PointCloudSlicer: Gesture-based segmentation of point clouds

Hari Hara Gowtham[1,2], Amal Dev Parakkat[1], Marie-Paule Cani[2]

[1] LTCI - Telecom Paris, [2] LIX - Ecole Polytechnique
Institut Polytechnquie de Paris, France

Abstract

Segmentation is a fundamental problem in point-cloud processing, addressing points classification into consistent regions, the criteria for consistency being based on the application. In this paper, we introduce a simple, interactive framework enabling the user to quickly segment a point cloud in a few cutting gestures in a perceptually consistent way. As the user perceives the limit of a shape part, they draw a simple separation stroke over the current 2D view. The point cloud is then segmented without needing any intermediate meshing step. Technically, we find an optimal, perceptually consistent cutting plane constrained by user stroke and use it for segmentation while automatically restricting the extent of the cut to the closest shape part from the current viewpoint. This enables users to effortlessly segment complex point clouds from an arbitrary viewpoint with the possibility of handling self-occlusions.

CCS Concepts

• ***Human-centered computing*** → ***Interaction design;*** • ***Computing methodologies*** → ***Shape modeling;***

1. Introduction and related work

The spread of 3D scanning devices such as LiDAR systems and recent advances in machine/deep learning techniques that make their direct manipulation possible [GWH*20] increased the popularity of point clouds, which are now used in many applications, from cultural heritage to robotics and automatization. In this context, segmenting the input points, i.e., partitioning them into distinct, semantically meaningful regions or clusters, is a fundamental problem essential for scene understanding and editing, object detection, and robot navigation.

Considering that the related mesh segmentation problem is a well-explored area in Computer Graphics [Sha08], where the input is a mesh rather than a point cloud, a trivial solution would be to create a mesh from the point cloud and then reuse any mesh segmentation algorithm. Mesh segmentation can be made simpler and more accurate by incorporating user inputs (e.g. through a simple sketching paradigm) that encode user perception and guide segmentation [JLCW06]. Though sketches improve the quality of mesh segmentation, creating a 3D mesh from point clouds is challenging, especially since the point clouds can have various artefacts like incomplete data, outliers, and heavily varying density.

Many methods, therefore, addressed the direct segmentation of point clouds. They can be broadly classified into two categories: learning-based and non-learning-based methods. The learning-based methods use machine/deep learning algorithms to learn patterns and features from annotated point clouds and predict the classification (see [GWH*20] for a survey). The works under this category span from using traditional machine learning approaches such as K-means and spectral clustering [ZCD20] to more sophisticated methods using CNNs [QSMG17, WSL*19], 2D convolution on projected 3D data [LDT*17], and creating unique structures like Superpoint graph or GCNs [LS18, LHW21]. Non-learning-based methods, on the other hand, such as region-based [VTHLB15] and graph-based methods [NL13], do not require annotated data since they rely on explicit hand-crafted features and algorithms to partition the point cloud. Unfortunately, the latter does not always meet the user intent.

The main objective of this work is to allow interactive point-cloud segmentation as in [SRS*19], but based on simple, intuitive slicing gestures defined over any 2D view of the point cloud. This enables us to benefit from powerful user perception while allowing the use of any segmentation criteria.

Our current preliminary solution is limited to straight slicing gestures, depicted as line strokes. Each of them is interpreted in 3D as a perceptual slicing plane, which is optimized to better match the local arrangement of 3D points. This plane is then used to partition the point cloud into two consistent sub-parts while robustly handling the case of multiple self-occlusions of the point cloud from the slicing viewpoint. To achieve this, we focus on two main challenges:

- How to adjust the cutting plane without worrying about aligning the point cloud to make an optimal cut?
- How to stop the cutting process from blindly segmenting the whole point cloud based only on the relative position and instead intelligently segment only the intended region of the point cloud?

DOI: 10.2312/egs.20231008

The proposed method facilitates quick "drawing over point clouds," allowing the user to define the segments wherever they want without requiring additional information (such as normal or texture) or complex and time-consuming mesh reconstruction.

2. Methodology

PointCloudSlicer works as follows: the user iteratively uses slicing gestures over the point cloud to define the two end-points of a line stroke (see Figure 2); for each such stroke, our algorithm starts by creating a cutting plane, initialized from the viewing angle and the line segment. Since no notion of intersection exists, this plane can intersect with multiple regions of our input point cloud. To avoid this, local clustering is applied on the points near the cutting plane to identify the intended region. We further refine the latter using an optimized, tilted plane to ignore the effect of viewpoint, which directly impacts the cutting plane. Finally, we use local geometrical cues from the optimized region to apply the relevant, localized segmentation to the point cloud. Our technical contributions are, therefore, two-fold:

- An optimization-based framework that uses a width measure depending on the local point distribution to define the optimal cutting plane.
- Using a local Delaunay-triangulation-based method as a geometric cue for identifying the point cluster that best matches user perception.

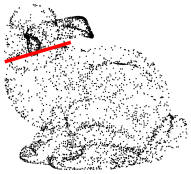

Figure 1 compares the results of a naive segmentation (based on the relative orientation of points w.r.to the initial plane) with those of PointCloudSlicer, given the simple slicing gesture in the inset (the red line shows the user cut). Note that the point cloud was arbitrarily aligned, making the naive segmentation (based on the initial plane parallel to the viewing direction) fail to segment the shape as intended (into points belonging to the head and the body) by having wrongly segmented points on the ears and making a titled cut on the head. Thanks to the underlying intelligent optimization, PointCloudSlicer was able to capture the expected partition. The remainder of this section details the main steps of our method.

2.1. Intelligent clustering

Given the initial slicing plane defined to include the user line stroke and the viewing direction, we first identify the region of the point cloud where the segmentation takes place. Therefore, we define the slice of interest as the set of the points that lie at a smaller distance than a predefined threshold δ to the cutting plane.

Since there is no available connectivity information, this slice blindly extends to the whole point cloud, in depth as well as sideways. To only select the perceptually relevant part, i.e. the closest to the user's stroke, we do a local reconstruction. Inspired by the literature [OPP*21], we start with computing the 3D Delaunay triangulation (which is proven to be useful for curve/surface reconstruction from point clouds since it can create non-overlapping triangles that represent the underlying shape of the point set, while also minimizing distortions and other artefacts) of the points lying inside the slice, and do a simple pruning on its edges based on their length (we remove all the edges longer than the average edge length in the Delaunay triangulation). Please note that this simple strategy could easily be replaced by any sophisticated algorithm, such as in [OPP*21]. The pruning process segments the graph of connected points into different connected components or clusters, enabling us to pick the closest cluster to the centre of the line stroke to define the cut.

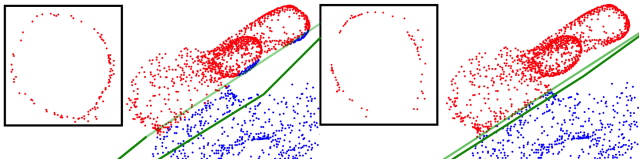

Figure 1: *Segmentation results from the user input. Left: based on a naive cutting plane, Right: using PointCloudSlicer. Cutting planes are shown in green, and points that participated in width measure computation are shown in the inset (values of M are 16.3199 and 11.4438, respectively).*

2.2. Cutting plane optimization

As shown in Figure 1, the cutting plane may need to be tilted to better match the local geometry of the shape and get the expected segmentation. To do this, we associate a width measure M to the cutting plane CP, defined as follows:

$$M(CP) = \sum_{i=1,\ldots,k} d(p_i, A)$$

Where $p_0, \ldots, p_k$ are the points belonging to the selected cluster, A is the average of the points in the cluster, and $d()$ is the Euclidean distance.

Intuitively, while still matching the user's stroke, minimizing the width measure helps us identify the region with minimal girth (which acts as a good perceptual measure for segmentation). Satisfying the constraint that the plane passes through the user-defined line segment, we minimize the width measure by tilting the cutting plane. This effect is salient near the neck region in Figure 1. At the end of this optimization, we get an "Optimal cluster", which can be used to segment the point cloud. In this preliminary implementation, we use greedy optimization in a binary search fashion, where the maximum and minimum possible tilt is set to $\pm 15^{\circ}$.

2.3. Segmenting the point cloud

The point cloud is finally segmented based on the optimal cutting plane while restricting the process to the region corresponding to the selected point cluster. Again, since there is no available structure other than plain coordinate information, direct segmentation is non-trivial. Therefore, we rely on the minimal spanning tree to achieve this segmentation.

We start by computing a minimal spanning tree (MST) of the point cloud to create an initial structure (as a pre-processing step

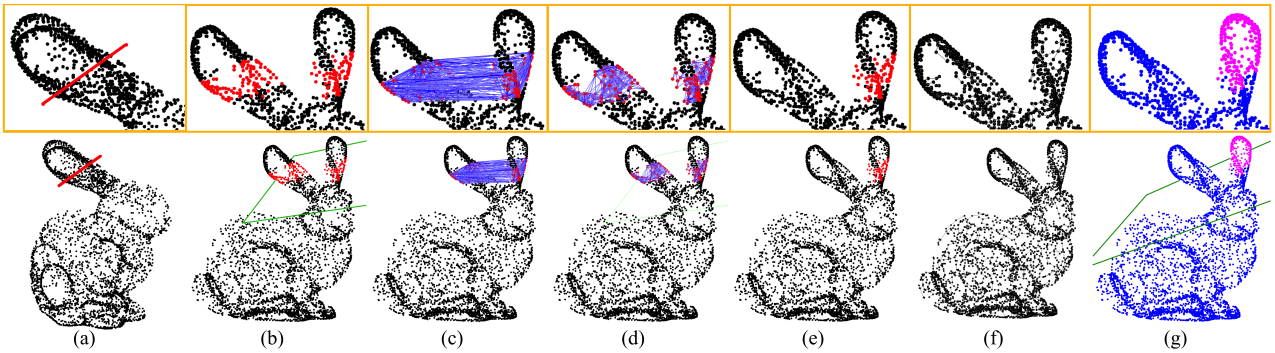

Figure 2: *Left to Right: (a) Input point cloud with the user cut in red color, (b) Points lying inside the slice (in red color) and the naive plane in green color, (c) Delaunay triangulation of points inside the slice, (d) Result after pruning, (e) Points in the selected cluster (in red color), (f) Minimal spanning tree used to bipartite the point cloud, (g) Final segmentation with the optimized plane in green color*

computed only once). The edges of the MST that belong to the optimal cluster and cross the optimal cutting plane are identified and removed. This removal of edges splits the MST into several disconnected components. We then segment the input point cloud based on the relative position of the points (the side of the plane on which it lies) belonging to the optimal cluster and their presence in a connected component. Here, we assume the MST captures the overall structure nicely so that the points in a connected component belong to a single perceptual bi-partition.

The segmentation process is given in Algorithm 1, Where Relative_Position(Point, Plane) computes the relative position (left or right) of the point w.r.to the plane, and Compute_Connected_Components(Graph) computes all connected components present in the Graph.

Putting it all together: The full pipeline of our method is shown in Figure 2. Since each user's slicing gesture only segments the cloud into two parts, this process has to be repeated until the cloud is fully segmented. To ease this iterative process, we label and then discard the smallest of these parts in terms of the number of points to keep only the larger region which may require further segmentation. This enables us to assign segment labels to the points, in the order of the segmentation steps.

Algorithm 1 Segmenting the Point Cloud

procedure SEGMENT(Point cloud P, Optimal cluster C, Optimal cutting plane OCP)
  MST = Minimal_Spanning_Tree(P)
  for each tuple (p_i, p_j): $p_i \in C, p_j \in C$ **do**
    if (p_i, p_j) is an edge in MST **then**
      pos1=Relative_Position(p_i, OCP)
      pos2=Relative_Position(p_j, OCP)
      if pos1 $\neq$ pos2 **then** MST=MST - (p_i, p_j)
  CC = Compute_Connected_Components(MST)
  for each component $con \in CC$ **do**
    if $p_i \in C$ and $p_i \in con$ **then**
      Label(p_i)=Relative_Position(p_i, OCP)
  return labelled points

3. Results & Discussion

We implemented PointCloudSlicer in C++ using the libIGL library and tested it on an 8-core MacBook M1 Pro with 16GB memory. Figure 3 shows a few results generated using PointCloudSlicer in less than 3 minutes (including all the computations). As can be seen, our method could directly segment the point clouds without worrying about the occlusions and could generate visually pleasing

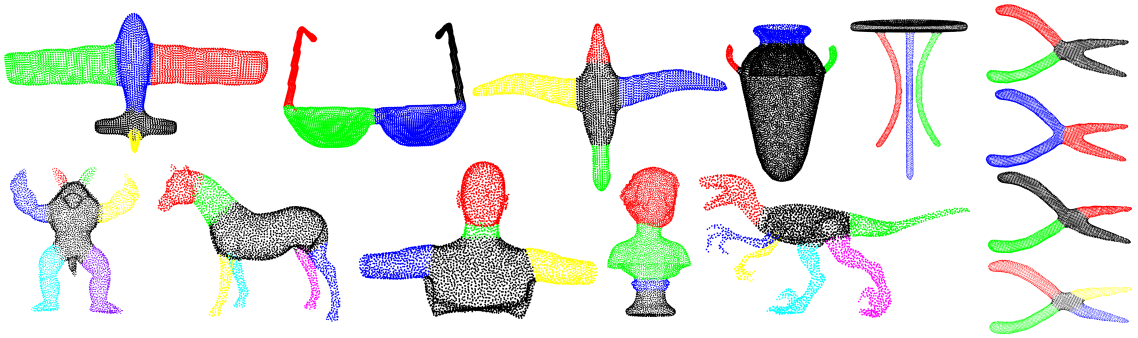

Figure 3: *Point clouds segmented using PointCloudSlicer. Courtesy: aim@shape and Kalogerakis et al. [KHS]*

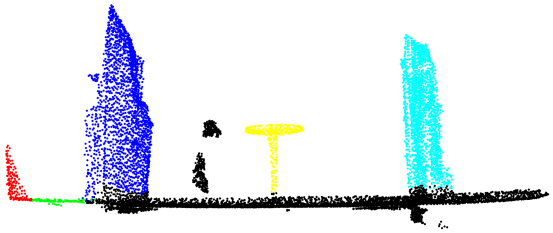

Figure 4: *A simplified RGB-scan segmented using our method*

results. Note that naively using a cutting plane to segment the point clouds (without our algorithm) would have been a failure in most of the examples shown in Figure 3 since, in most cases, we cannot draw a cutting plane without intersecting other parts of the model.

Thanks to our gesture-based inputs, the user may freely decide where to segment a shape to best fit their needs and perception. In contrast with former rule-based and knowledge-based segmentation methods, as shown in the inset on the right, the user could segment the point cloud to separate semantically meaningful parts without the presence of any sharp feature. Similarly, Pliers in Figure 3 show various segmentation options that can be generated from the same point cloud.

Finally, Figure 4 shows the result of PointCloudSlicer on a simple downsampled RGB-D scan taken from http://redwood-data.org/3dscan/index.html. As shown, our simple method could segment the scene satisfactorily.

4. Limitations & Future work

We have presented a simple and efficient point cloud segmentation method, which does not require any intermediate meshing scheme (saving us from possible errors related to meshing) and keeps the user in the loop, enabling any perceptual criteria to be used. The user interacts using straight-through slicing gestures, interpreted into a cutting plane, which is optimized and used to segment a relevant subpart of the point cloud.

Though the proposed method can achieve good visual results, PointCloudSlicer has three main shortcomings, which we aim to address in future work:

- Choice of planar cutting planes: Our method uses cutting planes to segment point clouds, which might not always be the right choice since the boundary of a perceptually salient region might be curved. However, this cutting plane could act as a base from which a better segmentation could be computed. In future, we intend to borrow the idea of snakes to curve the boundary defined by our cutting plane.
- Point clouds with genus>0: The segmentation process assumes that the input point cloud represents a genus=0 surface, meaning that each slicing gesture segments the current cloud into two parts. The method could be extended to surfaces with genus>0 by intelligently propagating labels and relating points to multiple cutting planes.
- Assumptions on MST: We assume the point cloud is densely sampled to avoid any connected component from having points that belong to the optimal cluster that lies on both sides of the cutting plane.

The current implementation of the PointCloudSlicer uses structures that require heavy computation (especially the pre-processing for computing MST, which took around 2 minutes for 20k points), restricting us from testing on point clouds having more than 20k points in interactive rates. So, in addition to the aforementioned algorithmic improvements, we plan to improve/optimize the underlying structures to further deploy and test our method for the interactive segmentation of complex scenes/scans, which we initiated with the example in Figure 4.

Acknowledgements

We thank the reviewers for their constructive comments. A part of this work is funded by the fellowship "Creative AI" from the institute Hi! Paris.

References

[GWH*20] GUO Y., WANG H., HU Q., LIU H., LIU L., BENNAMOUN M.: Deep learning for 3d point clouds: A survey. *IEEE TPAMI 43*, 12 (2020). 1

[JLCW06] JI Z., LIU L., CHEN Z., WANG G.: Easy mesh cutting. In *Computer Graphics Forum* (2006), vol. 25, Wiley Online Library. 1

[KHS] KALOGERAKIS E., HERTZMANN A., SINGH K.: Learning 3d mesh segmentation and labeling. In *ACM SIGGRAPH 2010 papers*. 3

[LDT*17] LAWIN F. J., DANELLJAN M., TOSTEBERG P., BHAT G., KHAN F. S., FELSBERG M.: Deep projective 3d semantic segmentation. In *CAIP* (2017), Springer. 1

[LHW21] LIN Z.-H., HUANG S.-Y., WANG Y.-C. F.: Learning of 3d graph convolution networks for point cloud analysis. *IEEE TPAMI 44*, 8 (2021). 1

[LS18] LANDRIEU L., SIMONOVSKY M.: Large-scale point cloud semantic segmentation with superpoint graphs. In *Proceedings of the IEEE CVPR* (2018). 1

[NL13] NGUYEN A., LE B.: 3d point cloud segmentation: A survey. In *2013 6th IEEE ICARM* (2013). 1

[OPP*21] OHRHALLINGER S., PEETHAMBARAN J., PARAKKAT A. D., DEY T. K., MUTHUGANAPATHY R.: 2d points curve reconstruction survey and benchmark. In *Computer Graphics Forum* (2021), no. 2. 2

[QSMG17] QI C. R., SU H., MO K., GUIBAS L. J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE CVPR* (2017). 1

[Sha08] SHAMIR A.: A survey on mesh segmentation techniques. In *Computer graphics forum* (2008), vol. 27, Wiley Online Library. 1

[SRS*19] STEINLECHNER H., RAINER B., SCHWÄRZLER M., HAASER G., SZABO A., MAIERHOFER S., WIMMER M.: Adaptive pointcloud segmentation for assisted interactions. In *Proceedings of the ACM SIGGRAPH Symposium on I3D* (2019). 1

[VTHLB15] VO A.-V., TRUONG-HONG L., LAEFER D. F., BERTOLOTTO M.: Octree-based region growing for point cloud segmentation. *ISPRS Journal of Photogrammetry and Remote Sensing 104* (2015). 1

[WSL*19] WANG Y., SUN Y., LIU Z., SARMA S. E., BRONSTEIN M. M., SOLOMON J. M.: Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog) 38*, 5 (2019). 1

[ZCD20] ZHANG S., CUI S., DING Z.: Hypergraph spectral clustering for point cloud segmentation. *IEEE Signal Processing Letters* (2020). 1