University of British Columbia

ENPH 479 Final Report

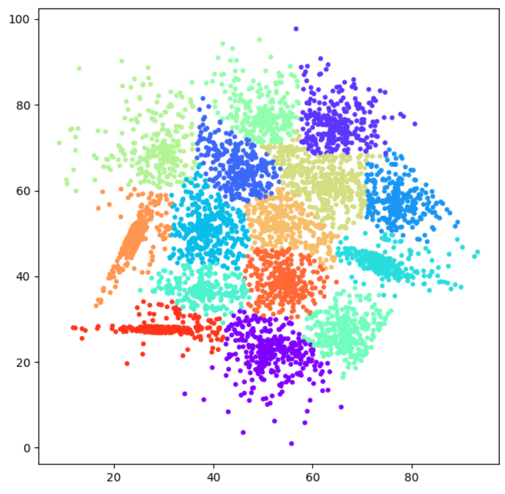
Design and Implementation of Point Cloud Features for the Topology Toolkit

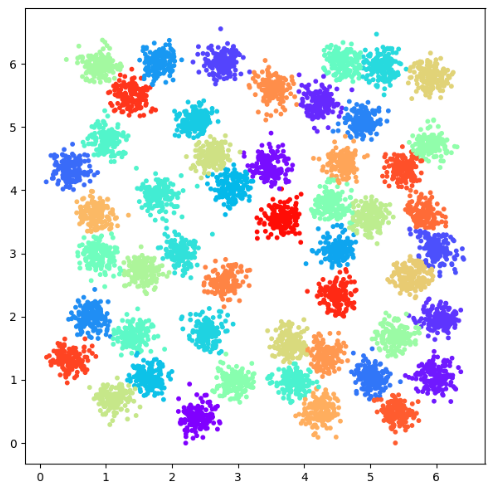
by Jim Shaw and Ryan Cotsakis

supervised by

Dr. Julien Tierny, Sorbonne Universite, France

Dr. Joshua Levine, University of Arizona, USA





Project 1862

Engineering Physics Project Laboratory

January 7, 2019

Executive Summary

Topological data analysis (TDA) for scientific visualization is a vibrant field of research with a plethora of novel techniques and applications. However, user-friendly access to these algorithms is seriously lacking. This was the original motivation behind the development of the Topology Toolkit (TTK), which is a library of TDA algorithms. Currently, TTK algorithms only operate on scalar mesh data formats. It can not handle point cloud data. Our project was to remedy this problem.

Firstly, we implemented two modules in TTK:

1. A module to interpret point cloud data as a scalar mesh. Furthermore, we designed a novel method for automatically setting parameters in the module.
2. A module which given a scalar mesh allows the user to conveniently simplify their data by retaining a subset of critical values for their mesh. We designed a novel method for estimating which critical values are **persistent** enough.

While the second module has more general applications than just point clouds, by using both together we can cluster point clouds. The second part of our project was to investigate the efficacy of clustering two dimensional point clouds using topological methods.

Our clustering method was completely non-parametric; the user could choose to not specify a single parameter. When compared to a set of parametric clustering methods, our method was generally as good as the top methods on a wide range of data sets. However, we noticed that none of the clustering methods that we tested, not even ours, were perfect on all data sets.

Our modules are currently implemented and functional. However, the automatic parameter selection algorithms can be improved. Our techniques have much room for refinement, and we outline possible approaches going forward. We also note that our modules can be sped up by using techniques that we did not have time to implement. We detail some approaches for doing this.

Contents

1 Introduction

Our project was to develop software for the Topology Toolkit (TTK) [14] which is a software library of topological data analysis (TDA) algorithms for scientific visualization. TDA is a very new field of research. Even with its rich theoretical foundation, it has not been used extensively because of a lack of available tools for end users. Our sponsors, Professor Joshua Levine at the University of Arizona and Professor Julien Tierny at Sorbonne University, as well as others, developed TTK as a user friendly set of tools that will allow more people to use techniques from TDA to visualize their own data sets.

TTK's algorithms only operate on scalar mesh (a.k.a a PL manifold with scalar field) data types. A scalar mesh is a set of points, a set of edges between the points, and real number specified at each vertex. Point cloud data is ubiquitous, and the sponsors wanted to be able to handle point cloud data. The idea was to initially develop a module to transform a point cloud to a scalar mesh in some way that is faithful to the geometry of the point cloud data. This module would then serve as an in-between conversion step to allow TTK algorithms to operate on point cloud data.

In addition to this module which converts points to a scalar mesh, the sponsors also welcomed the investigation of a possible use case of TTK. Our initial focus was working with point cloud data and a natural task with point cloud data was clustering. Therefore, we decided to investigate the efficacy of clustering 2-D point clouds using methods from TDA.

An interesting research question that arises from clustering point clouds is how one can non-parametrically cluster point clouds – that is, to allow the users to not have to specify any parameters. To do this, we decided to investigate how to automatically choose certain parameters to output a good clustering.

Since our project relies heavily on concepts from TDA, we offer a short introduction to the theory that is necessary for the comprehension of our algorithms in Section 3.1. In Section 3.2 we describe how we implemented our algorithms and designed our software which were ultimately packaged as deliverables. In Section 3.3 we describe how our system can be used to cluster point clouds, how we test the performance of our automatic parameter selection, and under which modes the clustering fails.

2 Background

Scientific visualization is a field which is concerned with graphical representations of data for the sake of deducing scientific conclusions, and TDA is a field which studies the topology of data. Here we use the word topology to mean geometry which is invariant under continuous deformation without gluing or cutting. A circle and a line have different topologies, for example, since one may not smoothly deform a line into a circle without gluing or cutting. However, a circle and a square have the same topology since we may stretch one into another. Past applications of TDA include visualizing atomic bonds [9] and the structure of galaxies [12].

2.1 Software Background

TTK is a software library written in C++. One may compile TTK and then use TTK's algorithms within their own code, or TTK can be plugged into Paraview [1]. Paraview is a front-end graphical user interface (GUI) for scientific visualization and is not associated with TTK. However, TTK can be realized as a set of tools inside Paraview. Users may call TTK algorithms from Paraview to analyze data without having to write any code. For more information on the structure of TTK as a software library, see [14].

Currently, we are not aware of competing software platforms which do TDA for visualization. As mentioned in [14], most available software for TDA come in the form of disparate software packages with each doing a separate task and having different specifications for input and output data formats.

2.2 Clustering Background

Clustering is the act of grouping data into similar groups. Similarity may be defined in many ways. In our case, we will cluster points in space, so that points which are close to one another will belong to the same cluster. Cluster analysis is a frequently used technique in data analysis [17].

Our project focuses mainly on the effective clustering of 2-D data and how to do it effectively. Our modules work for clustering 3-D data as well, although we did not focus on clustering 3-D data when testing the efficacy of our clustering technique. There are many algorithms for clustering in n-dimensional space, see [11]. None of the clustering algorithms in [11] use topological methods.

3 Discussion

3.1 Theory

3.1.1 Scalar Fields on Manifolds

We opt for an intuitive, informal exposition of the theory of TDA. See [4] for a much more thorough and mathematically rigorous explanation.

Scalar functions, i.e. scalar fields, map a **manifold** to the real numbers.

Definition 3.1 (Manifold)**.** We give two definitions of a manifold. We will use the informal version of our definition in our exposition.

1. (**Mathematical**) A manifold is a space that locally resembles Euclidean space.
2. (**Informal**) A manifold is a "nice" surface or volume without singularities. A 2-manifold is a surface. A 3-manifold is a volume.

3.1.2 Triangulation of Manifolds

Definition 3.2 (Triangulation)**.** A triangulation of a manifold is a representation of the manifold as a set of vertices (points in 3-D space) and edges between the vertices such that the resulting skeleton of vertices and edges are composed of triangles (for a surface/2-manifold) or tetrahedrons (for a volume/3-manifold).

Definition 3.3 (Piecewise Linear Manifold)**.** A surface or volume after triangulation is referred to as a **piecewise linear (PL) manifold**.

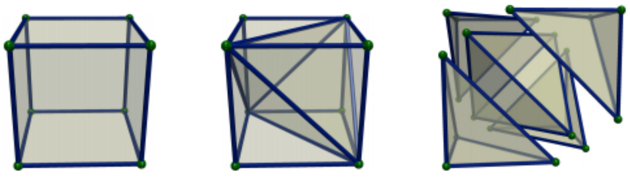


Figure 1: A triangulation of a 3-D cube into tetrahedrons. Figure from [13].

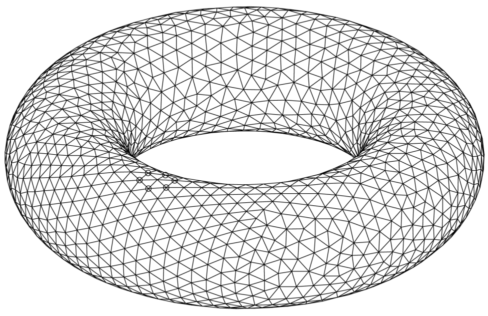


Figure 2: A triangulation of a 2-D surface into triangles. Note that this torus is a 2-D surface embedded into 3-D space. It is represented by triangles and not tetrahedrons. Figure from Wikipedia.

For example, after endowing the objects from Figure 1 or Figure 2 with a scalar field, we get an object which can be input to TTK's algorithms.

3.1.3 Critical Points of Scalar Functions

Definition 3.4 (Critical Point)**.** Given a scalar function $f : \mathbb{R}^n \to \mathbb{R}$, a critical point is a point in the domain such that $\nabla f(x) = 0$ i.e. the gradient vanishes.

For example, a scalar field from $\mathbb{R}^2$ to $\mathbb{R}$ can be represented by a 2-D surface. The peaks of the surface maximal critical points. The saddles and minima are also critical points, see Figure 3.

Notice that we did not define critical points for a scalar field on a general manifold, say, a torus for example. We defined them only in $n-$dimensional Euclidean space, $\mathbb{R}^n$. We can define critical points on a scalar function where the domain is a manifold and not just Euclidean space, but we leave out the formalism. The intuition of maxima, saddles, and minima are the same.

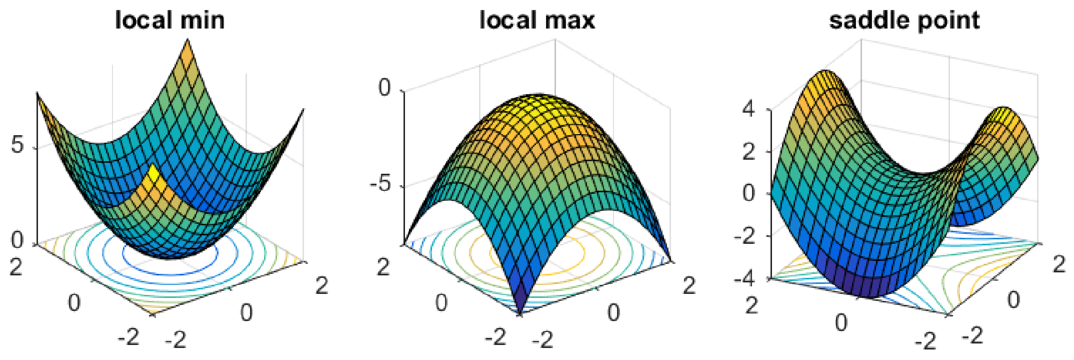


Figure 3: Three functions from $\mathbb{R}^2 \to \mathbb{R}$ showing three types of critical points : minimum, maximum, and saddle points. Taken from `https://www.offconvex.org/2016/03/22/saddlepoints/`.

Remark 3.5. Critical points as we have defined live on smooth, continuous manifolds with a scalar function. However, we work with PL manifolds which are *discrete* and can be represented in data. Thankfully, we can analogously define critical points on PL manifolds in a way that preserves the intuition of critical points as maxima and minima, see [4]. When we refer to critical points in the following sections, we will implicitly refer to critical points on PL manifolds.

The most important example of a PL manifold we will work with is a triangulated, flat grid. For clustering, we analyze critical points for this triangulated grid.

3.1.4 Persistence

At the heart of TDA is the concept of persistence. Loosely, the persistence of a feature in a scalar field is a measure of its size.

Definition 3.6 (1-D Persistence Diagram)**.** Given a parameter α, a scalar function $f : M \to \mathbb{R}$ let $A \subset M$ be the subset of the domain for which f maps to values less than α. The connectedness of A varies as α is varied, and so connected components can be said to be born and die as α is increased from 0 to ∞. The lifetime of each connected component is captured in the persistence diagram by plotting its death time on the y-axis against its birth time on the x-axis.

Figure 4 is the **persistance diagram** corresponding to the red function, f. A connected component of A is born at the bottom of each vertical blue bar, and dies by getting absorbed into an older connected component at the top of that blue bar. The vertical distance from each point in the persistence diagram to the green diagonal (the length of the blue bar) is called its **persistence**. The three points in the persistence diagram can be seen to quantify the *size* of the hills.

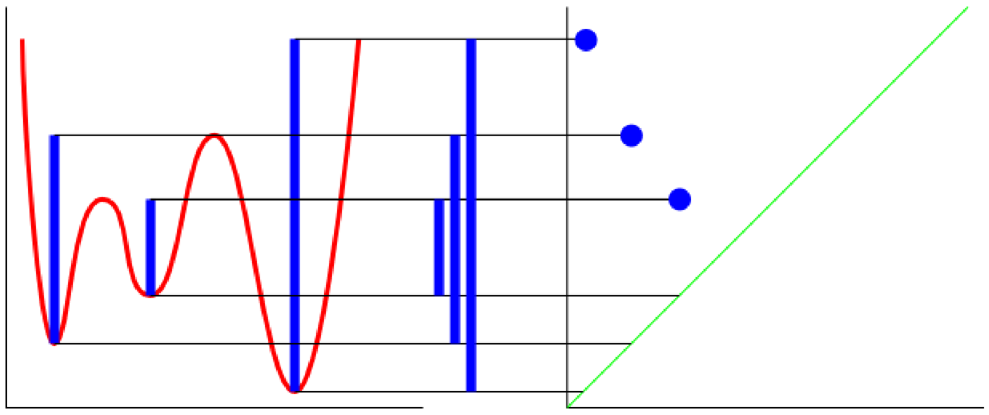


Figure 4: A visual representation of a persistence diagram. Each bar on the leftmost figure represents the height difference between a maximum and a minimum. This leads to a pairing and a plot on the rightmost graph, which is the persistence diagram. Taken from `https://faculty.math.illinois.edu/~ymb/talks/brown_perp/men13.html`

Remark 3.7. We may extend the definition of a 1-D persistence diagram above to multiple dimensions. For example, the red curve in Figure 4 will become a red surface instead in 2-D. The idea is the same, vary a parameter α and see how the topology of A changes. For the case of 2-D persistence diagrams, the points represent maximum-saddle pairs and minimum-saddle pairs instead of maximum-minimum pairs. We will mostly work with 2-D persistence diagrams.

The main usefulness of the concept of persistence is that persistent pairs of critical points stay persistent **even in the presence of noise!** This can be formalized and proven mathematically [4]. Similarly, noise creates persistence pairs which are not very persistent. See Figure 5.

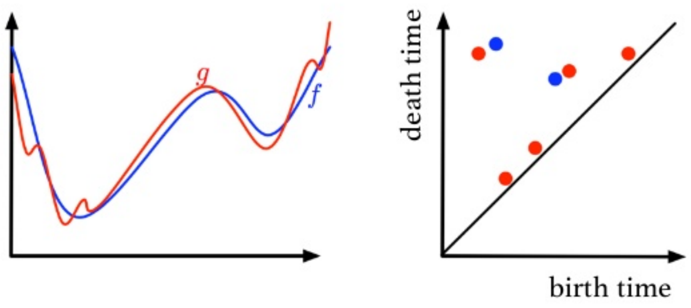


Figure 5: Persistence diagram for the red and blue curve on the left. The red curve is noisy and thus has some small bumps. These small bumps lead to critical pairs of small persistence corresponding to the red dots near the diagonal on the right persistence diagram. The blue persistent pairs do not get moved much on the persistence diagram with the addition of noise; there is **stability** in the persistence diagram. Image from Wikipedia.

What this means is that given a persistence diagram, we can identify non-persistent critical point pairs. These non-persistent critical point pairs will look like bumps on the red curve in Figure 5. We can then apply techniques such as the ones in [15] to remove those small bumps.

We frequently represent the persistence diagram in a different way, such as in Figure 11. We get the latter diagram by plotting the ith most persistence critical pair as (i, p_i) with p_i being the persistence of the pair.

3.2 Software and Algorithm Design

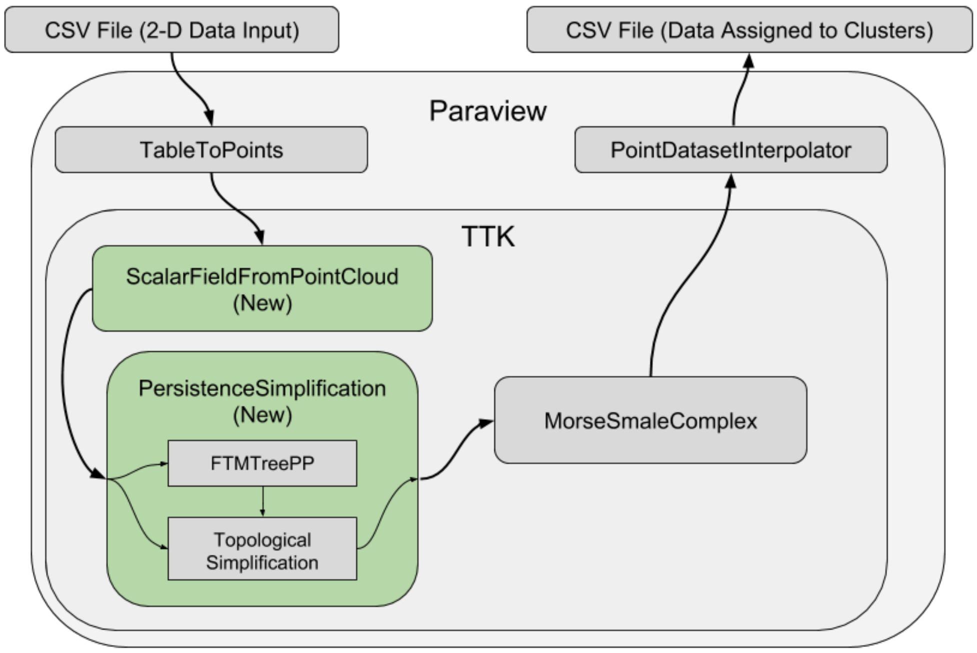

CSV File (2-D Data Input)
CSV File (Data Assigned to Clusters)
Paraview
TableToPoints
PointDatasetInterpolator
TTK
ScalarFieldFromPointCloud (New)
PersistenceSimplification (New)
FTMTreePP
Topological Simplification
MorseSmaleComplex

Figure 6: System level diagram for how the clustering procedure works. Each box represents a software module, either written for Paraview or for TTK. We created the two modules coloured green. The arrows represent a sequential flow of data from a file with point data to a file with labeled point data.

There are a number of useful modules that already exist in TTK and Paraview that we make use of to complete our clustering pipeline:

1. **Table to Points** converts a CSV file to a point cloud available for manipulation in Paraview.
2. **Persistence Diagram** computes the persistence diagram pertaining to a scalar field.
3. **Topological Simplification** takes a scalar field and a subset of the points in its persistence diagram, and smooths the scalar field such that the excluded (less persistent) extrema points are removed.

4. **Morse Smale Complex** can be used to partition the domain of a scalar field f into regions that map to the same maximum when traversing the domain in the direction of ∇f. See Figure 7.

5. **Point Dataset Interpolator** will map each point in the point cloud to a partition of the domain.

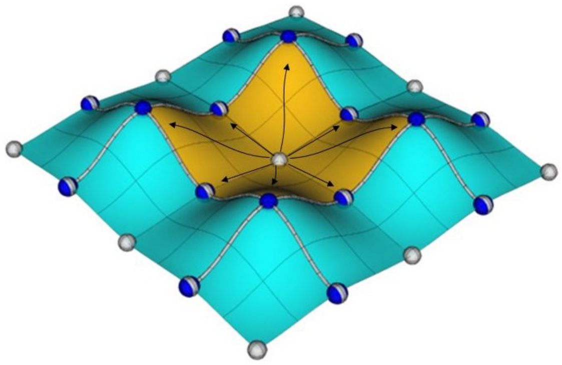


Figure 7: Partitioning domain by hill climbing/descent. The blue points are maxima, blue-white points are saddles, and white are minima. The yellow region represents all points on this surface where following the direction of $-\nabla f$ descends to the white minimum in the middle. We can partition the surface by regions where points climb down to a certain minimum. Alternatively, we can partition the surface by regions where points climb *up* to certain *maxima*. In our context, each region is a cluster.

We add two modules to this repertoire to give users the ability to cluster a set of points. Those are:

1. **Scalar Field From Point Cloud** takes a point cloud and creates a regular (equally spaced) grid in the space surrounding the points. Scalar values may be associated to the vertices of this grid by density or by distance to point cloud.

2. **Persistence Simplification** takes a scalar field and removes the less persistent extrema without requiring a trimmed persistence diagram.

Using these modules we can partition a point cloud into clusters; see Figure 6.

3.2.1 Scalar Field From Point Cloud

We created a module named *ScalarFieldFromPointCloud* for TTK. ScalarFieldFromPointCloud takes a point cloud as an input and constructs a triangulated regular grid surrounding the input point cloud. The grid size may be specified by the user. The grid is the domain of the scalar field that ScalarField-FromPointCloud outputs. Users may select two options for calculating the scalar values on each grid point: a kernel density estimation (KDE) or a distance field.

The scalar values for the KDE are calculated as follows: We construct identical Gaussian distributions centered at each point in the point cloud. A continuous scalar function $f : \mathbb{R}^n \to \mathbb{R}$ defined on the ambient space can be constructed by taking the sum of all of the Gaussian distributions. The standard deviation or width of the Gaussians is denoted as the *bandwidth*, which is a parameter the user may choose. The value of the output scalar field for a particular vertex in the regular grid is determined to be the value of f at that point. See Figure 8 for a graphic explanation.

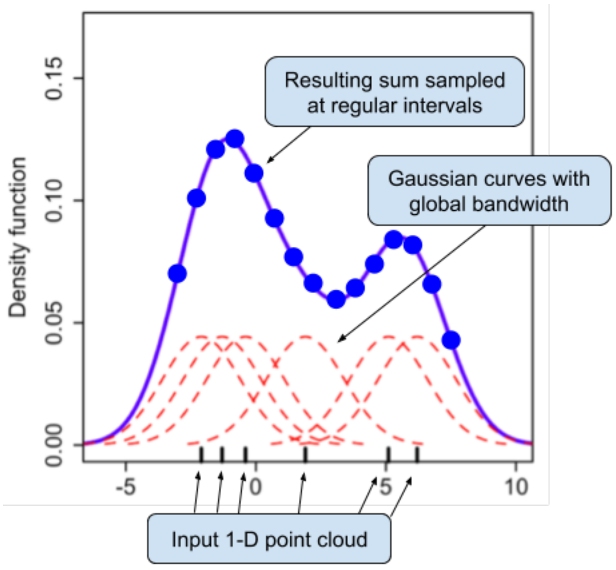


Figure 8: Gaussian kernel density estimate. A Gaussian of a certain bandwidth is centered at each point. The Gaussians are added and the purple curve is the resulting probability distribution that is estimated from the samples. Image taken from Wikipedia

We also implement a feature that automatically guesses what the bandwidth should be. For each point in the point cloud, we calculate the distance to the $k-$th closest neighbour. The parameter k is an integer, and we estimate it as follows

$$k = (0.587 \cdot n^{4/5})^{1/d}$$

and then rounding k up. n is the number of points in the point cloud, d is the dimension of the data (either 2-D or 3-D point clouds). This method is adapted from [10]. Finally, the mean distance to the $k-$th closest neighbour for every point in the point cloud is taken to obtain the final bandwidth which is used. This method of bandwidth selection can be used in ScalarFieldFromPointCloud module by selecting the "Automatic Bandwidth for KDE" option in Paraview.

For the distance field, the scalar values on the regular grid can simply be taken to be the distance to the closest point in the point cloud. In effect, this will yield very small values for points with nearby neighbours, whereas the KDE will yield the largest values in regions of high density.

Algorithmically, we simply computed the nearest neighbours by iterating over all points of the point cloud. This naive method may be improved. See Section 5.

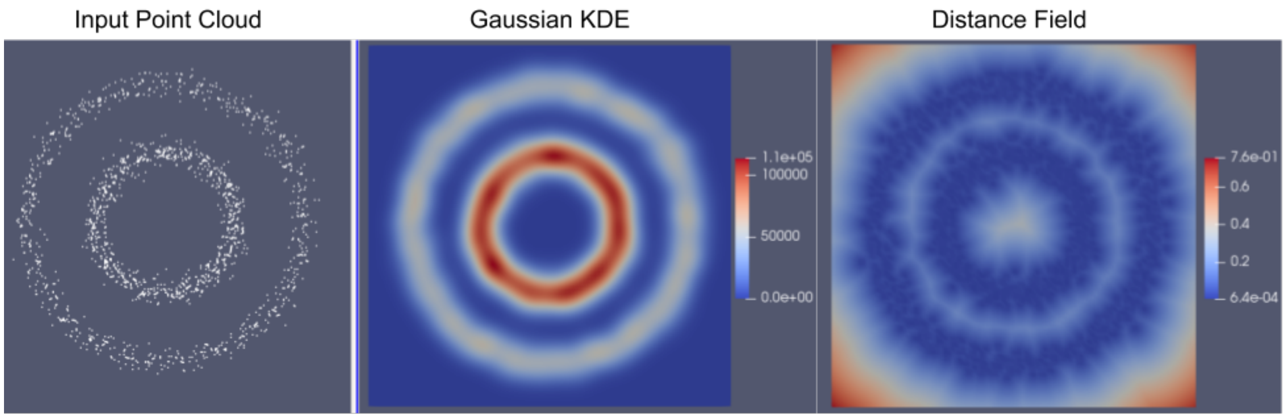


Figure 9: The input to ScalarFieldFromPointCloud on the left, the output scalar field in the middle when using the Gaussian KDE option, and the output scalar field on the right when using the distance field option. For the KDE, the colour map is red at locations with a high density of points, and blue in sparse regions; in the distance field, the colour map is red far away from the points and blue near the points.

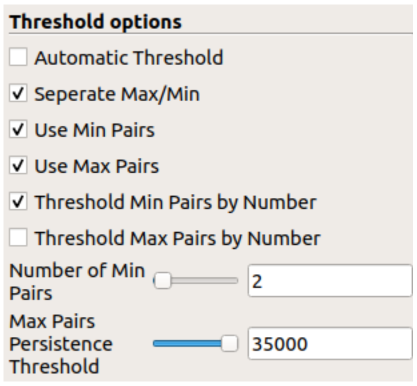
3.2.2 Persistence Simplification

The Persistence Simplification module employs Topological Simplification without the user having to compute the persistence diagram and select pairs manually. The user is also given the option to have Persistence Simplification choose which points are sufficiently persistent automatically, which is not a feature that was previously implemented in any TTK or Paraview plugin.

There are a number of use cases for the Persistence Simplification module. Although we are focused on clustering applications, we have made the module as versatile as possible. The user can override the automatic threshold by interacting with the GUI in Paraview to choose various threshold options:

- The user has the option to manually threshold which persistence pairs are used to simplify the scalar field based on a persistence threshold.
- The user can choose to simplify the scalar field using the maximum-saddle pairs, the minimum-saddle pairs, both together, and also separately with different threshold values for each.
- The user can threshold based on the number of pairs they would like to simplify with. The most persistent points will be chosen.

The module acts as a plugin for Paraview, but it can also be called as an object in future TTK modules. First, there is a layer of code that interprets the objects from Paraview, and makes them accessible to the TTK base code. The input scalar field is used to create an output scalar field by a class defined in the base code called *PersistenceSimplification* which can be constructed in future


Threshold options

- [ ] Automatic Threshold
- [✓] Seperate Max/Min
- [✓] Use Min Pairs
- [✓] Use Max Pairs
- [✓] Threshold Min Pairs by Number
- [ ] Threshold Max Pairs by Number

Number of Min Pairs	2
Max Pairs Persistence Threshold	35000

Figure 10: GUI for Persistence Simplification. By checking "Separate Max/Min", the user may select minimum-saddle pairs and maximum-saddle pairs in different ways. By ticking "Use Min/Max Pairs" the module will choose to retain minimum/maximum-saddle pairs. Since "Threshold Min Pairs by Number" is specified, we can choose the number of minimum-saddle pairs to retain. Alternatively, since "Threshold Max Pairs by Number" is not specified, we retain maximum-saddle pairs by thresholding persistence instead.

TTK modules. To create the output scalar field, the module makes use of two other objects defined in TTK, those are *FTMTreePP*, and *TopologicalSimplification*. First, the scalar field is given to the FTMTreePP object, and the persistence pairs are computed. The pairs can then be sorted based on their persistence, and then a persistence threshold is decided given that the user has specified that the persistence threshold be automatically determined.

To automatically decide a persistence threshold, we begin by looking at the least persistent pair. Let $a = 0.2$ and $b = 0.025$ be two parameters, and p_n be the persistence of the most persistent critical pair in our data set. If the critical pair in question has persistence p, we ask if the next most persistent pair has persistence greater than $(1 + a)p + bp_n$ and if the pair after that has persistence greater than $(1 + 2a)p + 2bp_n$. If the answer to both questions is yes, then the persistence threshold is determined to be $(1 + a)p + bp_n$. If the answer to either question is no, then we remove the initial pair and begin the analysis again on the next pair, until a persistence threshold is decided. An illustration of this process can be seen in Figure 11.

We noticed that noisy persistent pairs fit an exponential function quite well. a is chosen to be the parameter that describes the exponential growth of persistence in Figure 11. b is chosen to be a nice round number that seemed to perform adequately for our data. Note that a is fixed as a parameter, but ideally we want to estimate an ideal a for each data set by fitting an exponential to the data. We did not have time to do this. See our recommendations in Section 5.

The pairs with sufficient persistence are then given to the TopologicalSimplification object along with the input scalar field. The output scalar field of the TopologicalSimplification object becomes the output of PersistenceSimplification.

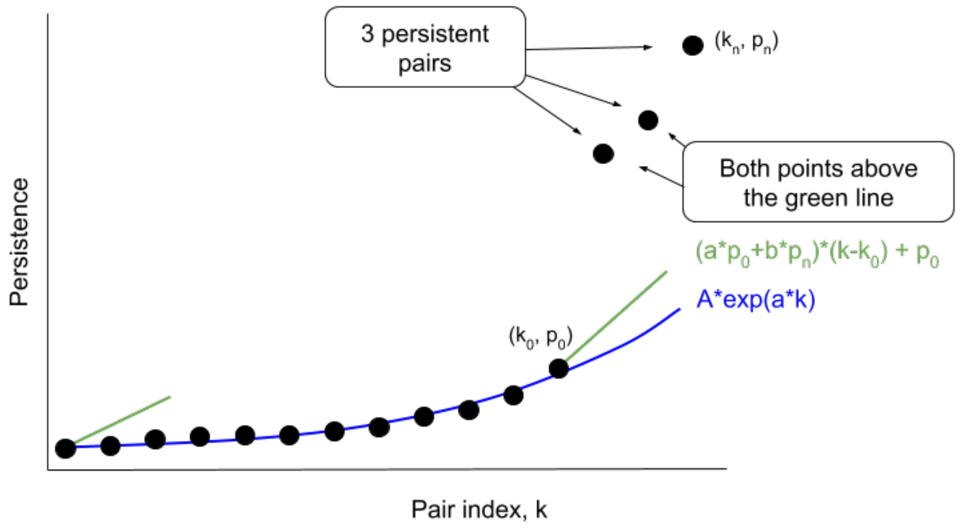


Figure 11: Automatic persistence thresholding. Each black point is a critical point pair. We sort the pairs by persistence. We assume the noise assumes an exponential shape. $(a \cdot p_0 + b \cdot p_n)(k - k_0) + p_0$ is the tangent line to the exponential plus some slope $b \cdot p_n$, where p_n is the persistence of the most persistent critical pair. The algorithm essentially looks point by point to see if the next point is above this green line. If the next two points lie above the line, then the all further points are persistent enough.

3.2.3 Alternative Algorithms

The efficacy of the automatic parameter selection for the two additional modules depends heavily on the application. To demonstrate that the parameters were being selected correctly, we observed how well different data sets were clustered with the clustering pipeline.

For the bandwidth selection in the ScalarFieldFromPointCloud module, we chose to have the Gaussians be identical. We tried another method with each Guassian having a (possibly different) bandwidth proportional to the $k-$th nearest neighbour [3]. When exploring this variation, we found that the least persistent clusters would have an even smaller persistence with this algorithm, making it likely for PerisistenceSimplification to discard it as noise.

As for the PersistenceSimplification module, we tried alternative methods that included looking at more than just next following two pairs, but neither this nor any other alternatives were tested extensively.

3.3 Testing and Results

We test the effectiveness of our automatic parameter selection techniques and analyze where they fail. In the case of clustering, with our modules a user may manually select the number of clusters and bandwidth for the density estimation. However, in the below tests, we only cluster with parameters selected through our automatic selection techniques.

We cluster on three groups of data sets:

1. The FCPS clustering suite [16].
2. A k-means clustering testing suite [7].
3. The scikitlearn clustering data sets [11].

The corresponding results are shown in Figures 12, 13 and 14. We found that the results for data set [7] perform reasonably well, so we analyze the other two data sets which are trickier and produce more interesting results.

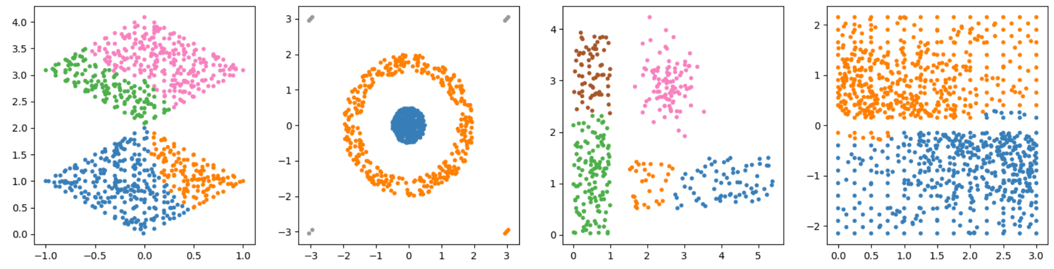


Figure 12: Our method, automatic persistence clustering, applied to four data sets from [16]. Each colour indicates a different cluster, as labeled by our algorithm.

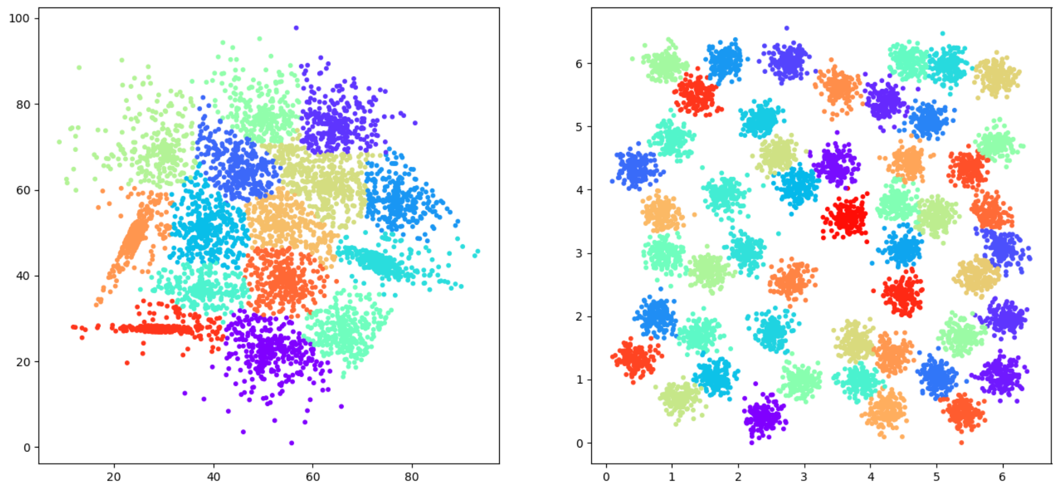


Figure 13: Our method, automatic persistence clustering, applied to data sets from [7]. Each colour indicates a different cluster, as labeled by our algorithm.

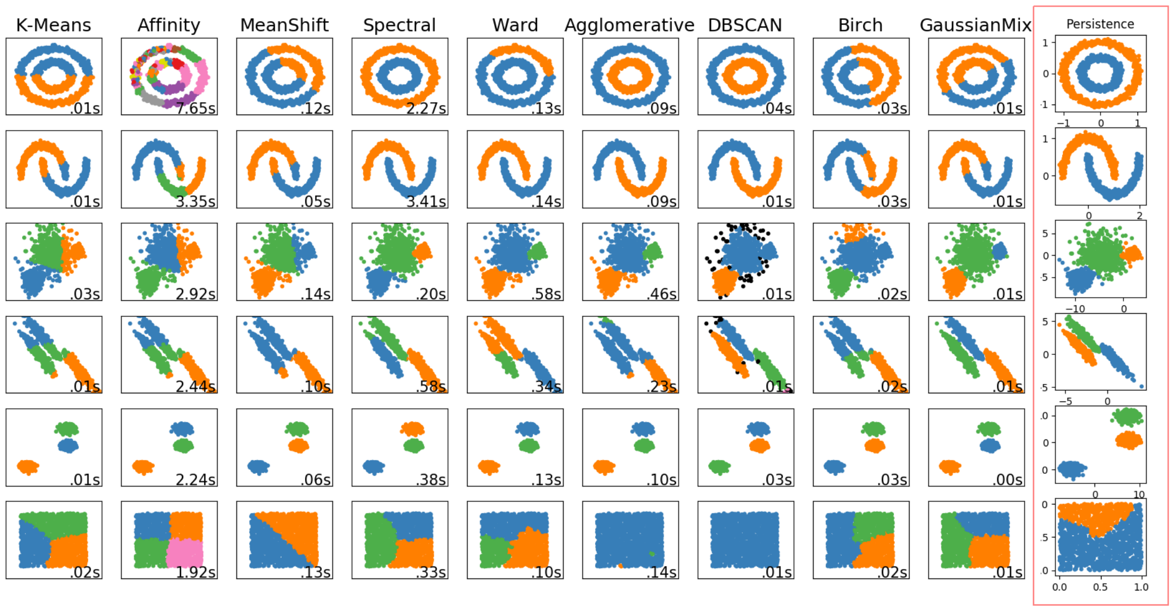


Figure 14: A comparison of clustering methods. Each colour indicates a different labeling according to the algorithm. Our method, "Persistence", is outlined in red. See `scikit-learn.org/stable/modules/clustering.html` for information on the other clustering methods.

3.3.1 Clustering Algorithm Comparison

We tested our clustering technique on a variety of data sets and compared it to nine other well-established clustering methods on 2-D data. The results are shown in Figure 14. The generation of these data sets is specified in `https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html`. There is randomness in the generation of the data sets which explains the discrepancy between Figure 14 and the first figure in the link.

In the data sets, we are given the ground truth for what the clusters should be. This means we can apply the Fowlkes-Mallows score for clustering [6]. The Fowlkes-Mallow score is a goodness of clustering score from 0 to 1, with 1 being a perfect clustering and smaller values being bad clusterings. See Section 7.1 for more information.

We have labeled each method and each data set with a Fowlkes-Mallows score and summed up the scores for all data sets and each algorithm in Table 1.

All algorithms in Figure 14 except ours which is outlined in red and denoted "Persistence" needs to have parameters specified. For these other algorithms, we used the same parameters as in `scikit-learn.org/stable/modules/clustering.html`, which were picked to perform well on these data sets. The best algorithms according to the sum of scores is Persistence and DBSCAN. DBSCAN, however, does not classify all points as seen by the black outliers in the figure.

	K-Means	Affinity	Mean Shift	Spectral	Ward	Agglom.	DBSCAN	Birch	Gauss. Mix	**Persistence**
Circles	0.50	0.36	0.42	1.0	0.66	1.0	1.0	0.51	0.50	**1.0**
Moons	0.74	0.67	0.77	1.0	1.0	1.0	1.0	0.57	0.75	**1.0**
Varied Density	0.87	0.88	0.90	0.96	0.95	0.95	0.75	0.74	0.98	**0.96**
Anisotropic	0.73	0.74	0.75	0.97	0.79	0.69	0.98	0.72	1.0	**1.0**
Blobs	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	**1.0**
No Structure	0.58	0.50	0.71	0.58	0.59	0.99	1.0	0.59	0.58	**0.77**
Sum of Scores	*4.42/6.0*	*4.15/6.0*	*4.55/6.0*	*5.51/6.0*	*4.99/6.0*	*5.63/6.0*	*5.73/6.0*	*4.13/6.0*	*4.81/6.0*	***5.73/6.0***

Table 1: The Fowlkes-Mallows score, a metric for goodness of clustering, for the corresponding algorithm (column) and data set (row). The metric ranges from 0 to 1, with 0 being a bad clustering and 1 being a perfect clustering. The data sets correspond in order to the data sets in Figure 14.

3.3.2 Clustering Failure

In Figure 14 we see that the only data set for which our automatic clustering method fails drastically is the last data set, which has only 1 cluster. Figure 15, is a graphic illustrating the clustering procedure for the final data set. We refer to this figure in continuing analysis.

Firstly, the data itself is not ideal for our method because compared to the other clusters in Figure 14 because:

1. This cluster has a large area and thus is being under sampled. This is especially important for kernel density estimations.
2. This data set is sampled from a uniform distribution on a square. We estimate density by summing Gaussians, which seems to work better when the data set is sampled from a Gaussian.

This explains the “patchy-ness” in the density estimation.

Despite the relatively low quality density estimation, our automatic threshold for persistence almost worked. In this test, a in the automatic persistence algorithm was set to 0.2. In reality, by fitting an exponential curve to the data, we get that a should be 0.33 instead. This leads to a correct clustering.

The data set in Figure 12 with two diamonds fails to automatically cluster for the same reasons as above. This can be seen in Figure 16. The same patchiness can be seen, which leads to an incorrect thresholding. Interestingly, fitting an exponential curve to the persistence curve yields $a = 0.53$, which we found would indeed lead to the correct two clusters. We note this in Section 5.

In both Figures 15 and 16 it seems like the bandwidth is too small for the density estimation. However, we noticed that the automatic bandwidth selection on the data sets in Figure 13 yielded bandwidths which if increased would yield incorrect clusterings. Therefore simply tuning the automatic bandwidth selection to give bigger bandwidths is not a solution.

3.3.3 Automatic Persistence Simplification - Other Data Sets

Our persistence simplification module may be used on more general scalar PL manifolds which do not necessarily come from point clouds. We ran our automatic persistence simplification algorithm on two data sets which come with TTK as examples, and then visualized the resulting level of simplification. The results can be seen in Figure 17 and 18.

In Figure 17, we seem to obtain the "correct" critical points based on the persistence curve and our results, which is encouraging. Finding the right peaks is analogous to finding the correct clusters in our previous analysis, which suggests the success of our algorithm.

In Figure 18, we seem to simplify our data too much. We only pick out two features, the tail and the head , while we end up ignoring the spikes on the dragon. While our results are not "wrong" because the right level of simplification depends on what the user wants to view, this seems to indicate that our algorithm is quite aggressive for simplifying data.

4 Conclusions

We have implemented a module that allows users of TTK to handle point cloud data by transforming point clouds into the scalar mesh data format which is described in Section 3.2.1.

We have contributed another module to TTK which allows for automatic persistence simplification. In Section 3.3 we have determined that it is possible to use these two modules along with other TDA methods to cluster 2-D point clouds non-parametrically and be as effective as other methods (see Table 1). Although we have shown that our clustering algorithm works well on the majority of our test suites, we have also identified types of data sets which pose challenges to our clustering method.

We have also investigated parameter selection in the context of TDA in Section 3.2. Our automatic persistence thresholding algorithm to our best knowledge is novel. Our automatic bandwidth selection algorithm has been shown to perform reasonably well in the context of TDA. However, we have not tested our parameter selection methods on volumes (3-manifolds).

5 Recommendations

We first list recommendations for our project which we would have been the immediate next steps if not for time constraints.

1. We have to find the nearest neighbour of a point in the Scalar Field from Point Cloud module. Implementing techniques such as k-d trees [2] would allow for better time complexity.

2. The parameter a in our algorithm for automatic persistence simplification is currently set to be a constant, see Figure 11. Since a represents the constant in e^{ax} which describes the noise, we should be able to learn the constant a for each data set. A way to do this is to do a robust linear

regression via RANSAC [5] on the log scale plot of Figure 11. The log scale plot of the noise would look something like $y = ax$ with outliers, and the slope of the linear fit should be a. In Section 3.3, we found that this would alleviate problems with clustering.

In the context of clustering, an interesting question is the choice of kernel function to be summed over in the density estimation. We use a Gaussian kernel, but other types of kernels exist. Some authors state that the choice of kernel function is not important for the estimation of density compared to the choice of bandwidth [8], but in the context of persistence clustering this may not be a case. We recommend:

1. Implementing a feature to allow different types of kernels for users to input in our Scalar Field from Point Cloud module.
2. An investigation into the effects of different kernels with respect to clustering. For example, would a uniform kernel work well for a uniform distribution?

Finally, since the rest of TTK operates on 3-manifolds, we strongly recommend an investigation into how our parameter selection techniques perform on 3-manifolds.

6 Deliverables

We have contributed two modules to TTK, a new clustering algorithm with these modules, and a detailed test of our how effective our algorithm is.

The additions to TTK include:

- A module that creates a scalar field and grid around an input 2-D or 3-D point cloud, and at each vertex of the grid associates a value to the distance to the nearest point in the point cloud. Alternatively, the module can create the scalar field by a KDE with a bandwidth automatically or manually specified by the user.
- A module which takes as an input a scalar mesh and simplifies it. The module allows the user to decide the persistence threshold for the maxima and the minima separately or simultaneously. The user can input this threshold as the number of maxima and minima that they require in the output simplified scalar field. If requested by the user, the module will automatically set the persistence thresholds to reasonable values.

Along with other tools available in TTK, we exploited the automatic parameter selection algorithms in both modules to create a novel clustering algorithm that can outperform other clustering algorithms. The clustering algorithm comes in the form of an assembled pipeline of TTK modules that will convert 2-D data points into a clustered data set in Paraview while requiring no input from the user.

The main discrepancies between the deliverables set forth in the project proposal lie in the automatic parameter selection. We did not anticipate that this would be such a central component of our project.

7 Appendices

7.1 Fowlkes-Mallows Score

The Fowlkes-Mallows score assigns a score to a particular clustering of a data set. A score of 1 indicates a perfect clustering; a clustering which matches the ground truth exactly. As clustering gets worse, the score approaches 0. The interpretation is not obvious, but Fowlkes and Mallows showed that for pairs of random clusterings of a given data set, the value of the score approaches 0 if the data set has a large number of points.

The Fowlkes-Mallows score is defined as

$$\text{Fowlkes-Mallows Score} = \frac{TP}{\sqrt{(TP + FP)(TP + FN)}} \tag{7.1}$$

where TP is the number of True Positives, pairs of points which belong in the same cluster in the ground truth and the algorithm labeling. FP is the number of False Positives, pairs of points which belong in the same cluster in the ground truth but not in the algorithm labeling. FN is the number of False Negatives, pairs of points which belong in different clusters in the ground truth but the same cluster in the algorithm labeling.

References

[1] Utkarsh Ayachit. The paraview guide: a parallel visualization application. 2015.

[2] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[3] Leo Breiman, William Meisel, and Edward Purcell. Variable kernel estimates of multivariate densities. *Technometrics*, 19(2):135–144, 1977.

[4] Herbert Edelsbrunner and John Harer. *Computational topology: an introduction.* American Mathematical Society, 2010.

[5] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[6] Edward B Fowlkes and Colin L Mallows. A method for comparing two hierarchical clusterings. *Journal of the American statistical association*, 78(383):553–569, 1983.

[7] Pasi Fränti and Sami Sieranoja. K-means properties on six clustering benchmark datasets. `http://cs.joensuu.fi/sipu/datasets/`, 2018.

[8] Arsalane Chouaib Guidoum. Kernel estimator and bandwidth selection for density and its derivatives.

[9] David Günther, Roberto A Boto, Juila Contreras-Garcia, Jean-Philip Piquemal, and Julien Tierny. Characterizing molecular interactions in chemical systems. *IEEE transactions on visualization and computer graphics*, 20(12):2476–2485, 2014.

[10] Jan Orava. K-nearest neighbour kernel density estimation, the choice of optimal k. *Tatra Mountains Mathematical Publications*, 50(1):39–50, 2011.

[11] Scikitlearn. Clustering. `https://scikit-learn.org/stable/modules/clustering.html`, 2019. [Online; accessed 02-January-2019].

[12] Nithin Shivashankar, Pratyush Pranav, Vijay Natarajan, Rien van de Weygaert, EG Patrick Bos, and Steven Rieder. Felix: A topology based framework for visual exploration of cosmic filaments. *IEEE Transactions on Visualization and Computer Graphics*, 22(6):1745–1759, 2016.

[13] Julien Tierny. `https://www-pequan.lip6.fr/~tierny/private/topologicalDataAnalysis.pdf`. Accessed: 2018-09-15.

[14] Julien Tierny, Guillaume Favelier, Joshua A Levine, Charles Gueunet, and Michael Michaux. The topology toolkit. *IEEE transactions on visualization and computer graphics*, 24(1):832–842, 2018.

[15] Julien Tierny, David Günther, and Valerio Pascucci. Optimal general simplification of scalar fields on surfaces. In *Topological and Statistical Methods for Complex Data*, pages 57–71. Springer, 2015.

[16] A Ultsch. Clustering with som: Uc. *Proc. Workshop on Self-Organizing Maps*, pages 75–82, 2005.

[17] Wikipedia. Cluster analysis — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Cluster%20analysis&oldid=876519847`, 2019. [Online; accessed 02-January-2019].

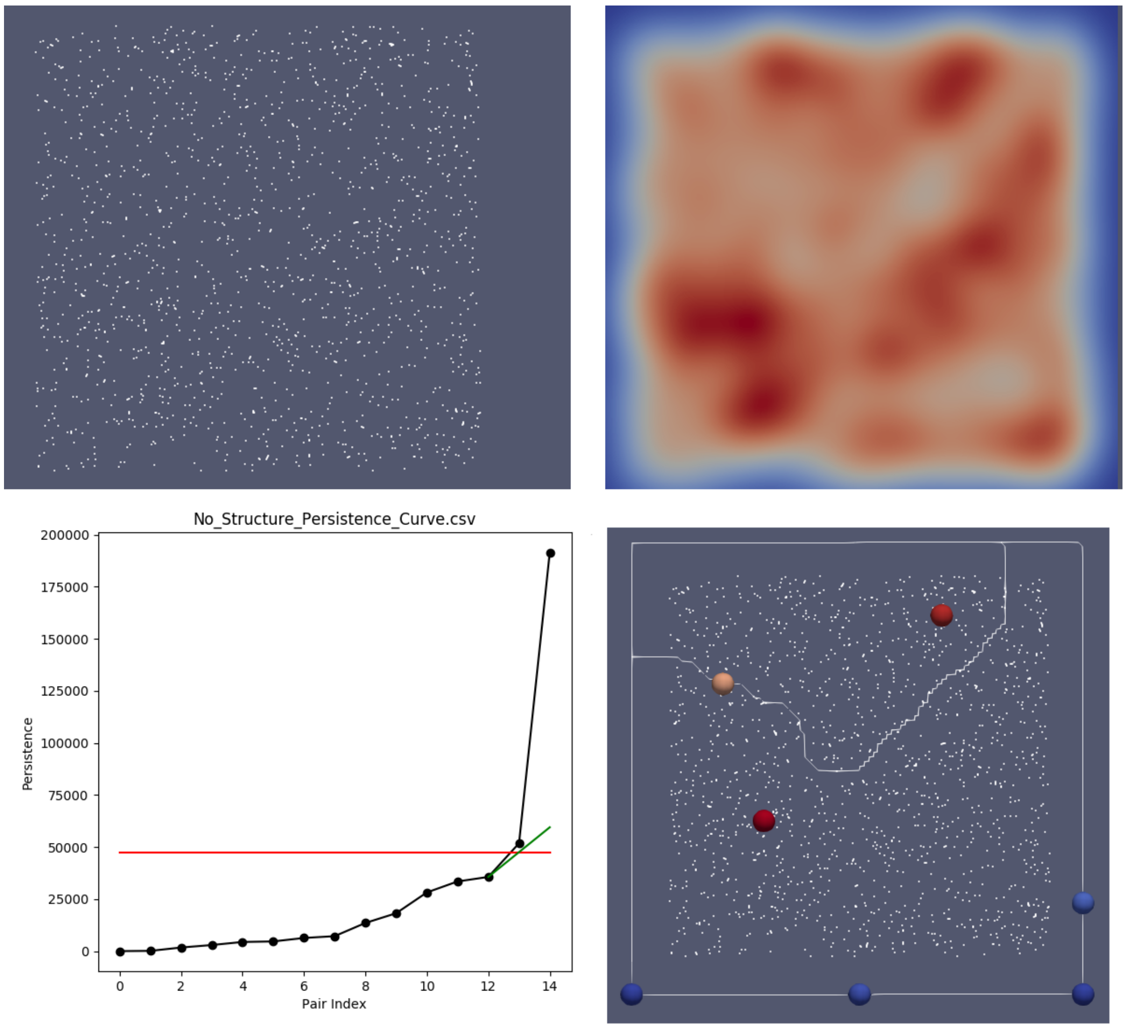


Figure 15: Clustering of a data set which is sampled from a uniform distribution on a square. Top Left - Original data set. Top Right - Kernel density estimation; red is high, blue is low. Bottom Left - Automatic persistence thresholding, with the red line representing the persistence at which we threshold. Each black dot represents a maximum-saddle pair. We order the pairs by persistence. The point just above the red threshold lies above the green line, hence the algorithm terminates. Bottom Right - Segmented domain, with each segment representing a cluster. The red spheres are maxima, blue minima, and yellow saddle.

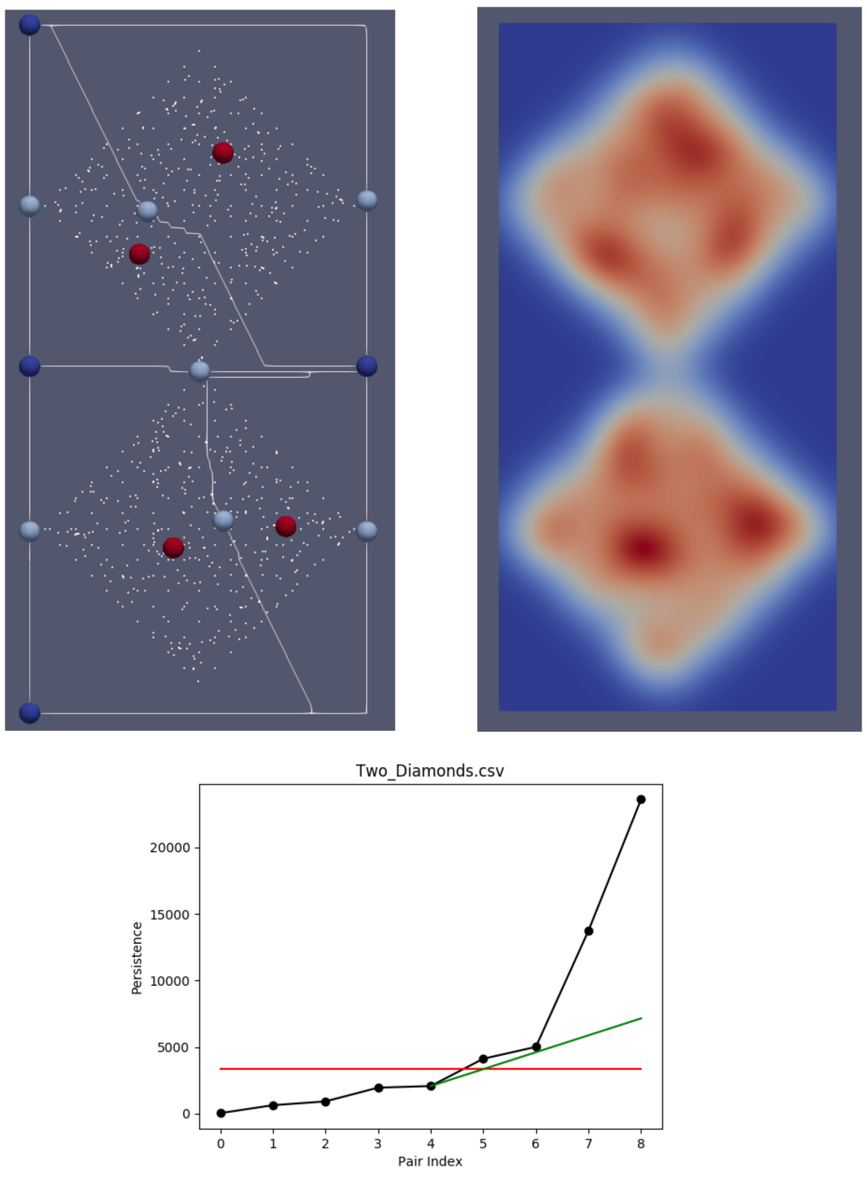


Figure 16: Detailed clustering of a data set in Figure 12. Top Left - Data set with segments representing clusters and spheres representing critical points. Top Right - Kernel density estimation; red is high, blue is low. Bottom - Automatic persistence thresholding, with the red line representing where the algorithm decides to threshold. Each black dot represents a maximum-saddle pair. We order the pairs by persistence. The point just above the red threshold lies above the green line, showing why the algorithm terminates at this threshold.

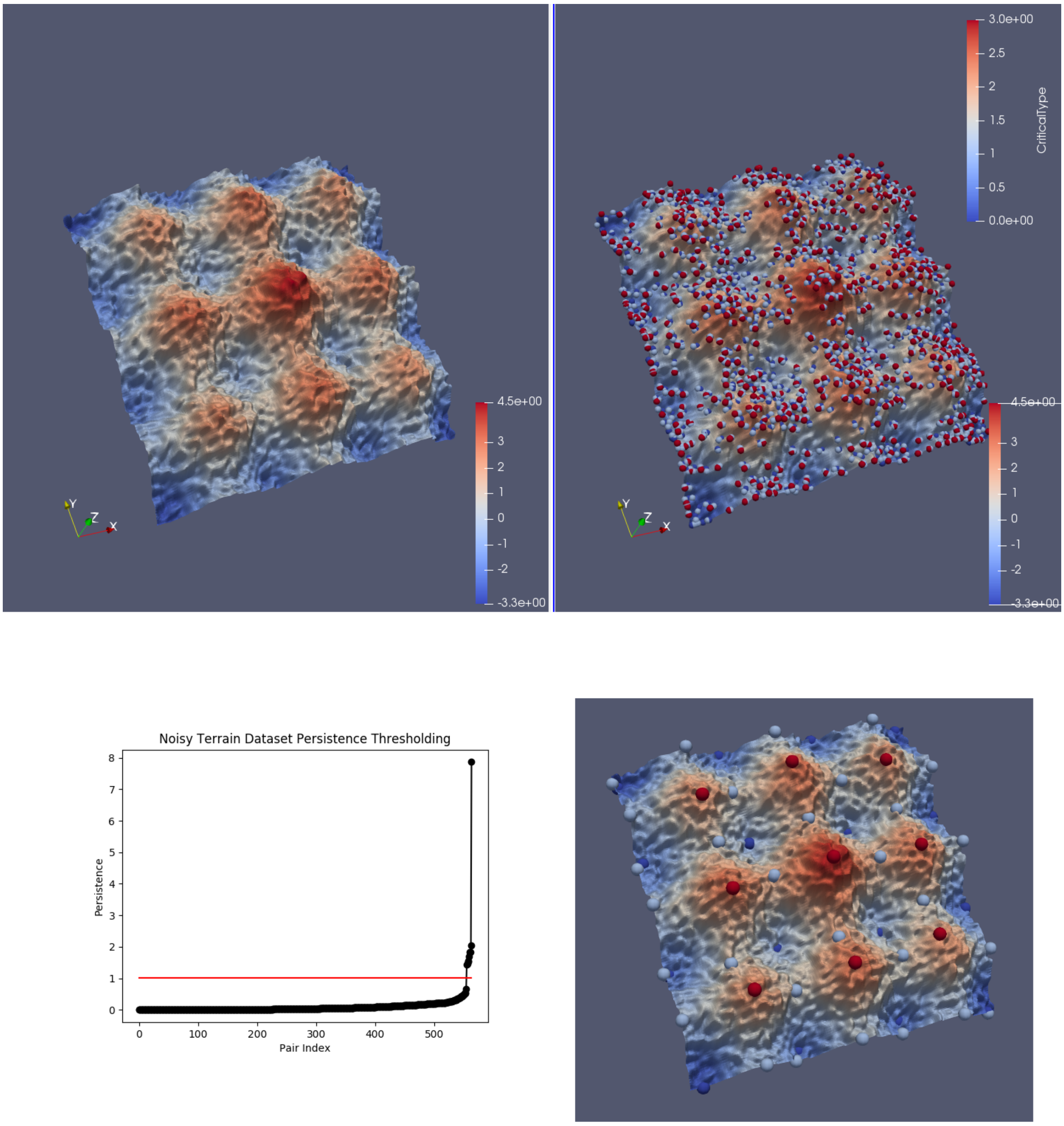
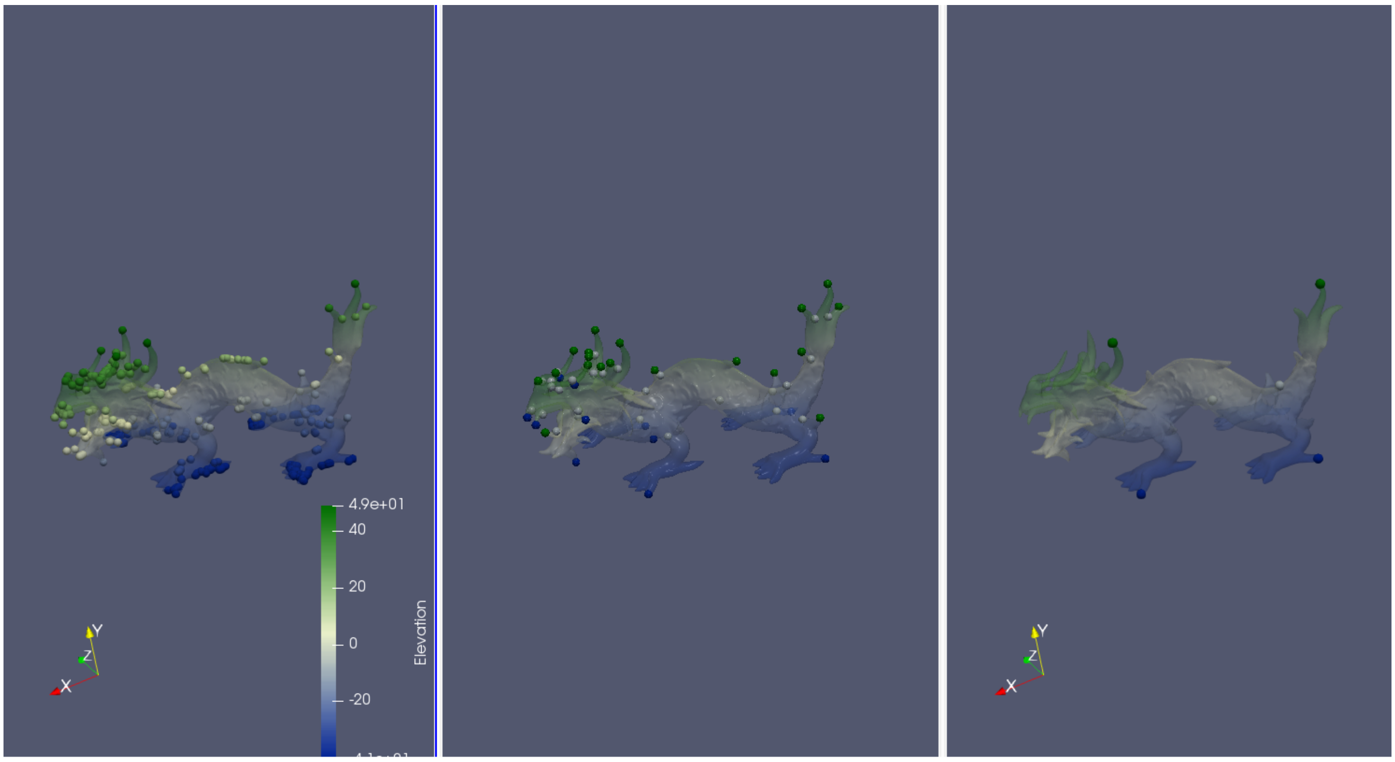


Figure 17: Top Left - A data set on a surface where the scalar value corresponds to the height of the vertex (the z-coordinate). Top Right - Critical points on the surface, many of which are spurious due to noise. Bottom Left - Automatic persistence thresholding. The red line indicates the persistence threshold obtained from the algorithm. Each black dot represents a maximum-saddle pair. We order the pairs by persistence. Bottom Right - The result after simplification.



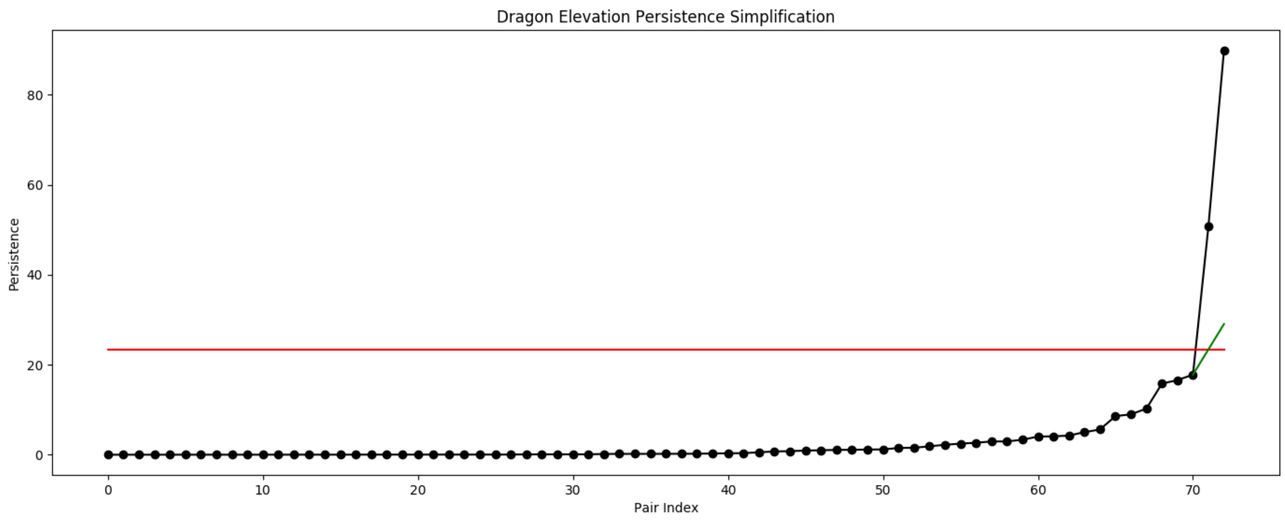


Figure 18: A data set which is a triangulation of a dragon with scalar values corresponding to height, green being high and blue being low. The spheres represents critical values, with green being maxima, blue being minima, and white being saddles. Top Left - Critical points with no simplification. Top Middle - A level of simplification which seems to retain the prominent features. Top Right - Automatic simplification which retains only two very prominent features. Bottom - The resulting thresholding algorithm on maximum-saddle pairs. We simplify by only retaining the two maxima above the red line in the output of the algorithm, leading to the top right picture.