which was first proposed by D. Cohen-Steiner et al. [5] as a method for segmenting meshes into planar regions. It was later extended to detect different types of approximating surfaces, such as spheres or cylinders [18], and quadric surfaces [21]. It was also extended to work with point clouds [10] and further improved by M. Skrodzkia et al. [16] and T. Zhao et al. [23]. The VSA method generates nearly planar sub-clouds; however, the defined border of one sub-cloud is rather too simplified. K.W. Lee et al. [10] demonstrate that by exploiting the borders of multiple neighboring sub-clouds, feature curves in the model can be found, however the found feature curves do not have to fully enclose any created sub-cloud and their quality depends on the size of the created sub-clouds.

Some approaches to segmentation, such as Exoskeleton [7], combine both the semantic and geometric approaches. Starting with the semantic segmentation of a 3D mesh, this method further decomposes it into patches using a modified VSA algorithm. It has been demonstrated to work for 3D objects, but it may be

challenging to correctly apply semantic segmentation to terrain data, especially due to the atypical properties of the data, such as noise and holes. A survey of segmentation methods specific for terrain data was done by M.D. Hossain [9].

A 3D model can also be segmented by deep learning networks. Parsenet [15] decomposes a point cloud into parametric surface patches and basic geometric primitives. The main advantage is that the results can be refined based on standard CAD modeling operations, however this method has trouble detecting small parts and producing seamless boundaries on noisy data. HPNet [22] segments a point cloud into primitive patches by combining geometric heuristic and deep learning. A common problem of methods based on deep learning is generating realistic training point clouds. Further overview of deep learning methods for segmenting point clouds was done by S. Sarker et al. [14].

In this paper, we propose a method for geometric segmentation of point clouds. It divides the input point cloud into nearly planar sub-clouds and detects feature curves in the input data, each represented by a set of vertices. The detected feature curves form the borders between the sub-clouds. Our algorithm utilizes the VSA approach [16] to segmentation in combination with alpha shapes [8] and poly-line interpolation. The individual steps of the algorithm will be described in the following section.

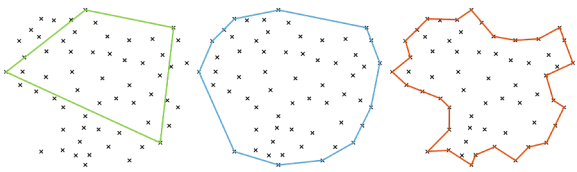


Fig. 1. Illustration of approaches to the definition of the point cloud domain. Using the boundaries defined by VSA in the leftmost image, using convex hull in the middle and alpha shape in the rightmost image.

3 Method Overview

The method proposed in this paper subdivides the input point cloud $P = \{p_i | i = 0, ..., n-1\}$, where each point has the coordinates $p_i = [x_i, y_i, z_i]$. The output is m disjoint nearly planar sub-clouds $R_0 ... R_{m-1}$, $R_i = \{p_{i,j} | j = 0, ..., k-1\}$, where each point has the coordinates $p_{i,j} = [x_{i,j}, y_{i,j}, z_{i,j}]$, each sub-cloud R_i being a sub-set of the points from P. After the segmentation, l feature curves $F_0 ... F_{l-1}$ in the input point cloud are detected and connected to form the borders between the sub-clouds. Each feature curve F_i is represented by a set of vertices, $F_i = \{q_{i,j} | j = 0, ..., s-1\}$, $q_{i,j} = [x_{i,j}, y_{i,j}, z_{i,j}]$.

Let us present the overview of the proposed algorithm. The individual steps of the algorithm, which will be described further, are:

1. Point-cloud segmentation,
2. Crossing point computation,
3. Domain border computation,
4. Poly-line interpolation,
5. Connecting the dots.

The first step is processing the input point cloud to form m nearly planar sub-clouds. In the second step crossing points - the points, where three or more sub-clouds neighbor - are computed. Then the domains of the sub-clouds and their borders are computed. The domain borders of neighboring sub-clouds are interpolated in the fourth step, and in the fifth step they are connected with the crossing points, forming the feature curves of the point cloud.

The sub-clouds R_i computed in step one are the sub-clouds created using Variational shape approximation method [16] and each can be projected onto its best fitting plane without the loss of important information. The feature curves F_i can be found as the border between two neighboring sub-clouds. This is a non-trivial task for point clouds, as they do not have firmly set neighborhoods. To detect the feature curves, this paper utilizes the following approach: first, the domains of the sub-clouds projected onto their best fitting planes are defined. Then the borders of these domains are projected back to 3D, and the borders of two neighboring sub-clouds are interpolated to form the feature curves.

It is important to correctly define the domain and its borders that the sub-cloud R_i occupies when projected onto the plane. The simplest approach would be to take the edges of the polygon bounding the sub-cloud R_i created by VSA and project them onto the best fitting plane of R_i. However, this could cut away parts of the point cloud and create a domain smaller than it really should be, especially since the point cloud could have a more complex shape. A different option would be to use the convex hull of the projected sub-cloud R_i. This could include in the domain parts of the plane that should be excluded as the shape of the point cloud could be concave. Both of these potential problems are illustrated in Fig. 1. Alpha shapes [8] offer a solution that captures the shape of the projected sub-cloud more accurately than a convex hull. They define the shape of a set of points in a plane. How closely the alpha shape copies the point cloud depends on the parameter α. The illustration of an alpha shape can be seen in Fig. 2. Once the alpha shapes for each sub-cloud are found, they are projected back into 3D. Neighboring sub-clouds have different alpha shapes that do not share edges. To find the feature curves in the original point cloud, it is necessary to interpolate between the neighboring parts of the alpha shapes and create poly-line borders between the sub-clouds. These poly-lines represent the feature curves detected in the data and form borders between the created sub-clouds.

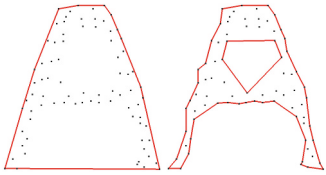


Fig. 2. Convex hull of a point cloud in 2D representing the letter A on the left and an illustration of a possible alpha shape on the right.

The individual steps of the algorithm are described in more detail in the following subsections. The proposed algorithm is also illustrated in Fig. 3.

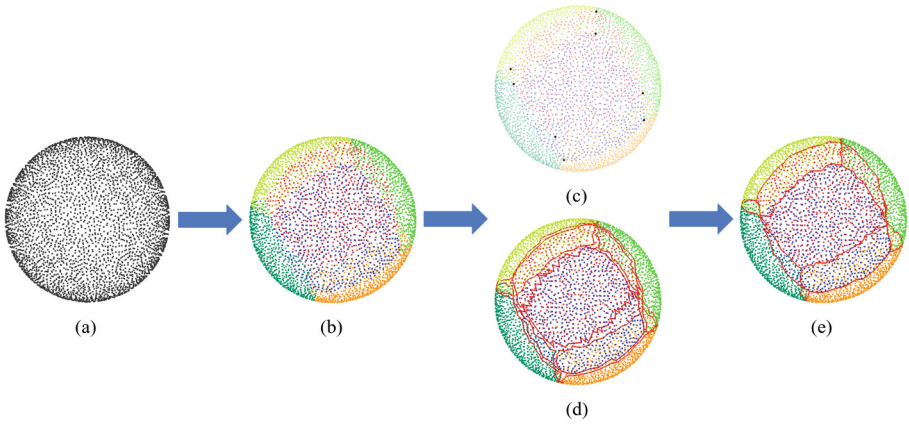


Fig. 3. Illustration of the steps of the algorithm: (a) input point cloud, (b) sub-clouds created by VSA, (c) detected crossing points (thick black), (d) computed domain borders of the sub-clouds (red), (e) resulting feature curves (red). (Color figure online)

3.1 Point-Cloud Segmentation

The first step is to divide the input point cloud into nearly planar sub-clouds. As already mentioned, to partition a point cloud $P = \{p_i | i = 0...n-1\}$, where each point has the coordinates $p_i = [x_i, y_i, z_i]$, into m disjoint sub-clouds $R_0...R_{m-1}$ the VSA algorithm is used. It works as follows: each created sub-cloud R_i can be approximated well by a plane with a normal N_i. This plane can be acquired using the principal component analysis (PCA) approach [13]. These planes are called proxies of the given sub-clouds. The created sub-clouds minimize the following energy $L^{2,1}(R_i, N_i)$ defined by [16]:

$$L^{2,1}(R_i, N_i) = \sum_{p_j \in R_i} w_j ||n_j - N_i||_2^2 \tag{1}$$

where n_j is the normal of the point p_j and w_j is its weight. To compute the normal n_j, the PCA algorithm is used on the point p_j and its neighborhood defined as its k nearest neighbors from P. The main disadvantage of this approach is the need to compute the normals for each of the input points.

The overview of the implemented algorithm VSA is outlined in Algorithm 1 and described in the following text. It is based on the following steps:

- Initialization - given an integer parameter m, m points $p_0, ..., p_{m-1} \in P$ are chosen as the seed points of the m proxies. For each proxy P_i the neighbors of the seed point p_i, paired with the index of the proxy i, are added to a priority queue Q. The priority of each pair in the queue reflects how well the point normal n_j is approximated by the proxy normal N_i.
- Flood - while Q is not empty, a pair [point p_j, proxy index i] is dequeued, and if p_j has not been assigned to a proxy yet, it is assigned to the proxy i and its neighbors are added to the queue paired with the index i.
- Proxy update - the normal of each proxy is updated to the normal of the best fitting plane of the assigned points.
- Seed point update - a new seed point is selected from the assigned points as the point with the closest normal to the new proxy normal. If the number m is known upfront these last three steps are then repeated while assignments to proxies change or until maximal number of iterations is reached.

Controlled Level of Detail. The VSA method can be modified so that the number of desired proxies m does not need to be known upfront. Instead, it changes with relation to a parameter $maxD$ that controls the maximum deviation from its best fitted plane of the subset assigned to proxy P_i.

For this to work, two new operations need to be defined, split and merge, which increase and decrease the number of proxies respectively [16]. They take place after the proxy update step of the algorithm. To avoid the creation of subclouds with low number of points we add a final merge pass merging those with fewer points than threshold t_{pc}. The details of the split and merge operations are described below.

Split. This operation increases the number of proxies. It is performed if in a proxy P_i at least l points are further from the proxy than $maxD$. The largest spread direction of P_i is computed using PCA, and the proxy is then split at the center perpendicularly to its best-fitted plane and the direction projected onto the plane. For each newly created sub-cloud, its new proxy normal is computed using PCA.

Merge. This operation decreases the number of proxies. It is performed if for any subset $P' = P_i \cup P_j$, created by two neighboring subsets, fewer than l points are further from the proxy than $maxD$. Then the two subsets are replaced by their union P'. The new proxy normal of P' is computed using PCA.

Input: $P = \{p_i, i = 1...n-1\}$, $\forall p_i = \{x_i, y_i, z_i\}$ - point cloud; m - number of proxies; k - number of nearest neighbors; $maxD$ - maximum allowed distance of points from best fitted plane
Output: $R_0...R_{m-1}$ - set of distinct point clouds
Initialization();
while *Assignments to proxies change OR unreached maximum number of iterations* **do**
    Flood();
    Proxy update();
    Split();
    Merge();
    Seed point update();
end
Final merge();

Algorithm 1: Point-cloud segmentation based on the VSA method.

3.2 Crossing Points Computation

Now that the sub-clouds have been established, it is needed to compute the crossing points - points where three or more proxy planes meet. This means that they lie on the border of multiple sub-clouds, and polygon meshes can be created from the VSA division by connecting the crossing points.

Both D. Cohen-Steiner et al. [5] and M. Skrodzkia et al. [16] compute a crossing point as the intersection between three or more proxy planes. This could create crossing points rather far from the original presumed surface, namely if the planes have a big difference in their normals, which is quite common for non-smooth surfaces such as terrain models. Therefore we chose a different approach to the detection of crossing points, outlined in Algorithm 2 and described below.

Two proxies neighbor if at least one point from one proxy neighbors with any point from the other. These neighborhoods can be interpreted as a proxy graph, where the vertices represent the proxies and an edge forms if the two proxies neighbor. Using the Bron-Kerbosch algorithm [1] the maximal clique can be found. It is a group of vertices from the graph, where each pair is connected by an edge and there is no additional vertex that could be added while preserving the connectivity condition of the group. In the proxy graph this means that in a clique each pair of proxies neighbors each other. If there is three or more members in the clique, they should create at least one crossing point.

The o crossing points $C_0...C_{o-1}$ in the dataset are determined from the maximal cliques as follows. For each point p in each proxy in each clique cq are found all points in radius r, if p has two or more neighbors, each from a different proxy, p is saved as a candidate point of a group of proxies containing its proxy and the proxies of the neighbors. For each saved group of proxies the candidate points are then averaged to create one crossing point.

There is a possibility that no candidate points will be found for some cliques. This is caused by the difficulty of correctly estimating neighborhoods in point

Input: $P_0 ... P_{m-1}$ - a set of proxies and their assigned distinct point clouds,
r - radius used for neighborhood definition
Output: $C_0 ... C_{o-1}$ - crossing points
cq = maximal clique in graph of $P_0 ... P_{m-1}$;
for *each proxy* P_i *in* cq **do**
  for *point* p *in* P_i **do**
    Find all points closer to p than r and their proxies nn_p;
    if $|nn_p \cap cq| + 1 == |cq|$ **then**
      Save p as candidate crossing point for cq;
    end
    if $|nn_p \cap cq| + 1 > 2$ *and* $|nn_p \cap cq| + 1 < |cq|$ **then**
      Save p as candidate crossing point for $(nn_p \cap cq) \cup P_i$;
    end
  end
end
for *each saved group of candidates* GC_i **do**
  Create a crossing point C_i by averaging points in the group;
end

Algorithm 2: Crossing points computation.

clouds. To compute the crossing point for such a clique, we find a group of m nearest points from these m proxies, where each point is from a different proxy.

Sometimes, crossing points should be created in areas where only two proxies meet. This can happen for terrain data scanned in tiles near the edges of the dataset. In the case that we are processing such data, we need to add more crossing points created from points near the borders of the dataset that have some neighbors from a different proxy. This is described in Algorithm 3.

3.3 Domain Border Computation

The third step of the proposed algorithm is computing the domain borders of the created sub-clouds projected onto their best fitted plane. For this, it is advantageous to use alpha shapes as they approximate the shape of the point cloud well. The alpha shape of a set of points is formed by all edges between two points, where there exists a circle of radius $1/\alpha$ that has the two points on its boundary and does not contain any other points from the set. Alpha shape is a generalization of the convex hull, this means that every convex hull is an alpha shape, but not every alpha shape is a convex hull [8].

To determine the domains, alpha shape is computed for each sub-cloud. The sub-cloud is projected onto its best-fitting plane, then each pair of points is tested against the empty circle condition mentioned in the previous paragraph. If the condition is met, an edge connecting the two points is added to the alpha shape. This outputs the alpha shape as a set of edges. In it we find the longest polygon cycle - the outer part of the alpha shape, given correctly set parameter α. This forms the border of the sub-cloud. The borders of two neighboring sub-clouds will not be identical and will have to be further processed.

Input: $P_0 ... P_m$ - a set of proxies and their assigned distinct point clouds
Output: $C_0 ... C_{r-1}$ - crossing points
for *each proxy* P_i **do**
  for *point* p *in* P_i **do**
    if p *is near the border of the dataset* **then**
      Find k nearest neighbors of p and their proxies nn_p;
      if $|nn_p| + 1 == 2$ **then**
        Save p as candidate crossing point for $nn_p \cup P_i$;
      end
    end
  end
end
for *each saved group of candidates* GC_i **do**
  Create a crossing point C_i by averaging points in the group;
end

Algorithm 3: Crossing points computation on the edges of the dataset.

3.4 Poly-Line Interpolation

To create the feature curves from the detected borders, it is necessary to compute the proxy neighbor of each point on the border a_i. First, the neighboring points from other proxies Q_i in a radius r are found. The point $q \in Q_i$ that is the closest to a_i determines the proxy that is the neighbor of a_i. This way we can establish which part of the alpha shape borders each one of the neighboring proxies.

To estimate the feature curve between two sub-clouds it is needed to interpolate the borders of the two neighboring sub-clouds. As each border is represented by a poly-line, this translates to interpolating two poly-lines L_i and L_j, where each has a different number of points. First, it is needed to label the poly-lines so that L_i contains more points than the poly-line L_j. Then for each point in L_i the closest point from L_j is found. The point on the feature curve is then computed as the average of this pair of points.

Given two sub-clouds P_i and P_j and their respective alpha shapes A_i and A_j, neighboring parts of the alpha shapes can be found by detecting all points from A_i that have P_j as the nearest proxy neighbor. These points are then connected into the shortest sequence along the alpha shape. The same procedure is then computed for all points from A_j that have P_i as the nearest proxy neighbor. This leaves us with two poly-lines that will be interpolated in the next step.

This creates a set of feature curves. Each feature curve lies in between two sub-clouds, but the curves do not connect to each other to form seamless borders between the sub-clouds. This is addressed in the last step of the algorithm.

3.5 Connecting the Dots

The final step of the proposed algorithm is to connect the created feature curves with the detected crossing points, so that they would fully divide the sub-clouds from each other. Each feature curve lies between two sub-clouds. For each of

the two points on the ends of the feature curve find a distinct closest crossing point neighboring the same two sub-clouds.

4 Results

The proposed method was implemented using *C#* and ran on a computer with an *Intel(R) Core(TM) i7-9700K* CPU, 32 GB of memory and *Windows 10* OS. The method was tested on point clouds with various point count and density, acquired from public repositories [11,17], and point clouds we created - *"Sphere"*, *"Cube"* and *"Noisy Sphere 1-3"*. Two types of data were tested - those representing 3D objects, resized to fit into a bounding box of size $1 \times 1 \times 1$ without altering their proportions, and terrain data acquired by LIDAR scanning, resized so the coordinates of the points are in the interval $< 0, 1 >$ in all axes.

The method requires to set the values of the following parameters:

1. For the Point-cloud segmentation step: starting number of proxies m, maximum iterations it, maximum divergence from proxy $maxD$.
2. For alpha shape computation step: value of α for each sub-cloud.
3. For the Crossing points computation and Division into neighboring parts steps: radius r used for proxy neighbor computation.

The parameter m was set to 50 for all inputs. The segmentation algorithm achieved better results in terms of the nearly planar condition than with lower values of m and further increasing it has not shown any further significant improvement. Similarly, the parameter it was set to 100 for all inputs. The parameters $maxD$, the parameter α for each sub-cloud and parameter r for neighbor computation were set empirically for each input.

The implemented VSA algorithm incorporates the merge and split operations which reduce the dependance of the output on the values of the parameters. The most important parameter is $maxD$, the target approximation error threshold, defining how well the resulting sub-clouds adapt to the complexity of the input point cloud. We propose to set $maxD$ as roughly 10% of the size of the bounding box of the data as this provides a good starting point. Higher values of the parameter are advisable for noisy or sparse data, for denser data lower values work better (e.g. the *"Stanford Bunny"* dataset). If the dataset should be approximated by lower amount of proxies, the parameter $maxD$ should be a higher value (e.g. the *"Sphere"* dataset).

Similarly, we propose setting the parameter r as roughly 10% of the size of the bounding box of the data. For sparser data the value of r has to be increased. When it comes to the parameter α for each sub-cloud setting them manually gives us bigger control over the result.

The specific values of the parameters used to obtain the presented results will be discussed along with the analysis of the results in the following subsections. The results are difficult to evaluate using any standard metric, since, as of yet, there is no definition of an ideal curve between two sub-clouds, especially for terrain data. Comparison with other methods also proved difficult, as the most

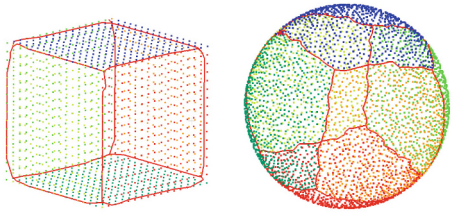
similar works, method proposed by Cao et al. [4] and method proposed by Z. Lu et al. [12], both work with meshes. Therefore the results were evaluated visually by experts in the field working with geodata.

Objects

The proposed method was first tested for point clouds representing 3D objects - namely dataset *"Cube"*, dataset *"Sphere"* and dataset *"Stanford Bunny"*. The results can be seen in Fig. 4 along with the parameters used to achieve them. The parameter α was set to the same value for all created sub-clouds. The created nearly planar sub-clouds are visualized by the different coloring of the points and the created feature curves are visualized as the red poly-lines.

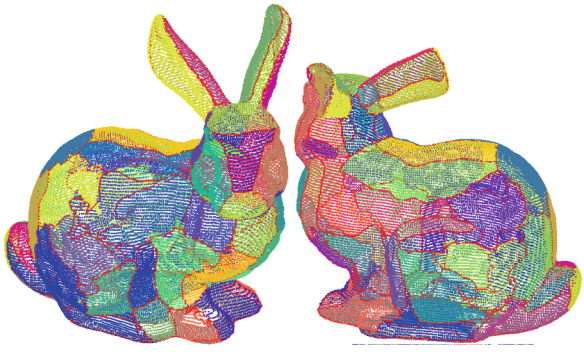
As expected, the *"Cube"* dataset is divided into six sub-clouds, one for each side of the cube with feature curves following the edges of the cube. The *"Sphere"* dataset is also divided into six sub-clouds, similarly to the *"Cube"*.

In Fig. 4 it can also be seen that the detected feature curves for the *"Cube"* dataset do not lie directly on the edges, but are located more towards the center of the point cloud. This is especially noticeable at the crossing points located at the corners of the cube. This is caused by the fact that the crossing points and the feature curves are computed as averages of points from the sub-clouds.



(a) Cube
$maxD = 0.1$,
$r = 0.2$, $\alpha = 0.01$

(b) Sphere
$maxD = 0.2$,
$r = 0.5$, $\alpha = 0.029$



(c) Stanford Bunny
$maxD = 0.03$, $r = 0.03$, $\alpha = 0.029$

Fig. 4. Resulting segmentation of object data.

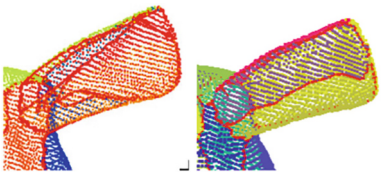
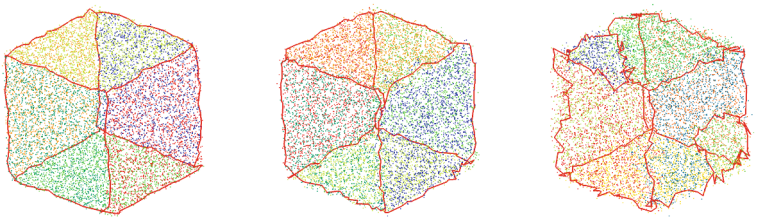


Fig. 5. Stanford Bunny - feature curves going "through" the object ($maxD = 0.05$) on the left, results without the artifact ($maxD = 0.03$) on the right.

For the more elaborate model *"Stanford Bunny"* the value of parameter r was chosen much lower than for the other two datasets. This is because of the higher density of the point cloud. The value for the parameter $maxD$ was also set lower.

The importance of correctly setting the parameter $maxD$ can be demonstrated on the example in Fig. 5. The value of this parameter influences the created sub-clouds, namely their planarity. The lower the value, the more sub-clouds are created, since $maxD$ controls the maximum deviation of the sub-cloud from its best fitted plane. Incorrectly setting this parameter would negatively influence the computation of alpha shapes and in turn the created feature curves. This problem could present itself as feature curves leading "through" the object, for example for parameter $maxD = 0.05$ on the ear of the bunny. If the parameter $maxD$ was decreased to 0.03, this artifact did not appear. In conclusion, if this kind of artifact appears, the value of $maxD$ should be decreased.

The method was also tested for a synthetic datasets with noise. The Fig. 6 presents results for a cube model with varying amounts of noise along with the parameters used to achieve them. The parameter α was set to the same value for all created sub-clouds. The rule for setting the parameters usually is, that the more noisy the dataset is, the higher the value of $maxD$ and α. Figure 6 shows that as the noise grows stronger, the VSA method separates the cube into more sub-clouds that necessary. The feature curves are more "fractured" because they are created as interpolation of points in the data. One of the focuses of our future work will be improving our method so that it is more robust to noise.



(a) Noisy cube 1 $maxD = 0.1$, $r = 0.5$, $\alpha = 0.01$

(b) Noisy cube 2 $maxD = 0.2$, $r = 0.5$, $\alpha = 0.01$

(c) Noisy cube 3 $maxD = 0.2$, $r = 0.5$, $\alpha = 0.02$

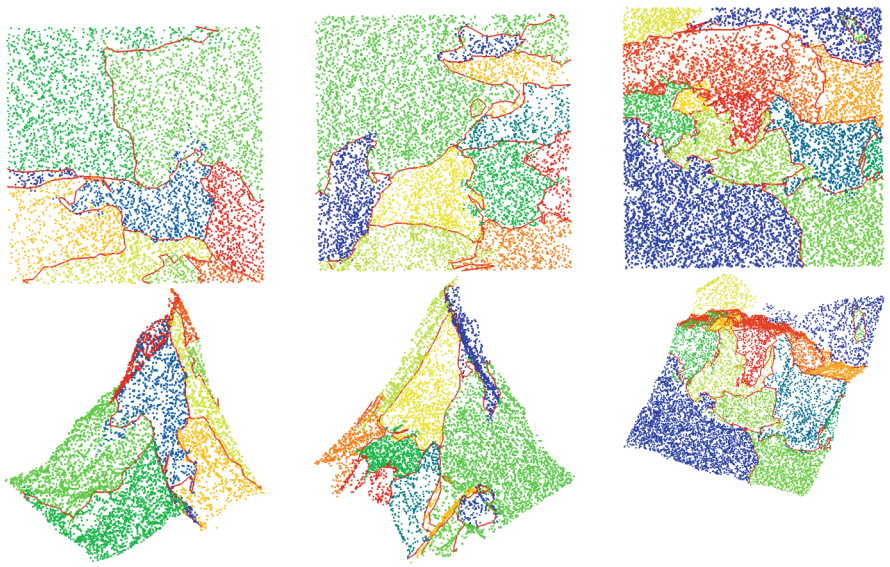
Fig. 6. Resulting segmentation of noisy data.

Terrain Data

The proposed method was also tested for terrain point clouds acquired by LIDAR scanning - namely the Mount Saint Helens dataset, tiles *"Mount Saint Helens NW"*, *"Mount Saint Helens NE"* and *"Spirit Lake West SW"*. The results and the used parameters are in Fig. 7. The created VSA sub-clouds are visualized by the different coloring of points and the created feature curves as the red colored poly-lines. The parameter α for each sub-cloud was chosen empirically.

Looking at the results for *"Mount Saint Helens NW"* (Fig. 7(a)), it can be seen that the method is able to successfully divide terrain data into nearly planar sub-clouds, and detect the most important feature curves that form the borders between the sub-clouds. The number of created sub-clouds depends on the value of the parameter $minD$, similarly to for the object datasets.

Terrain data has some properties, such as noise, varying point densities and more elaborate changes in the underlying surface, which can cause some specific problems. First, most noticeable and more common complication, is the existence of "island" sub-clouds - sub-clouds that neighbor only with one other sub-cloud. This can be seen in both *"Mount Saint Helens NE"* (Fig. 7(b)) and *"Spirit Lake West SW"* (Fig. 7(c)). These sub-clouds cannot be processed using the same approach as described in Sect. 3.5. Instead, the whole alpha shape is proclaimed as a feature curve around the sub-cloud.



(a) Mount Saint Helens NW $maxD = 0.08$, $r = 0.04$ (b) Mount Saint Helens NE $maxD = 0.08$, $r = 0.03$ (c) Spirit Lake West SW $maxD = 0.08$, $r = 0.03$

Fig. 7. Resulting segmentation of terrain data.

In more rare cases, if the configuration of points is such that there is a larger hole in a place where two or more sub-clouds should meet, the points of those sub-clouds do not neighbor each other. This problem corresponds to the problem of neighborhood detection in point clouds. This complication can cause two types of artifacts - missing feature curves and missing crossing points. Both can be seen in the dataset *"Spirit Lake West SW"* in Fig. 7(c) and in more detail in Fig. 8. The missing crossing point artifact (in Fig. 8 in the upper rectangle) is in a place, where crossing point would be expected, but is not detected by the algorithm in the Crossing points computation step, as the two proxies do not neighbor each other near the border due to the hole in the data. The missing feature curves artifact (in Fig. 8 in the lower rectangle) was caused by incorrect proxy neighborhood assignment in the Division into neighboring parts step.

Execution Time Analysis

This subsection discusses the time needed for the execution of the algorithm for the previously discussed input data. The algorithm steps mentioned in this subsection correspond to the steps described in Sect. 3. The measured times can be seen in Table 1 and are visualized in the graph in Fig. 9.

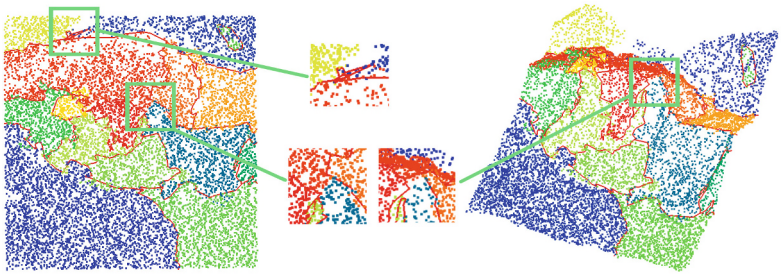


Fig. 8. Missing crossing point artifact and missing border artifact in the dataset *"Spirit Lake West SW"*.

For most of the inputs the most time consuming step is Step 1 - Point cloud segmentation. Its is implemented using the VSA algorithm, where the parameter that influences the results and time needed for computation the most is the parameter $maxD$. Lowering its value causes the creation of a higher number of sub-clouds. This means the method will be more time consuming, as it causes more strict requirements on the planarity of the created sub-clouds and causes more splits and merges. Another property causing the VSA to increase in time requirements is the complexity of the data. The more complex the dataset is, the more sub-clouds will be created and more splits and merges will be done during the computation. The more noisy the data, the more time is proportionally spent in the Step 1, as is demonstrated for the inputs *"Noisy cube 2"* and *"Noisy*

cube 3". Terrain datasets are also generally more complex then point clouds representing objects and therefore for them the VSA algorithm will be more time consuming. Naturally, it is possible to incorporate approaches to increase the speed of the VSA algorithm, such as parallelization.

The time needed for the computation of Step 2 (Crossing point computation), 4 and 5 (Poly-line interpolation and Connecting the dots) is influenced by the density of the data and the size of the radius r used during neighborhood computation. More neighbors for a point mean longer computation time. The results for most of the object datasets have been acquired using a larger radius, and therefore more time is proportionally spend in the Step 2. The influence of the radius size is also very visible for the object datasets for the Steps 4 and 5. The *"Stanford Bunny"* dataset has a smaller radius of $r = 0.03$ and proportionally spends much less time in the Steps 4 and 5. This is also the reason why the terrain datasets spend the least time in those steps.

Since it is needed to correctly set the parameters to get the best possible output, the method (or more often just some of its steps) have to be re-run multiple times before the correct value of the parameters are found. The most often repeated step is Step 3 - Domain border computation. Fortunately, it is usually the fastest, especially if we take into consideration the fact that the times presented in Table 1 are for the computation of the domains for all of the subclouds, computing one domain is faster.

Table 1. Time needed for computation.

Dataset	Point count	Step 1 [s]	Step 2 [s]	Step 3 [s]	Step 4 + 5 [s]
Cube	1 538	1.854	5.229	0.687	5.160
Sphere	3 842	2.684	15.967	0.457	16.243
Mount Saint Helens NW	6 324	12.44	0.045	1.091	0.094
Mount Saint Helens NE	8 440	27.900	0.048	1.777	0.232
Noisy cube 1	10 000	3.182	79.544	3.180	31.105
Noisy cube 2	10 000	3.329	84.166	2.991	29.084
Noisy cube 3	10 000	37.335	27.201	1.389	28.140
Spirit Lake West SW	11 993	36.494	12.195	2.326	0.074
Stanford Bunny	34 834	224.475	272.800	7.804	0.521

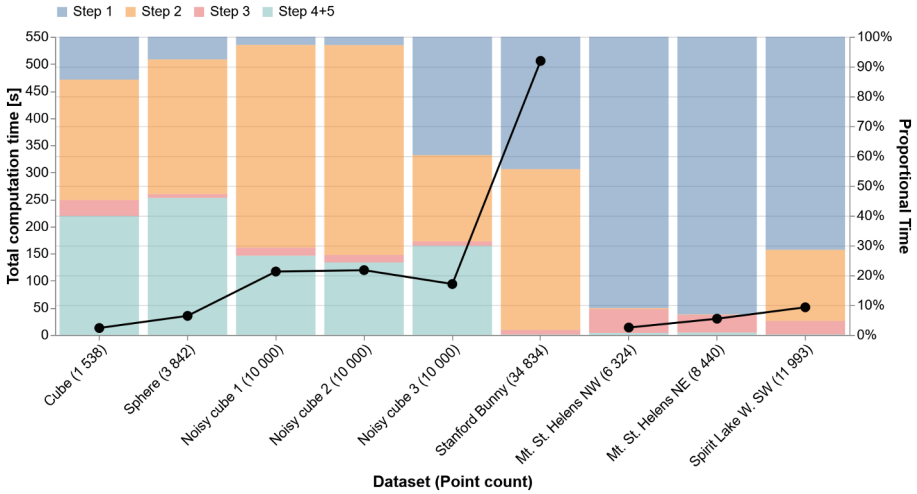


Fig. 9. Graph of the times needed for computation of the presented results. Objects are grouped together on the left and terrain datasets on the right.

5 Conclusion

This paper proposes a method for point cloud segmentation that captures the high level geometric structure of the point cloud. The method works with the input point cloud without reconstructing the underlying surface and the feature curves are determined from the created sub-clouds.

The created nearly planar sub-clouds are computed using the VSA algorithm modified for point clouds. Each is a sub-set of points from the input point cloud, where no two sub-sets share points. The method detects the most important feature curves in the data, each represented by a set of vertices. First the border of the domain of each sub-cloud projected onto its best fitted plane is obtained by computing the alpha shape, then the alpha shape is divided into poly-lines that are finally interpolated with the poly-lines of neighboring sub-clouds.

The main motivation behind this work was to create a method for terrain data, which was successful, and as demonstrated, the proposed approach works for point clouds representing 3D objects as well. The output of the algorithm simplifies the problem of working with the 3D point cloud data, as one would work with multiple sets of data in 2D. The feature lines projected into 2D form the borders of the sub-clouds and can be trivially improved using standard 2D techniques to remove possible self-intersections and form simple polygons.

The main downside of the method is the difficulty of establishing neighborhoods in point clouds and the need to manually set parameters of the method. In the future, we aim to improve it to be more robust to the neighborhood establishment problem, so it can work with more noisy input. Based on longer cooperation with experts on working with geodata, we will focus on developing a reliable automatic method for parameter setting, so that the results will

still correctly reflect the realities of the data. Further improvements could also include the smoothing of the feature curves and more focus on increasing the speed of the algorithm, namely the Point cloud segmentation step, which was demonstrated to be the most time consuming for the terrain data.

Acknowledgments. This work was supported by a specific research project SGS-2025-022 New Data Processing Methods in Current Areas of Computer Science of the University of West Bohemia.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this paper.

References

1. Aloysius, D.R.: Bron-Kerbosch Algorithm. PopulPublishing (2012)
2. Baharami, H., Piovarci, M., Tarini, M., Bickel, B., Pietroni, N.: Fabricable discretized ruled surfaces. ACM Trans. Graph. **44**(3), 1–15 (2025). https://doi.org/10.1145/3734519
3. Bazazian, D., Parés, M.E.: EDC-net: edge detection capsule network for 3D point clouds. Appl. Sci. **11**(4), 1833 (2021). https://doi.org/10.3390/app11041833
4. Cao, Y., Yan, D.M., Wonka, P.: Patch layout generation by detecting feature networks. Comput. Graph. **46**, 275–282 (2015). https://doi.org/10.1016/j.cag.2014.09.022
5. Cohen-Steiner, D., Alliez, P., Desbrun, M.: Variational shape approximation. ACM Trans. Graph. (2004). https://doi.org/10.1145/1015706.1015817
6. Dai, A., Ritchie, D., Bokeloh, M., Reed, S., Sturm, J., Nießner, M.: Scancomplete: large-scale scene completion and semantic segmentation for 3D scans. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (2018). https://doi.org/10.1109/CVPR.2018.00481
7. de Goes, F., Goldenstein, S., Desbrun, M., Velho, L.: Exoskeleton: curve network abstraction for 3D shapes. Comput. Graph. (2011). https://doi.org/10.1016/j.cag.2010.11.012
8. Edelsbrunner, H.: Tessellations in the Sciences: Virtues, Techniques and Applications of Geometric Tilings. Springer (2011)
9. Hossain, M.D., Chen, D.: Segmentation for object-based image analysis (OBIA): a review of algorithms and challenges from remote sensing perspective. ISPRS J. Photogram. Remote Sens. **150**, 115–134 (2019). https://doi.org/10.1016/j.isprsjprs.2019.02.009
10. Lee, K.W., Bo, P.: Feature curve extraction from point clouds via developable strip intersection. J. Comput. Des. Eng. (2016). https://doi.org/10.1016/J.JCDE.2015.07.001
11. Levoy, M., Gerth, J., Curless, B., Pull, K.: The Stanford 3D scanning repository (2005). https://graphics.stanford.edu/data/3Dscanrep
12. Lu, Z., Guo, J., Xiao, J., Wang, Y., Zhang, X., Yan, D.: Extracting cycle-aware feature curve networks from 3D models. Comput.-Aided Des. (2021). https://doi.org/10.1016/j.CAD.2020.102949
13. Salem, N., Hussein, S.: Data dimensional reduction and principal components analysis. Procedia Comput. Sci. **163**, 292–299 (2019). https://doi.org/10.1016/j.procs.2019.12.111

14. Sarker, S., et al.: A comprehensive overview of deep learning techniques for 3D point cloud classification and semantic segmentation. Mach. Vis. Appl. **35**(4) (2024). https://doi.org/10.1007/s00138-024-01543-1
15. Sharma, G., Liu, D., Maji, S., Kalogerakis, E., Chaudhuri, S., Měch, R.: ParSeNet: a parametric surface fitting network for 3D point clouds. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J.-M. (eds.) ECCV 2020. LNCS, vol. 12352, pp. 261–276. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58571-6_16
16. Skrodzkia, M., Zimmermannb, E., Polthier, K.: Variational shape approximation of point set surfaces. Comput. Aided Geom. Des. (2020). https://doi.org/10.1016/j.CAGD.2020.101875
17. University of Washington Libraries: Mount saint helens - lidar data (2003). https://wagda.lib.washington.edu/data/type/elevation/lidar/st_helens/toutle03.html
18. Wu, J., Kobbelt, L.: Structure recovery via hybrid variational surface approximation. Comput. Graph. Forum (2005). https://doi.org/10.1111/J.1467-8659.2005.00852.X
19. Wu, Y., Liu, J., Gong, M., Miao, Q., Ma, W.: Joint Semantic Segmentation using representations of Lidar Point Clouds and camera images (2023). https://doi.org/10.2139/ssrn.4426132
20. Xie, Y., Tian, J., Zhu, X.X.: Linking points with labels in 3D: a review of point cloud semantic segmentation. IEEE Geosci. Remote Sens. Mag. **8**(4), 38–59 (2020). https://doi.org/10.1109/mgrs.2019.2937630
21. Yan, D.-M., Liu, Y., Wang, W.: Quadric surface extraction by variational shape approximation. In: Kim, M.-S., Shimada, K. (eds.) GMP 2006. LNCS, vol. 4077, pp. 73–86. Springer, Heidelberg (2006). https://doi.org/10.1007/11802914_6
22. Yan, S., Yang, Z., Ma, C., Huang, H., Vouga, E., Huang, Q.: Hpnet: deep primitive segmentation using hybrid representations. In: 2021 IEEE/CVF International Conference on Computer Vision (ICCV), pp. 2733–2742 (2021). https://doi.org/10.1109/iccv48922.2021.00275
23. Zhao, T., Busé, L., Cohen-Steiner, D., Boubekeur, T., Thiery, J.M., Alliez, P.: Variational shape reconstruction via quadric error metrics. Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Proceedings, pp. 1–10 (2023). https://doi.org/10.1145/3588432.3591529