Lappeenranta University of Technology
Department of Information Technology

Computing the persistent homology of range images with alpha shapes

(Master's thesis)

Author: Martynov Ivan, a0315930
Supervisor: Professor, Ph.D. Heikki Haario
Examiner: Researcher/Teacher, Ph.D. Tuomo Kauranne

Lappeenranta
2008

Abstract

Laser scanning is becoming an increasingly popular method for measuring 3D objects in industrial design. Laser scanners produce a cloud of 3D points. For CAD software to be able to use such data, however, this point cloud needs to be turned into a vector format. A popular way to do this is to triangulate the assumed surface of the point cloud using alpha shapes. Alpha shapes start from the convex hull of the point cloud and gradually refine it towards the true surface of the object.

Often it is nontrivial to decide when to stop this refinement. One criterion for this is to do so when the homology of the object stops changing. This is known as the persistent homology of the object. The goal of this thesis is to develop a way to compute the homology of a given point cloud when processed with alpha shapes, and to infer from it when the persistent homology has been achieved. Practically, the computation of such a characteristic of the target might be applied to power line tower span analysis.

Contents

1 Introduction

Over the last decade or so, range imaging with laser scanners has become an increasingly important tool in industrial design and manufacturing. Laser scanners can produce a point cloud that lies on the surface of even very complex an object. For scanning results to be useful in industrial design, however, such a point cloud must be converted into a surface built from polygons, such as a simplicial complex. This is a nontrivial task, because the apparent connections between neighboring points depend on our assumptions on the smoothness and other properties of the surface.

A useful tool for formulating a sequence of increasingly accurate renderings on such a surface are alpha shapes that proceed smoothly from the convex hull of the point cloud towards an increasingly fine-featured representation. Eventually, however, the points will become disconnected components each, by which time we will have gone too far. A good indicator of when the radius of alpha shapes is optimal is the persistent homology of an object. This means a set of Betti numbers that initially keep increasing and eventually stay invariant for an optimal range of alpha values. This thesis will implement an algorithm in MATLAB that computes the persistent homology in two and three dimensions and produces the stable Betti numbers. The algorithm will also render the laser scanned point cloud as a simplicial complex built from an alpha shapes representation with such an optimal alpha value.

In section 2 we describe problems which may require tools of combinatorial topology theory. The approaches to solve the tasks are mentioned as well. In section 3 we shall dive in the theory of combinatorial topology, recall its basics and a definition and features of simplicial complexes [1]. In next section we get familiar with such a topological characteristic as a target's homology and with Betti numbers as its intuitive presentation [15]. Section 5 demonstrates different techniques of an object's approximation and tells about the author of one of them. An idea of combining these approximation methods is shown. There is a persistent homology concept presented as well as one of the ways to compute Betti values of a simplicial

complex. Some combinatorial topology methods, which have been already developed, are introduced in section 6. We demonstrate our results in section 7. There we illustrate how our routines work with synthetic and real data. Finally, the last section is dedicated to conclusions and directions of probable future work.

2 Problem description

There are many industrially important modeling problems that require us to define the connectivity degree of an object, the number of holes of it or the topological structure in general. Methods to realize this lie in the combinatorial topology and they allow us to describe homology of the object by delineating it by tetrahedra, triangles and segments. There are techniques to characterize a simplicial representation of the target and they are very useful. In this thesis we show how to construct complexes and calculate the number of objects, holes and cavities. Sometimes we do not have a real image of the object but several vertices of it. We use these points to make a tetrahedralization (a triangulation in two dimensions) of the object and to get its topological structure. One of the most efficient ways to build the complex is based on the Delaunay triangulation. In the paper we represent a synthesis of alpha shapes and Delaunay triangulation. The alpha number helps us to cut unnecessary segments and thereby to delineate connected components, holes and voids of the object. The process of building simplices starts with an alpha value big enough (simply infinity, if possible) and then continues by decreasing the parameter gradually until the homology stops to change. After processing the points we eventually get the final simplicial complex and the homology of the object.

We would like to mention a couple other reasons to use the scanned vertices instead of the whole real image. First, it does not take much space from computer since the coordinates of the points are simply written in a text file. Second, it is very difficult to find the number of holes and cavities the target has and its degree of connectivity from a dense set of points. In other words, the scanned points are much better for analyzing and processing.

Nowadays, there is one technique to get a point cloud of the object and it is getting more and more popular and widely used. It is based on laser scanning. The target object is being scanned and we get the 3D coordinated of it. After that we may build a simplicial structure of the object and define its homology. There are several real tasks that can use such an approach. For example, depicting the structure

of power line tower system. The problem is to figure out how many wires it has and where they exactly go through the power towers (see Figure 1).

Figure 1: Power lines

We range a part of the power line with laser scanning and store the vertices we have got (see Figure 2). Then we apply an algorithm to make a simplicial complex of it and to compute the homology of the target, namely the Betti numbers. As the result we get a tetrahedralization of the power tower (see Figure 3).

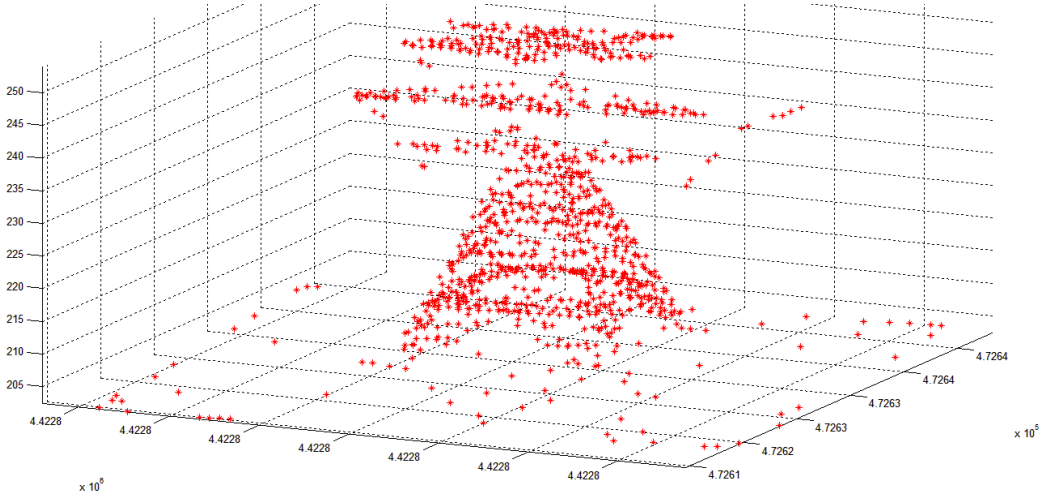

Figure 2: The vertices

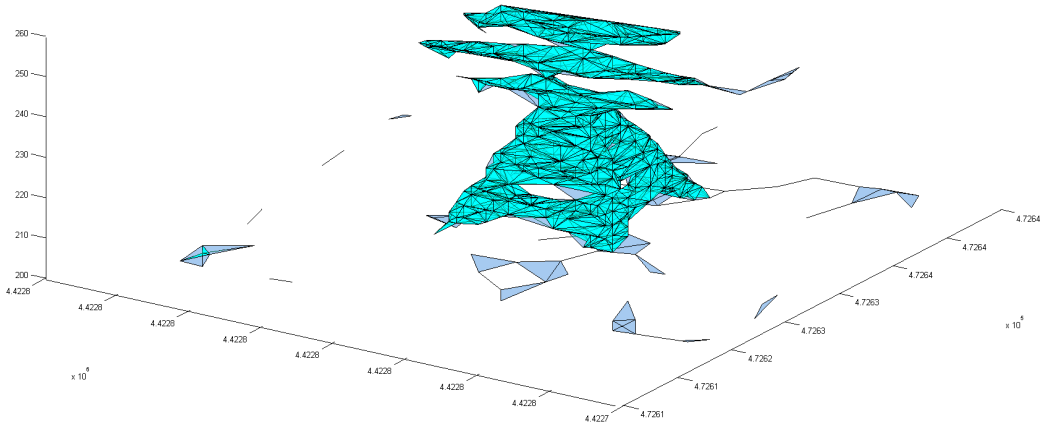

Figure 3: The tetrahedralization

3 Simplices and simplicial complexes

In this section we recall the basic definitions and other information about a simplex construction.

First, let us recall what affinely independent points are. Consider a set of points $a^0,\ a^1,\ \ldots,\ a^k$ in a space $\mathbb{R}^n$ for an arbitrary natural number n and for a value k which is between zero and n $(0 \leqslant k \leqslant n)$. These points make vectors $\overrightarrow{a^i a^j}$ which generate a linear subspace denoted by $V(a^0,\ \ldots,\ a^k)$. Since $\overrightarrow{a^i a^j} = \overrightarrow{a^i a^0} + \overrightarrow{a^0 a^j}$ then the vectors $u_1 = \overrightarrow{a^0 a^1},\ u_2 = \overrightarrow{a^0 a^2},\ \ldots,\ u_k = \overrightarrow{a^0 a^k}$ compose a system of generators (a basis) of the space $V(a^0,\ \ldots,\ a^k)$ whose dimension is thus less than or equal to k. The points $a^0,\ a^1,\ \ldots,\ a^k$ are said to be *affinely independent* ones if the dimension of the subspace $V(a^0,\ \ldots,\ a^k)$ equals k, that is if the vectors $u_1,\ \ldots,\ u_k$ are linearly independent. We have the result that any $k+1$ points $a^0,\ a^1,\ \ldots,\ a^k$ of the space $\mathbb{R}^n$ belong to a hyperplane $R(a^0,\ \ldots,\ a^k)$. This plane is constructed as an apposition of all vectors $u \in V(a^0,\ \ldots,\ a^k)$ to one of the points $a^0,\ a^1,\ \ldots,\ a^k$, for example to the point a^0. Any vector u from the space $V(a^0,\ \ldots,\ a^k)$ is a linear combination of the vectors $u_1,\ \ldots,\ u_k$. The points $a^0,\ a^1,\ \ldots,\ a^k$ are affinely independent if and only if they lie in the k-dimensional hyperplane $R(a^0,\ \ldots,\ a^k)$ and do not belong to any smaller dimensional plane.

Let us have affinely independent points $a^0,\ a^1,\ \ldots,\ a^k$ in a space $\mathbb{R}^n$ $(k \leqslant n)$. Vectors $u_1 = \overrightarrow{a^0 a^1},\ u_2 = \overrightarrow{a^0 a^2},\ \ldots,\ u_k = \overrightarrow{a^0 a^k}$ are linearly independent. Applying all linear combinations of the vectors to the point a^0 we shall get all points x of a hyperplane $R^k = R(a^0,\ \ldots,\ a^k)$ and only these points. We may write them in the following way:

$$\begin{aligned} x &= a^0 + \mu_1 u_1 + \ \cdots\ + \mu_k u_k \\ &= a^0 + \mu_1 (a^1 - a^0) + \ \cdots\ + \mu_k (a^k - a^0) \\ &= (1 - \mu_1 - \mu_2 - \ \cdots\ - \mu_k) a^0 + \mu_1 a^1 + \ \cdots\ + \mu_k a^k \end{aligned}$$

If the value $(1 - \mu_1 - \mu_2 - \ \cdots\ - \mu_k)$ is denoted by μ_0 then the point x will be written as $x = \mu_0 a^0 + \mu_1 a^1 + \ \cdots\ + \mu_k a^k$.

Hence, the hyperplane $R(a^0,\ \ldots,\ a^k)$ is a set of points which are represented as "weighted sums" of the points $a^0,\ a^1,\ \ldots,\ a^k$ and the

"weights" $(\mu_1,\ \mu_2,\ \ldots,\ \mu_k)$ are restricted by only the condition:

$$\mu_0 + \mu_1 + \ \cdots\ + \mu_k = 1.$$

Thanks to this criterion the numbers $\mu_1,\ \mu_2,\ \ldots,\ \mu_k$ are uniquely defined. And they are called *barycentric coordinates* of the point $x \in \mathbb{R}^n$ in the barycentric coordinate system composed of the points $a^0,\ \ldots,\ a^k$.

Let us have a set of affinely independent points $a^0,\ \ldots,\ a^k$ in $\mathbb{R}^n,\ k \leqslant n$ and consider a set of points $x \in \mathbb{R}^n$ such that their barycentric coordinates are positive. Then this family of the points x is called *an open k-dimensional simplex* with the vertices $a^0,\ \ldots,\ a^k$ and it is denoted by $T^k = |a^0,\ \ldots,\ a^k|$. Respectively, a family of points x whose barycentric coordinates in the system $a^0,\ \ldots,\ a^k$ are non-negative is called *a closed k-dimensional simplex* with the vertices $a^0,\ \ldots,\ a^k$ and it is denoted by $\overline{T^k} = \overline{a^0,\ \ldots,\ a^k}$. If it is clear from the context then we call the family of the points purely *a simplex*.

Figure 4 shows examples of geometrical interpretations of simplices.

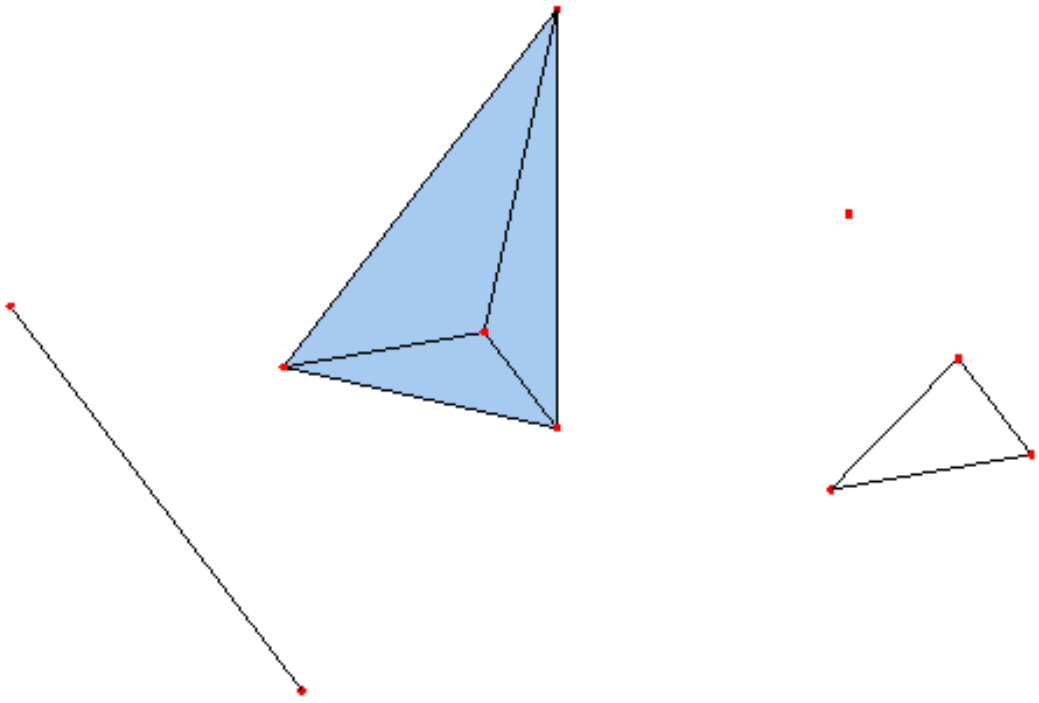

Figure 4: 0, 1, 2, 3-simplices

The 0-simplex (0-dimensional simplex) is the point, an open 1-simplex is represented by the line segment without its ends. Obviously, we shall get a closed 1-simplex by adding both ends, which are 0-simplices themselves. Next, the triangle illustrates a 2-simplex and the tetrahedron is, respectively, an interpretation of a 3-dimensional simplex.

The set of all vertices of the simplex is called its *skeleton*. It is easily seen that $\overline{T^k}$ is the closure of T^k and the simplex uniquely defines its skeleton. Let $a^{j_0}, \ \ldots, \ a^{j_r}$ be a few vertices of a k-simplex T^k. Then they are certainly affinely independent and define the simplex $T^r = |a^{j_0}, \ \ldots, \ a^{j_r}|$ in the plane $R(a^{j_0}, \ \ldots, \ a^{j_r}) \subset R(a^0, \ \ldots, \ a^k)$. The simplex T^r is called *an r-dimensional facet* of the k-simplex T^k (with the vertices $a^{j_0}, \ \ldots, \ a^{j_r}$). Also, the simplex T^r represents itself as a set of points $x \in R^k(a^0, \ \ldots, \ a^k)$ whose every coordinate j is equal to zero if j is different from $j_0, \ \ldots, \ j_r$.

We shall need to use *oriented simplices* and let us describe their definition. An *orientation* of a k-simplex is an equivalence of vertices' orderings. In other words, $T^k = |a^0, \ \ldots, \ a^k|$ is equivalent to $|a^{j(0)}, \ \ldots, \ a^{j(k)}|$ if a sign of j is 1 and vice versa if $\mathrm{sign} j = -1$.

Here we come to the point to introduce simplicial complexes. Intuitively they can be imagined as a set (a complex) of simplices. And, in fact a simplicial complex is indeed a family of open simplices. A simplicial complex may fulfill to one or more additional conditions and the complex is then accompanied by the corresponding adjective. For example, we consider only finite simplicial complexes in this thesis. That is, the complexes consisting of finite sets of simplices. The complex K is called *an n-dimensional* one if all its simplices have dimensions no greater than n and at least one simplex has the dimension n. A complete complex K assumes that every facet of a simplex of K is itself an element of the complex. In fact, we deal with finite complete simplicial complexes whose elements are disjunctive oriented simplices of the given space $\mathbb{R}^n$ and a complex of such type is called *a triangulation*

4 Homology and Betti numbers

In order to introduce the homology we start from a chain complex. Let K be a simplicial complex (a triangulation, further we shall sometimes skip this characterization) and C_k is said to be the k^{th} *chain group* of K and it represents a family of oriented k-simplices. For example, C_2 can be simply imagined as all triangles in the complex K.

We call an element of C_k a *k-chain* if it is got from the following formula: $\sum\limits_j n_j T^j$ where n_j are integer terms and $T^j \in K$.

Let us introduce the *boundary operator* which maps from C_k to C_{k-1} and appears to be a homomorphism. The boundary operator $\delta_k \colon C_k \to C_{k-1}$ is linearly defined on a k-chain c by the following formula:

$$\delta_k T = \sum_j (-1)^j |a^0,\ a^1,\ \ldots,\ \hat{a^j},\ \ldots,\ a^k|$$

where $T = |a^0,\ \ldots,\ a^k| \in c$ and $\hat{a^j}$ denotes the deleted element from the sequence. Eventually, the boundary operator helps us to accumulate all chain groups into a *chain complex* C:

$$\cdots \to C_{k+1} \xrightarrow{\delta_{k+1}} C_k \xrightarrow{\delta_k} C_{k-1} \to \cdots .$$

After defining the boundary operator we characterize two subgroups of the k^{th} chain group, and namely the *cycle group* $Z_k = \ker \delta_k$ and the *boundary group* $B_k = \operatorname{im} \delta_{k+1}$ [12]. We would like to mention a very important property of boundary operators which says that $\delta_k \delta_{k+1} = \emptyset$ always. In other words, it can be understood in the way of the idea that a boundary does not have a boundary. Let us illustrate this property. Consider a simple complex consist of a tetrahedron, its triangles, the lines and the vertices, so we take a simplex $T = |a,\ b,\ c,\ d|$ and apply the boundary operators δ_3 and δ_2 on it. The process looks as follows:

$$\begin{aligned}\delta_2(\delta_3(T)) &= \delta_2(|b,\ c,\ d| - |a,\ c,\ d| + |a,\ b,\ d| - |a,\ b,\ c|) = \\ &= |c,\ d| - |b,\ d| + |b,\ c| - |c,\ d| + |a,\ d| - |a,\ c| + \ldots \\ &+ |b,\ d| - |a,\ d| + |a,\ b| - |b,\ c| + |a,\ c| - |a,\ b| = \emptyset.\end{aligned}$$

An idea of a proof for a general case is described below. Let us have a simplex $T = |a^0,\ \ldots,\ a^k|$ and apply an operator $\delta_k\delta_{k+1}$ to it: $\delta_k\delta_{k+1}(T)$. An element of the result might be represented as

$$(-1)^p|a^0,\ \ldots,\ a^{i-1},\ \hat{a^i},\ a^{i+1},\ \ldots,\ a^{j-1},\ \hat{a^j},\ a^{j+1},\ \ldots,\ a^k|,$$

that is the entries $\hat{a^i}$ and $\hat{a^j}$ were deleted. And we have exactly two elements with this structure with the fixed i and j (let us say that $i < j$), however these elements have different p powers. If the i^{th} vertex was deleted first, then the index of the j^{th} vertex decreased by one. Therefore, the first p equals $i + j - 1$. For the second element the j^{th} vertex was eliminated first and, consequently, the index of the i^{th} vertex did not change. Hence, the second p is equal to $i + j$. It means that the elements have different signs and eliminate each other, since they have absolutely the same vertices.

Now, we finally come to the point to introduce homology. It is defined by its groups. Precisely, the k^{th} *homology group* is a quotient space Z_k mod B_k, that is $H_k = Z_k/B_k = \ker\delta_k/\mathrm{im}\,\delta_{k+1}$. The rank of the k^{th} homology group is said to be the k^{th} *Betti number* of the complex and we shall denote it by β_k.

Intuitively, Betti numbers can illustrate some simple geometrical elements of objects [10]. We consider only the first three Betti numbers. The zeroth Betti number represents the degree of connectivity of an object: how many connected components belong to the object. You may think about the first Betti number as a value which shows how many holes the target has. And the second Betti number is to be understood as a quantity of voids of the target object. The voids are cavities we do not see, like we do not see an emptiness inside a ball or in a bicycle tube, they are kind of 2-dimensional holes. Since there are no cavities in a plane the second Betti number is zero when we are in $\mathbb{R}^2$, as well as the first Betti number in the one-dimensional case.

5 Alpha shapes and the persistent homology

In this section we describe a technique of making a triangulation of a set of points. The definition of alpha shapes and the persistent homology is given as well. We use synthesis of the alpha shape algorithm and the Delaunay triangulation routine to compute the persistent homology.

5.1 Delaunay triangulation

In order to make a proper triangulation (a tetrahedralization in three dimensions) of an object the Delaunay technique is usually used. This algorithm is reliable and produces nice triangulations avoiding narrow triangles. Besides, it is fast to compute and there are software implementations available. The technique of triangulating a set of vertices was invented by the russian mathematician Boris Delaunay (see Figure 5) in 1934. Let us describe history of this corresponding member of the Academy of Sciences of the USSR.

Delaunay was a child of a professorial family, and he continued this tradition. Being a son of a mathematician he brought up a physicist, however his grandchild became a poet and a human rights defender. Boris successfully graduated from the Kiev University where he started to produce results in number theory and algebra. Considering the scientific activities of Delaunay, we would like to say he made important works in different areas of mathematics. In the number theory he had results in theory of the indefinite equation of the third degree with two undeterminates.

With taking a glance in algebra we find a geometrization of Galois theory. Most of Delaunay's works lie in geometry. He has developed the theory of regular space partitions, the theory of reduction of quadratic forms, the theory of lattice covering of a space by spheres, the theory of stereohedra and mathematical crystallography.

As every great scientist, he was fond of not mathematics only but alpinism as well. He loved mountain journeys since childhood. Every summer the family got round to Switzerland. It was actually

cheaper to go abroad than to travel in Caucasus in those years. He has made his first ascension in 1903 when he was thirteen years old. It happened in Cinque Torri (Five Towers, 2361 meters), the place near the famous village Cortina d'Amprezzo. Cinque Torri represents itself a small complex of middle peaks and is considered as an ideal place for apprentices in mountain climbing. This place is famous also by the fact that it was a battle-ground in the Great War and you still can find helmets in Alpine taverns which store also cartridge cases, military uniforms and rusted rifles of those times. Mountains became a part of Boris' life and he decided to be an alpinist, and in fact it was not an obstacle for him continuing his research in mathematics.

Figure 5: Boris Delaunay

There were only three professional mountain climbers in Russia before the revolution in 1917. They were Panyutin, Golubev and Delaunay. Boris was one of the first sportsmen who got the sport master degree in mountain climbing and one of the founders of Soviet alpinism. However, mountain tourism led him later with many famous people such as Igor Tamm (a member of the Academy, Nobel laureate

in physics), his son Eugene Tamm (a professor, famous climber and the captain of the first Soviet Himalayan expedition to conquer Chomolungma in 1982), A. Alexandrov (a member of the Academy, great mathematician) and others. Delaunay wrote the book "The Peaks of the Western Caucasus" and thereby confirmed the famous words of Karl Weierstrass: "It is true that a mathematician who is not also something of a poet will never be a perfect mathematician". Probably, this is even more impressively reflected in his grandson. And Delaunay quite successfully combined climbing with being a professor of Leningrad and Moscow Universities.

And now we come back to geometry and the algorithm of a Delaunay triangulation. Let us say we have a set of points X. Then a Delaunay triangulation is a triangulation (remind that a triangulation is a finite complete simplicial complex) such that there are no vertices from X inside the circumcircle of any triangle (see Figure 6). In the 3D case we replace the words circumcircle and triangle by circumsphere and tetrahedron. The circumcircle of a triangle is said to be empty if it does not contain points except the three points from X that form this circumcircle. The points on the border of the circumcircle are the ones allowed. However, if there are four points on the same circle then the Delaunay triangulation is not unique. And a Delaunay triangulation does not exist if all points are on one line, because there is no triangulation at all.

A Delaunay triangulation might be generalized for other metrics, however we can not ensure that a Delaunay triangulation exists or is unique then. A generalization for the n-dimensional Euclidean space exists and says that there are no points from X inside the circum-hypersphere of any simplex of a triangulation. The condition for the uniqueness of the Delaunay triangulation is represented as follows: all points of the set X are in the *general position*. It means that there are no $n+1$ points on one hyperplane and no $n+2$ points on the same hypersphere. In two dimensions the meaning of the general position is simple: there are no three points on the same line and no four points on one circle.

The proof of this fact is not difficult and the idea is the following. If we have a point cloud X in the n-dimensional Euclidean space then we

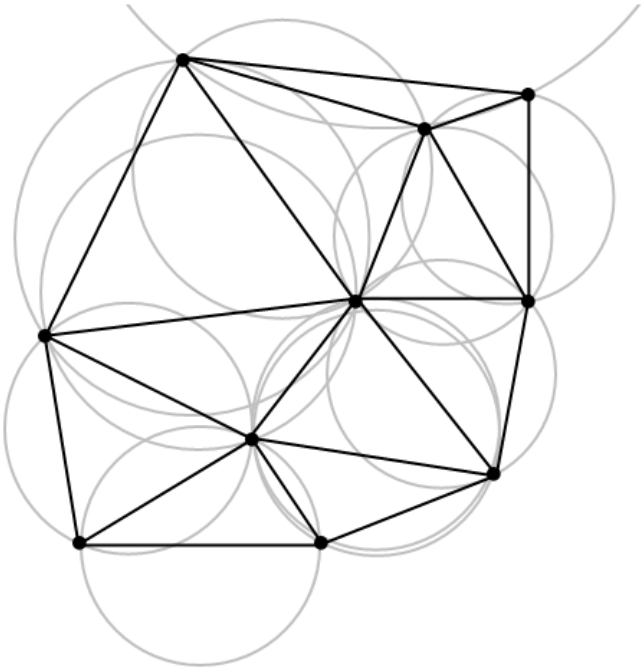

Figure 6: Delaunay triangulation

extend it into the $n+1$-dimensional one by putting the last coordinate equals $|x|^2 = \sum_i x_i^2$ for each $x \in X$. Then we construct the convex hull of this new set of points and it is well known that it is unique for a set of points. Thus, we project the convex hull in n dimensions excluding the upper part of the hull (see Figure 7) and, consequently, we get the unique triangulation of X (we assume that all facets of the hull present themselves as simplices).

The Delaunay triangulation is perfectly implemented in the Qhull software tool [2] and it is embedded in MATLAB. There are several routines in MATLAB based on the Qhull algorithms. They are delaunay (to triangulate a set of points), delaunayn (might be applied for dimensions from three to nine), convhulln, voronoin, griddatan and others.

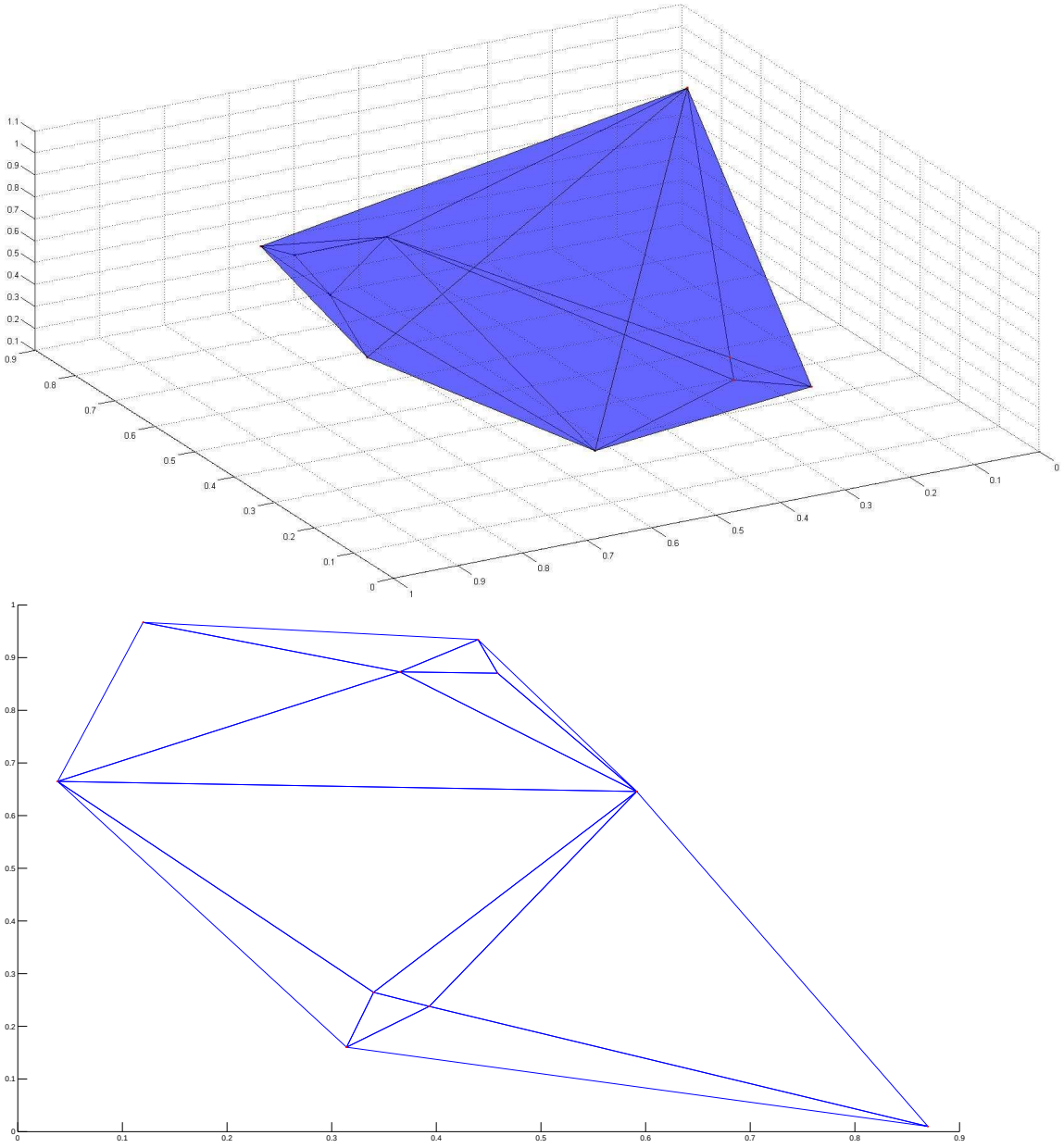

Figure 7: Projection of the bottom part of the convex hull

5.2 Alpha shapes

When we are talking about alpha shapes we first take a set of points into account. For this set we may construct a convex hull which is in most cases not enough to get sufficient information. We want to get a more precise approximation of an object the points represent. The alpha shape of the cloud of the points may be considered as a generalization of the convex hull. The word alpha refers us to the following idea: we use a real parameter which is in fact a variable. Alpha can be shown either as a limit for a distance between points or a radius of a circle. In the first case, if the distance is less than alpha then these points are connected. In the second case, if points are inside of the circle then they are to be connected as well. If we take the value big enough then it will produce exactly the convex hull of the object. And it is often taken as the starting point to make an alpha shape. After this, the alpha value is being gradually reduced to refine the approximation. After several steps (most likely already after the first one) the alpha shape stops to be convex and at some point it can even become disconnected to represent few components of the object. However, if alpha is very small, such that there are no points to be connected, then the alpha shape of the point cloud is empty. It is also a successful result which is in fact the point cloud itself, every point is a separate topological component.

Now let us give the formal definition of alpha shapes. We take a set of points X and consider a positive value α ($0 < \alpha < \infty$). We establish an *α-ball* as an open ball with a radius that equals α. For convenience and completeness we determine a 0-ball as a point and an ∞-ball as an open half-space. An α-ball B is said to be *empty* if its intersection with the set of the points is empty: $B \bigcap X = \emptyset$. Every subset Y of X with a size value that equals $k + 1$ for $k = 0,\ 1,\ 2,\ 3$ represents a k-simplex T^k, which is indeed the convex hull of Y. Assuming that all points of X are in the general position we ensure that all k-simplices are "properly" k-dimensional ones. Let us consider the values of k such as $0 \leqslant k \leqslant 2$. Then we call a k-simplex T^k an *α-exposed* one if an α-ball B exists with $Y = \partial B \bigcap X$, where ∂B is a boundary of B in the form of either a sphere or a plane. Eventually, a fixed value α defines sets $S_{k,\alpha}$ of α-exposed k-simplices for $0 \leqslant k \leqslant 2$. The *$\alpha$-shape*

of X is then the polytope whose boundary is built of the triangles $S_{2,\alpha}$, the segments from $S_{1,\alpha}$ and the vertices which come from $S_{0,\alpha}$. The α-shape is denoted by X_α. It might also be useful to mention that the k-simplices $S_{k,\alpha}$ are called k-faces of X_α as well.

In this paper we consider alpha complexes which are indeed simplicial complexes based on the techniques of alpha shapes and a Delaunay triangulation. Thus, we take a set of points and make a tetrahedralization (in three dimensions) or a triangulation (in two dimensions) of the cloud. In order to make an alpha complex of the family of the points we keep only those 1-simplices from the Delaunay triangulation (tetrahedralization) whose vertices have a distance between them less than alpha [4].

The first value of alpha might be taken as infinity or an adequately big number to get the convex hull. Next, after correcting the Delaunay triangulation, we compute the Betti numbers of the built complex, which in fact represents a chain complex by all its tetrahedra, triangles, segments and vertices. On the following step, we decrease the alpha value, rebuild the simplicial complex and recalculate the Betti values. We continue reducing our parameter until the Betti numbers will stop changing, unless we reach the point which equals zero or too small alpha number. If the refinement is successful we get the proper Betti values and here we achieve the idea of persistency.

5.3 The persistent homology

The *persistent homology* is a topological invariant which is the same among homeomorphic objects. And in real computations it is represented by the Betti numbers of a simplicial complex which tetrahedralize (triangulate) an object by the set of its vertices. Here it is very important to choose how to change alpha, because if the progression is too slow we may decide to stop the process too early and if it is too big we may jump to a very small value rather fast. In both cases we lose persistency. However, as far as we find a rather wide range of alpha values that have identical Betti numbers we may say that it is a stable alpha scope. Such the stable range is a good indication that the target's persistent homology has been reached.

Now we shall describe an algorithm for computing Betti numbers. It is called the *reduction algorithm* [14]. Let us take a complex K and the k^{th} chain group C_k. The k-simplices will form the standard basis for C_k. The boundary operator $\delta_k \colon C_k \to C_{k-1}$ then can be represented through the standard bases of the chain groups as a matrix M_k whose elements are $\{-1, 0, 1\}$. The matrix is said to be the *standard matrix representation* of δ_k. The number of rows of M_k is a number of $(k-1)$-simplices and it is denoted by m_{k-1}. Respectively, the symbol m_k is reserved for the number of columns of the matrix and shows how many k-simplices we have. The cycle group of C_k is $Z_k = \ker \delta_k$ and it is represented by the null-space of the standard matrix representation; and the range-space of M_k corresponds to $B_{k-1} = \operatorname{im} \delta_k$, the boundary group of C_{k-1}.

The reduction algorithm uses simple row and column operations on the matrix M_k in order to derive the matrix for δ_k to a diagonal form. Each row operation leads to changes in the basis for C_{k-1} and the basis of C_k is modified whenever the algorithm applies a column operation. For instance, if we take the i^{th} and the j^{th} $(i \neq j)$ elements of the C_k basis (e_i and e_j, respectively) into account, then we may replace, for example e_i with $e_i + q \cdot e_j$, where q is an integer number. Gradually, we reduce the matrix M_k to its normal form.

$$\tilde{M}_k = \left(\begin{array}{ccc|c} b_1 & & 0 & \\ & \ddots & & 0 \\ 0 & & b_{r_k} & \\ \hline & 0 & & 0 \\ & & & \end{array}\right)$$

In this normal form representation r_k is for the rank of the matrix: $r_k = rank(M_k) = rank(\tilde{M}_k)$, the ranks of the original matrix and the reduced one are equal. Every b_j is greater or equal than one $(b_j \geqslant 1)$ and we have got one more feature for all b_i-s: $b_i | b_{i+1}$ $\forall i$ such that $1 \leqslant i < r_k$. And now we extract all information about the k^{th} homology group H_k:

- Those elements b_i whose values are greater than one are the torsion coefficients of H_k;
- The rank of the group Z_k is calculated as $m_k - r_k$ since $\{e_j \mid r_k + 1 \leqslant j \leqslant m_k\}$ is a basis for Z_k. Remind that Z_k represents the kernel of the operator δ_k;
- Similarly, $\{b_j \tilde{e}_j \mid 1 \leqslant j \leqslant r_k\}$ performs a basis for the boundary group B_{k-1} and consequently, $rank(B_{k-1}) = rank(M_k) = r_k$.

Combining the last entries we get a formula for the k^{th} Betti number: $\beta_k = rank(Z_k) - rank(B_k) = m_k - r_k - r_{k+1}$ [3]. Now let us specify the formulas for first three Betti numbers.

For the zeroth Betti number we shall have m_0 — a number of vertices (0-simplices), r_0 — rank of the matrix M_0 and the rank of M_1 is represented by r_1. However, the matrix M_0 has $m_{0-1} = m_{-1}$ rows and, namely, m_{-1} of the -1-simplices or the empty sets. It means that m_{-1} is equal to zero and $M_0 = \emptyset$ whose rank is zero. Therefore, the formula for the zeroth Betti number looks like $\beta_0 = m_0 - r_1$.

For the first Betti number we first consider two dimensions. In this case, as we know, the second Betti number is zero and the formula for it is next one: $\beta_2 = m_2 - r_2 - r_3 = 0$. Here the parameter r_3 performs the rank of the matrix M_3 whose number of columns is the number of the 3-simplices or the tetrahedra. Obviously, there are no tetrahedra in 2D space. Consequently, the rank is zero and we have the equation $m_2 - r_2 = 0$ which heads to the result that the rank of the matrix M_2 equals the number of the 2-simplices (the triangles). We apply this result to the formula for the first Betti number. Precisely, $\beta_1 = m_1 - r_1 - m_2$.

In the case of three dimensions there is no refinement for the first Betti number, the formula is the following $\beta_1 = m_1 - r_1 - r_2$, because there are might be tetrahedra. However, we change the formula for the second Betti number. The idea is the same: the third Betti number is zero and its formula helps us to compute the rank of the matrix M_3. In details, $\beta_3 = m_3 - r_3 - r_4 = 0$ and $r_4 = 0$ because there are no 4-simplices in the 3D space. Hence, $r_3 = m_3$, the rank of M_3 is the number of the tetrahedra we have. Finally, the formula for the second Betti number is $\beta_2 = m_2 - r_2 - m_3$.

We even can do the same reasonings for the elementary 1D case and get the result that the first Betti number is equal to the difference between the number of vertices and the number of segments and other Betti numbers are zeros. And, generally, for the $(n+1)$-dimensional Euclidean space the formula for the n^{th} Betti number is
$\beta_n = m_n - r_n - m_{n+1}$ and $\beta_0 = m_0 - r_1$ for the first Betti value. Let us underline the results for the Betti numbers for three different dimensions.

1. One dimension: $\beta_0 = m_0 - m_1,\ \beta_1 = \beta_2 = 0$;
2. Two dimensions: $\beta_0 = m_0 - r_1,\ \beta_1 = m_1 - r_1 - m_2,\ \beta_2 = 0$;
3. Three dimensions: $\beta_0 = m_0 - r_1,\ \beta_1 = m_1 - r_1 - r_2,$
   $\beta_2 = m_2 - r_2 - m_3$.

6 Software to compute the persistent homology

On the web it is possible to find many different codes and applets which are dedicated to making a triangulation or a tetrahedralization of a set of points or an object. However, there are not so much developed tools for computing the persistent homology of the target. We would like to mention some of them. In this part of the thesis we present different techniques which were exploited with the purpose to compute or/and to illustrate the persistent homology. All of them process a set of points in two or three dimensions.

6.1 Ashape: a pedestrian alpha shape extractor

Ashape represents itself as a simple wrapper for a family of routines implemented in MATLAB by people who work in this company. These routines are combined together in the Aslib.m file and serve for extracting and displaying 2-dimensional alpha shapes and alpha patches from a set of data points. Aslib gives a construction which has all relevant parameters, data and results holding graphics handles as well to plotted elements. However, Aslib records handles of a function to every subroutine and it can be used to make more wrappers. Thus, it is open for further development.

To illustrate the process implemented in these routines we use a set of points found in the Internet and they delineate a teapot (see Figure 8). First we take original number of points and apply the procedures to them. The process starts to define a shape of the target using α-circles which are 2-dimensional α-balls (see Figure 9). On the next step it continues refining the set of the points and we may see this intermediate result in Figure 10. In the picture we see three different color bars which represent frames of some parts of the object. And the final result is presented in Figure 11. There are two rows with different colours. The first row performs the frames of the separated parts of the target and the second row illustrates the colours of the patches which are bounded by these frames.

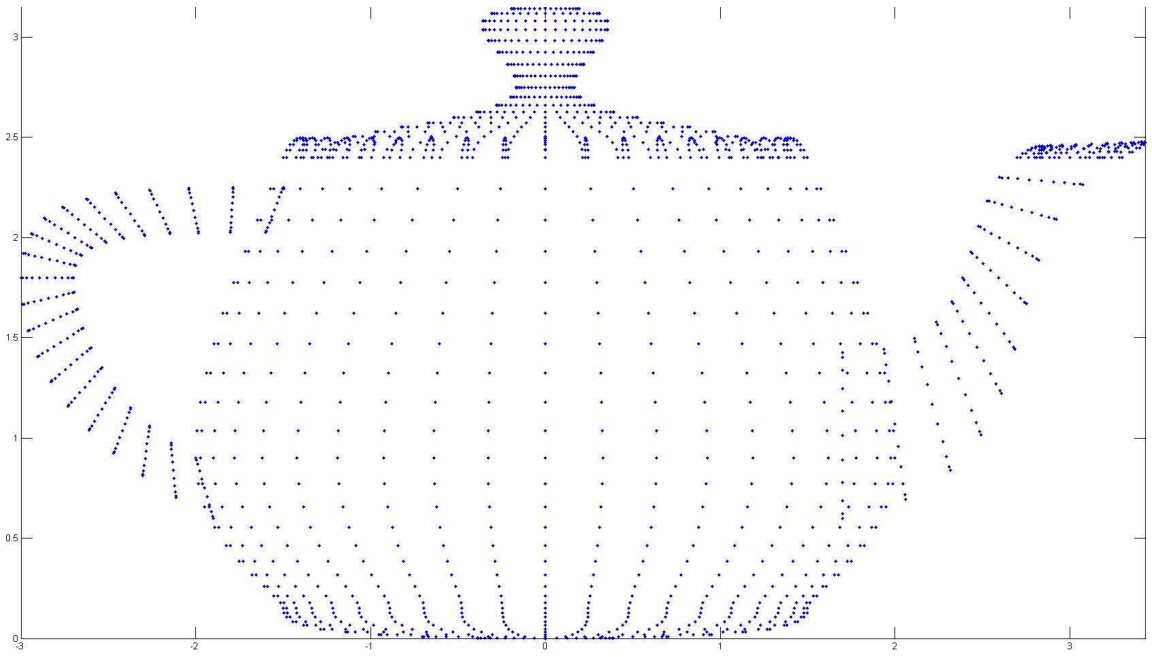

Figure 8: The teapot points

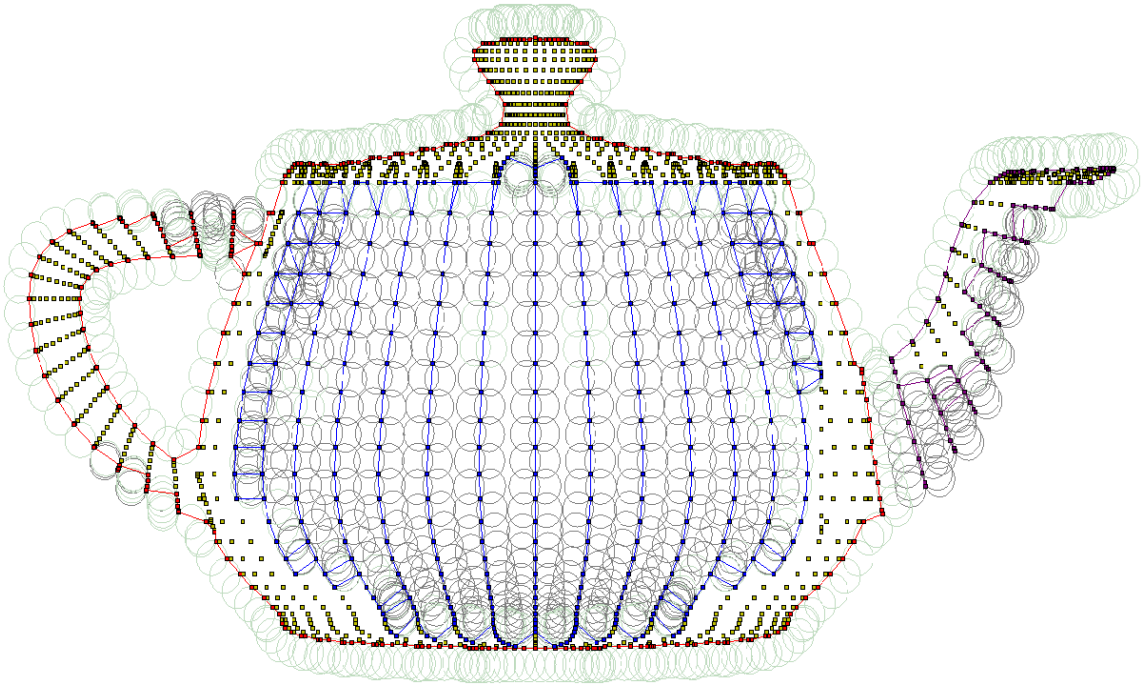

Figure 9: The beginning of the process

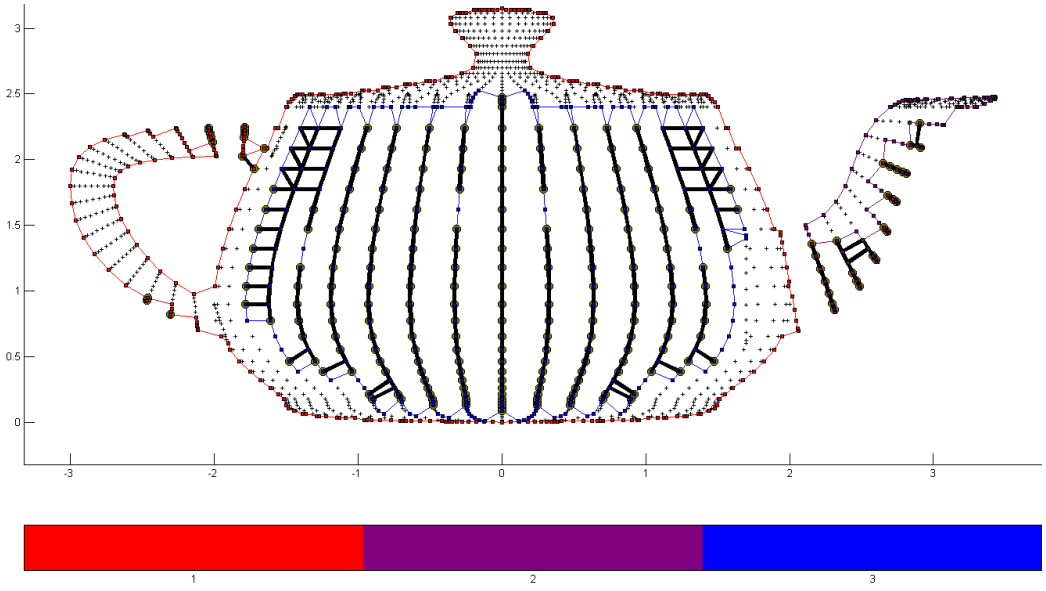

Figure 10: Refinement

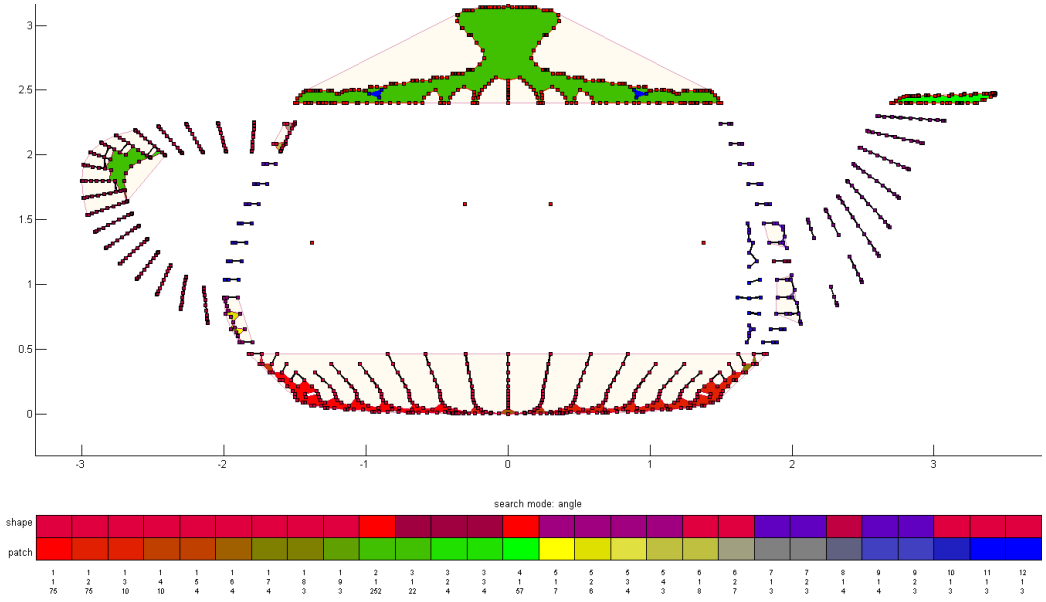

Figure 11: The result

As you can see, the result is not good because it returns that our points produced many separate objects whereas we know that it should yield one object with one hole.

Now we produce more points to get a better result. Originally, we have 3644 points and using one technique, which will be described later (see p. 35), we produce 42332 vertices (see Figure 12) of our teapot. And now let us see the stages of the processing of the device point cloud.

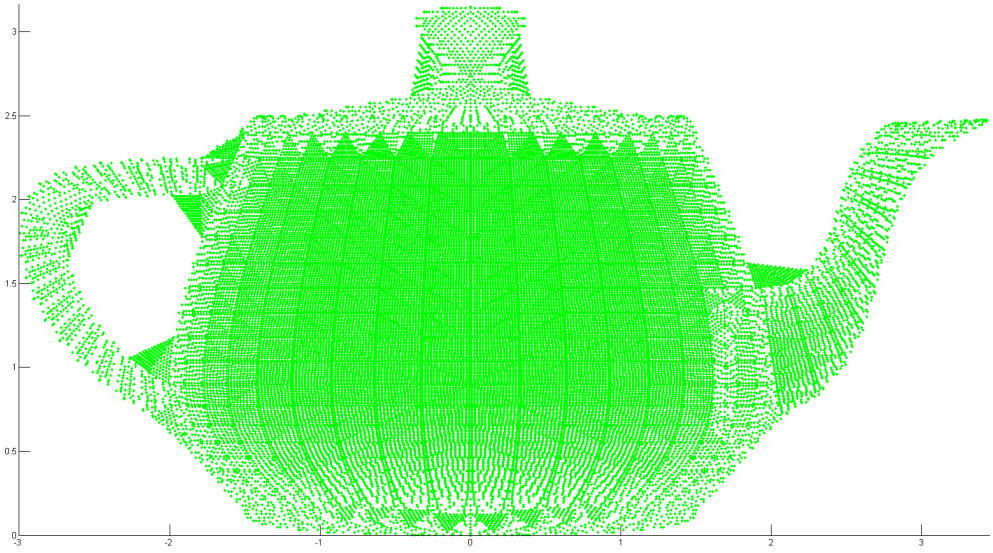

Figure 12: The kettle points

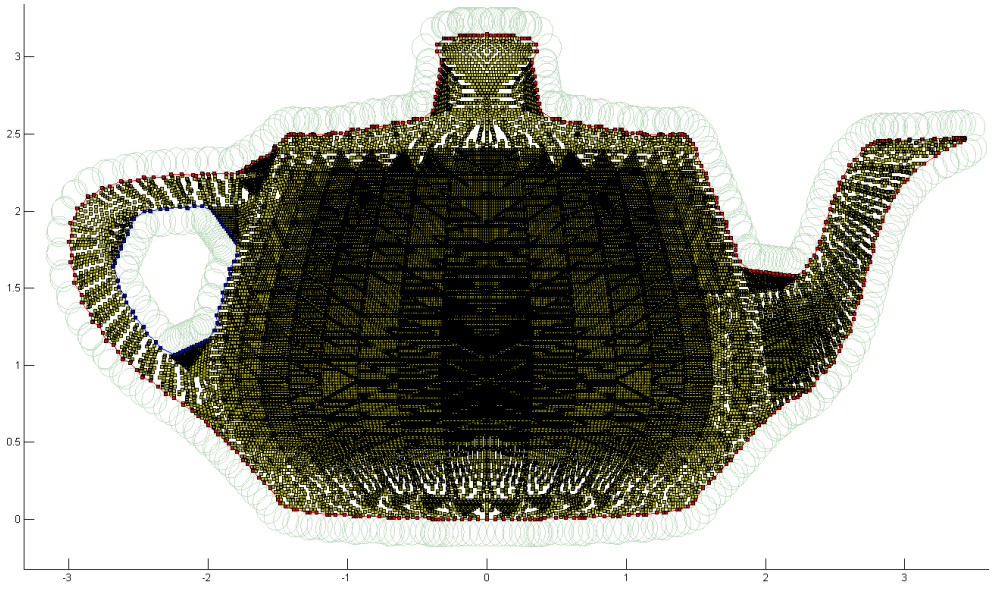

Figure 13: The beginning of the process

The procedure is the same: it uses α-circles for processing the vertices (see Figure 13). Then refinement of the point cloud continues the implementation and is shown in Figure 14. The final result is illustrated in Figure 15.

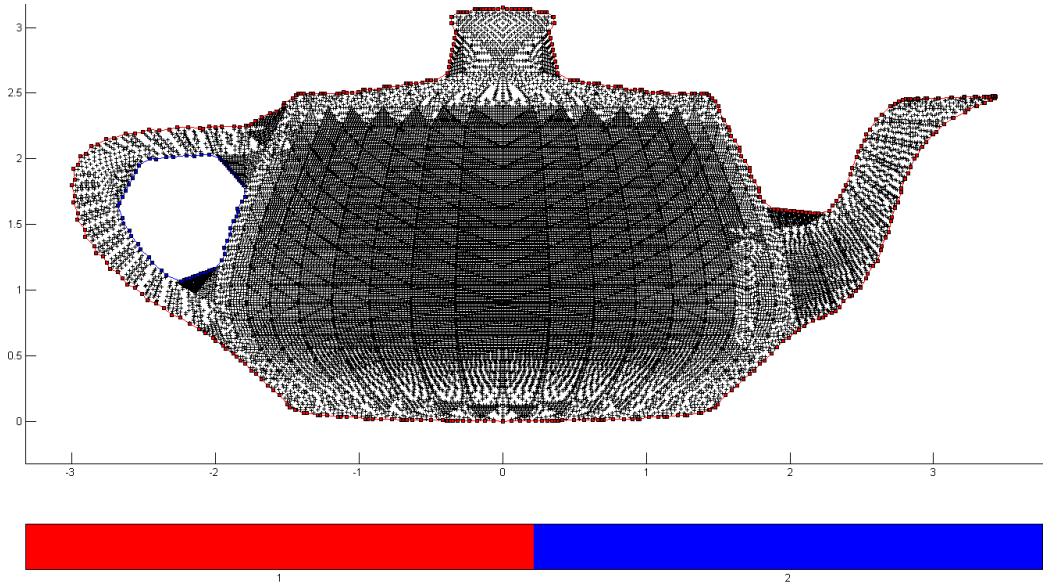

Figure 14: Refinement

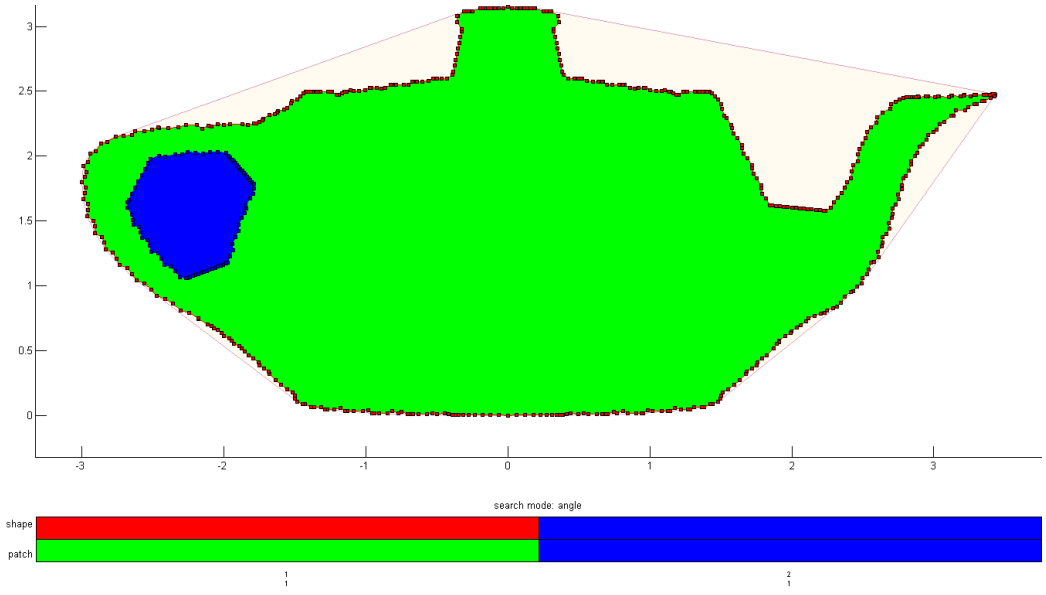

Figure 15: The result

Here it is easily seen that the result is much better and even more, it returns the correct representation of the target: one object which has only one hole. The comparison we made was invoked to show that you really need a lot of points to get the proper result using Ashape and Aslib. It is also necessary to record the computation time we have spent in both cases. In the first time around 14 seconds were needed to finish the computations, and the corresponding result in the latter case was around 635 seconds (ten and a half minutes). Considerably good timing for 42332 vertices indeed.

If the reader desires to download and use this set of routines we refer you to [11].

6.2 PLEX — simplicial complexes in MATLAB

PLEX presents itself as a set of routines written in MATLAB and C++. They were developed by a group of researches at Stanford University from 2000 to 2006. The functions and procedures were designed and coded by Vin de Silva mostly. Dawn Banard (nowadays Dawn B. Woodard) has made mod 2 methods, which are all written in C and embedded in MATLAB using the corresponding MEX functions. She, Patrick Perry and Peter Lee have incorporated several valuable improvements.

Being a MATLAB library PLEX might be considered as a MATLAB toolbox whose prime purpose is computing homology and the persistent homology. The idea of solving such tasks is the following: a space X is to be homeomorphic to a simplicial complex and the homology of the space is being computed by recovering its Betti numbers. Therefore, the toolbox first constructs the simplicial complex using the data points of the space X as the vertices of the complex. After building this structure, PLEX routines allow us to compute the Betti numbers of the triangulation (the tetrahedralization) of the space X.

The approach to construct a simplicial complex is based on covering the space X by open sets U_i which are good in the following sense: they are all contractible as are the finite intersections

$U_{i_0} \bigcap \ldots \bigcap U_{i_k}$. Then we construct a complex with an i^{th} vertex for every non-empty U_i and with a k-cell for a set of $(k+1)$ indices $\{i_0,\ \ldots,\ i_k\}$ such as $U_{i_0} \bigcap \ldots \bigcap U_{i_k} \neq \emptyset$.

The authors of PLEX describe an ϵ-approximation to build a simplicial complex homeomorphic to the space X as one of possible ways to do this [8]. A set of points, which represents this space, is taken to make a complex. For generality, let us assume that the space X is in the N-dimensional Euclidean space $\mathbb{R}^N$ and we have n points (every point $p_i \in \mathbb{R}^{\mathbb{N}}$). In the beginning of building we create a space P_ϵ which consists of the points and open balls $B_i\left(\frac{\epsilon}{2}\right)$ with their centers in the points p_i. The open balls make a good covering of the space P_ϵ and we use it to create a complex. In fact, the ϵ-approximation technique to make a complex is a composition of alpha shapes and a triangulation and we would call it either an α-approximation or ϵ-shapes.

The PLEX creators describe this method, however they have another one which has been implemented in the routines. The complex $C_\epsilon(P)$ is being built with every data point p_i and k-cell for each $\{p_0,\ \ldots,\ p_k\}$ such as the distance between every pair of points in this cell is less than ϵ. That is, $\parallel p_i - p_j \parallel < \epsilon$. As you remember, the skeleton of the simplicial complex uniquely defines its structure. Consequently, it is computationally cheaper to store the 1-simplices only and restore the full complex whenever you need.

In order to use the toolbox PLEX you have to do a couple of simple procedures. Namely, you need to download the zip archive plex-2.0.1-windows.zip from the web page of PLEX on your computer and extract it in some directory [9]. The next step is adding the two following catalogues into the MATLAB search path:
plex-2.0.1-windows/matlab/Plex/ and
plex-2.0.1-windows/matlab/Plex/metric/.

It works like this. In the main window in MATLAB you choose `File-Set Path...` in the menu. You proceed by clicking the button `Add Folder...` and put the mentioned above catalogues into the path. And then you simply check whether it was successful or not by typing the command `plex` in the MATLAB command window. To get familiar with PLEX the authors offer to start making simplicial

complexes by hand and the first task is to build a complex to display the word HELLO [6]. Here we show letters which are presented by their complexes and constructed by hand. Let us consider the letter A first. In order to present it in a form of a simplicial complex we need to define a number of vertices and their coordinates. First we draw the letter on a piece of paper and after we figure out that we need 11 vertices. Thus, we need to start building our complex from 0-simplices. There is the function

$$C \texttt{ = attach}(C1, C2)$$

in the toolbox. It assembles two simplicial complexes into one. Therefore, we apply this routine: C `= attach(plex,1:11)`.

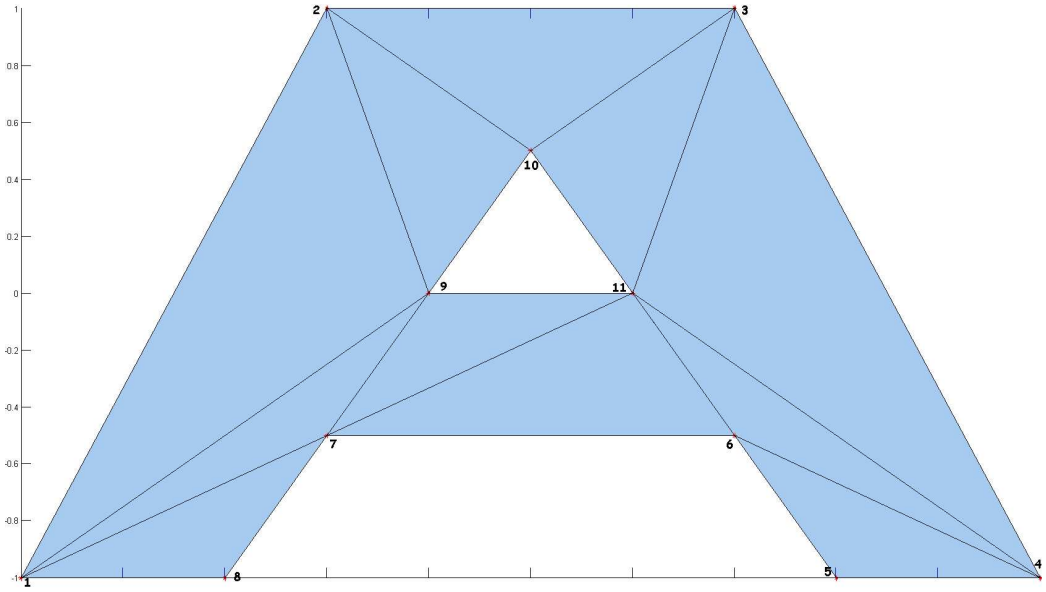

Figure 16: The letter A

From Figure 16 we may see which vertices are to be connected. Consequently, we add indices of the segments into the complex C: C `= attach(`$C, s1$`)` where

$$s1 = \begin{pmatrix} 1 & 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 & 4 \\ 2 & 7 & 8 & 9 & 3 & 9 & 10 & 4 & 10 & 11 & 5 \\ 4 & 4 & 5 & 6 & 6 & 7 & 7 & 7 & 9 & 9 & 10 \\ 6 & 11 & 6 & 7 & 11 & 8 & 9 & 11 & 10 & 11 & 11 \end{pmatrix}.$$

The same idea is kept for 2-simplices. We attach $s2$ to the complex: C `= attach(`$C, s2$`)` where

$$s2 = \begin{pmatrix} 1 & 1 & 1 & 2 & 2 & 3 & 3 & 4 & 4 & 6 & 7 \\ 2 & 7 & 7 & 3 & 9 & 4 & 10 & 5 & 6 & 7 & 9 \\ 9 & 8 & 9 & 10 & 10 & 11 & 11 & 6 & 11 & 11 & 11 \end{pmatrix}.$$

The picture was produce by the PLEX routine

$$\texttt{plot3complex}(C, X)$$

where X represents the coordinates of the vertices and equals

$$\begin{pmatrix} x & 0 & 0.3 & 0.7 & 1 & 0.8 & 0.7 & 0.3 & 0.2 & 0.4 & 0.5 & 0.6 \\ y & -1 & 1 & 1 & -1 & -1 & -0.5 & -0.5 & -1 & 0 & 0.5 & 0 \end{pmatrix}.$$

The coordinates were taken by hand as well.

One detail we want to indicate is that you do not have to construct all simplices but the 2-simplices are needed. If we invoke the function `attach` to make the complex C `= attach(plex,`$s2$`)` then we get the same result, because other simplices are being automatically constructed. However is works in the only case when we have no 1-simplices and 0-simplices which are facets of the trianlges.

The toolbox PLEX has the built-in function `betti(`C, n`)` which returns an n^{th} Betti number of the complex C. After applying it to our letter simplicial complex we have got the results $\beta_0 = 1$ and $\beta_1 = 1$ which completely satisfies us.

We have coded several letters in MATLAB using the toolbox in order to be able to construct words in future. We assembled the characters in one routine

$$\texttt{[}C\ X\texttt{] = choose2DLetter}(L)$$

which returns a complex C and coordinates X for the letter L. It has been made to practise building simplicial complexes and there is no need to make the whole alphabet unless someone wants to apply the letters for some reason. Each character's simplicial complex is constructed by specifying coordinates of its points and by labeling the vertices. The procedure `[`$C\ X$`] = choose2DLetter(`L`)` has been coded in the following way:

```
switch L
   case 'A'
      [C X] = A2D;
   case 'B'
      [C X] = B2D;
   case 'D'
      [C X] = D2D;
   case 'E'
      [C X] = E2D;
   case 'H'
      [C X] = H2D;
   case 'I'
      [C X] = I2D;
   case 'K'
      [C X] = K2D;
   case 'L'
      [C X] = L2D;
   case 'M'
      [C X] = M2D;
   case 'N'
      [C X] = N2D;
   case 'O'
      [C X] = O2D;
   case 'T'
      [C X] = T2D;
   case 'U'
      [C X] = U2D;
   case 'Y'
      [C X] = Y2D;
   otherwise
      error('an unknown letter, the routine is
terminated')
end
```

In this procedure every case is occupied for a letter. For example,

the routine [C X] = B2D describes the letter B in two dimensions.

```
function [C X] = B2D
% Letter 'B' using simplices in 2D space
C = attach(plex, [1 2 5; 2 5 8; 2 8 9; 2 9 10; 2 3 10;
   3 10 11; 3 10 11; 3 4 11; 4 8 11; 4 5 8; 5 6 13; 6 13
14;
   6 7 14; 7 14 15; 1 7 15; 1 12 15; 1 12 13; 1 5 13]');
X = [0 0 1 1 0.15 1 1 0.15 0.15 0.85 0.85 0.15 0.15 0.85
0.85
   -1 1 1 0.3 0 0 -1 0.15 0.85 0.85 0.4 -0.85 -0.15 -0.15
-0.85];
```

Using this set of the letters we may easily build a complex for the word HELLO. For this purpose we have made the function

$$\texttt{[}C\ X\texttt{]} = \texttt{word2DComplex}(W)$$

which constructs a word using a simplex technique (W performs a word — a char array). Here is the structure of the function. We use the PLEX function

$$\texttt{simplices}(C, k)$$

which gives k-cells of the simplicial complex C.

```
dist = 0.5; % a distance between letters
[C X] = choose2DLetter(W(1));
m = max(X(1,:));
m0 = max(simplices(C,0));
m1 = max(max(simplices(C,1)));
m2 = max(max(simplices(C,2)));
for j = 2:length(W)
   % get the current letter of the word
   [Ccurr Xcurr] = choose2DLetter(W(j));
   Y = Xcurr(1,:);
```

```
    % shift all coordinates appropriately
    Y = Y + (m - min(Y)) + dist;
    Xcurr(1,:)  = Y;
    % increase the indices of the vertices
    s0 = simplices(Ccurr,0) + m0;
    s1 = simplices(Ccurr,1) + m1;
    s2 = simplices(Ccurr,2) + m2;
    % add simplices into the entire complex
    C = attach(C, s0);
    C = attach(C, s1);
    C = attach(C, s2);
    % assembling coordinates of the points
    X = [X Xcurr];
    % redefine the parameters
    m = max(Y);
    m0 = max(s0);
    m1 = max(max(s1));
    m2 = max(max(s2));
end
```

Now we invoke this function to get a simplicial complex and coordinates of the word: `[C X] = word2DComplex('HELLO');`. After that we launch the function `plot3complex(`C, X`)` to make the visual presentation (see Figure 17). The functions `betti(`$C, 0$`)` and `betti(`$C, 1$`)` returns the results 5 and 1, respectively (and, of course, the second Betti number occurs to be zero).

If we have a set of points and we do not know how to identify a simplicial complex for it, then there is one way to build the complex using the PLEX routines. First, we calculate the distances between all points of the space X using the function

$$D \texttt{ = px\_euclid}(X).$$

Now we have to choose a threshold value ϵ in order to get the

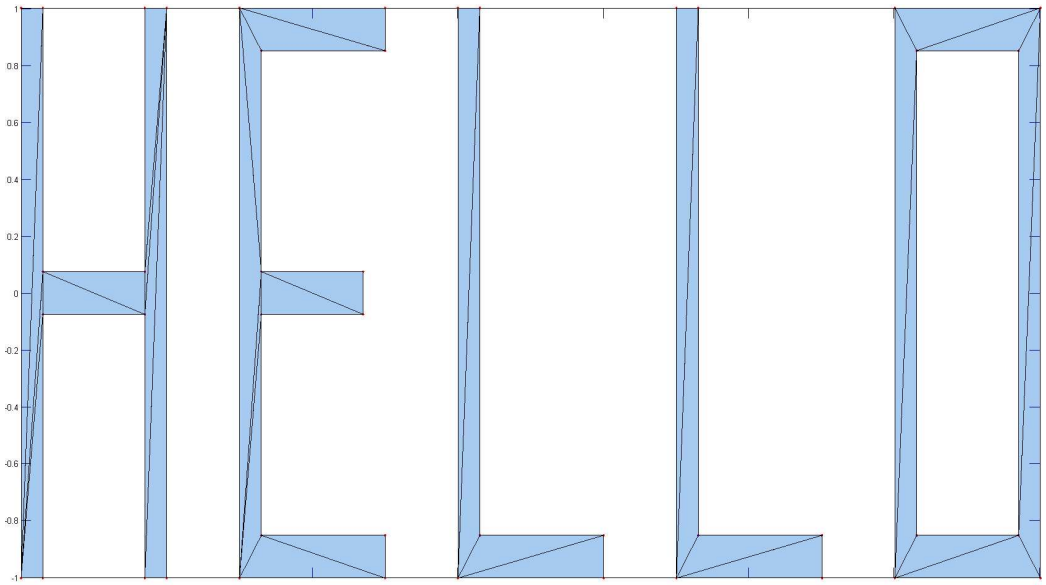

Figure 17: The word HELLO triangulated

proper triangulation. The simplicial structure is being computed by the routine C = `plex`$(D < \epsilon)$. This returns the skeleton of the complex and in order to get the full representation we invoke the function C = `expand`$(C, 1, 2)$. After this we can compute the Betti numbers of our simplicial complex and plot it to take a look.

In order to show this idea practically we need some data. Let us make data points from an object whose view appearance we know. For instance, we consider the letter B. We know that it is one solid object with two holes. Since we have the routine to get a triangulation then we would love to get a lot of points inside of every triangle. Otherwise, there are too little points. To make the points we generate a point cloud in the unit triangle and then map them into every triangle from the original triangulation. Here we come to the point when we need to recall barycentric coordinates of a point. A triangle has three vertices a^0, a^1, a^2 which are affinely independent and every interior point x is represented by its barycentric coordinates: $x = \mu_0 a^0 + \mu_1 a^1 + \mu_2 a^2$. Therefore,

we calculate the barycentric coordinates of points inside the unit triangle and use these coordinates to get the inner points of each triangle from the triangulation.

Let us assume that we have already generated enough points X in the unit triangle and want to get their barycentric values. Let us consider an inner point $p \in X$ of the unit triangle (see Figure 18) to show a way to compute its barycentric coordinates.

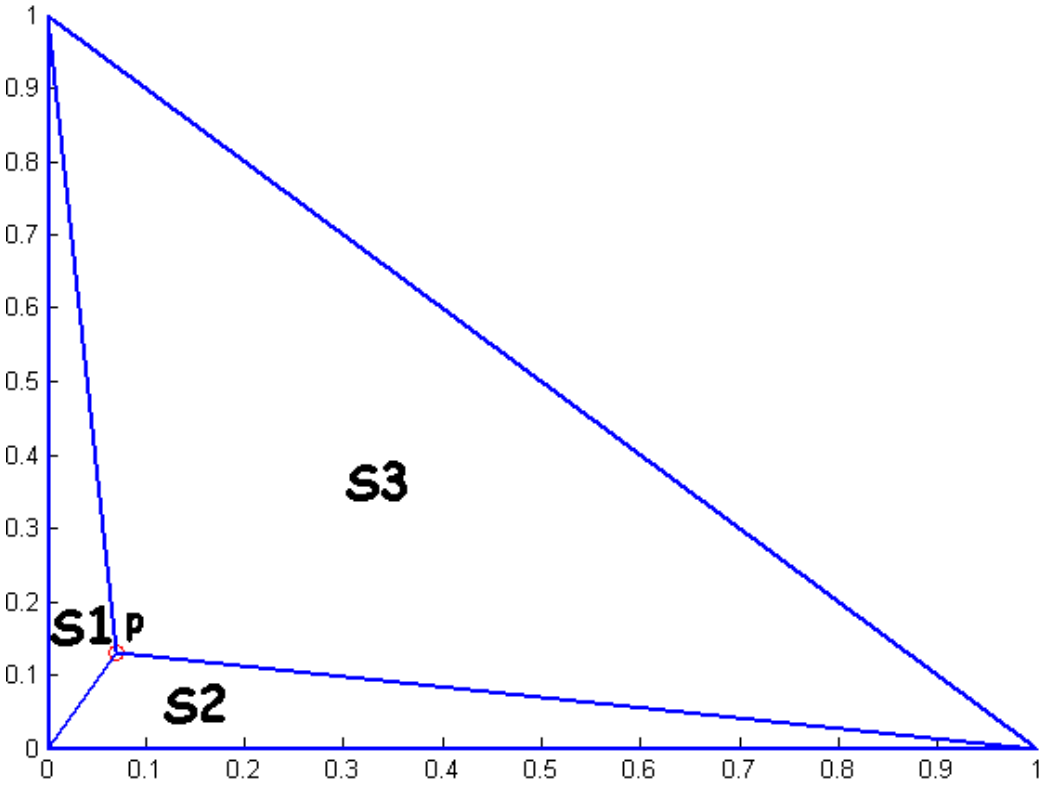

Figure 18: Division of the unit triangle by a point

The point divides the triangle into three parts as shown in the picture. We remind that barycentric coordinates are also understood as "weights" of the point. Here it is illustrated very well because every barycentric number of p is the relation of the corresponding part area to the area of the whole triangle.

That is,

$$\mu_0 = \frac{S1}{S},\ \mu_1 = \frac{S2}{S},\ \mu_2 = \frac{S3}{S}$$

where the symbol S denotes the area of the unit triangle and equals $\frac{1}{2}$ and $S1,\ S2,\ S3$ occupy the denotations of the areas of the separated parts. Hence, for this occasion we may use the formulas $\mu_0 = 2S1,\ \mu_1 = 2S2,\ \mu_2 = 2S3$ to compute the barycentric values of the point p. Once we computed them for p let us see an example how it maps into any another triangle. The triangle which has been chosen has the coordinates of its vertices equal to $\begin{pmatrix} x & 0 & 0 & 0.15 \\ y & -1 & 1 & 0 \end{pmatrix}$. The point p is mapped to the point marked in Figure 19.

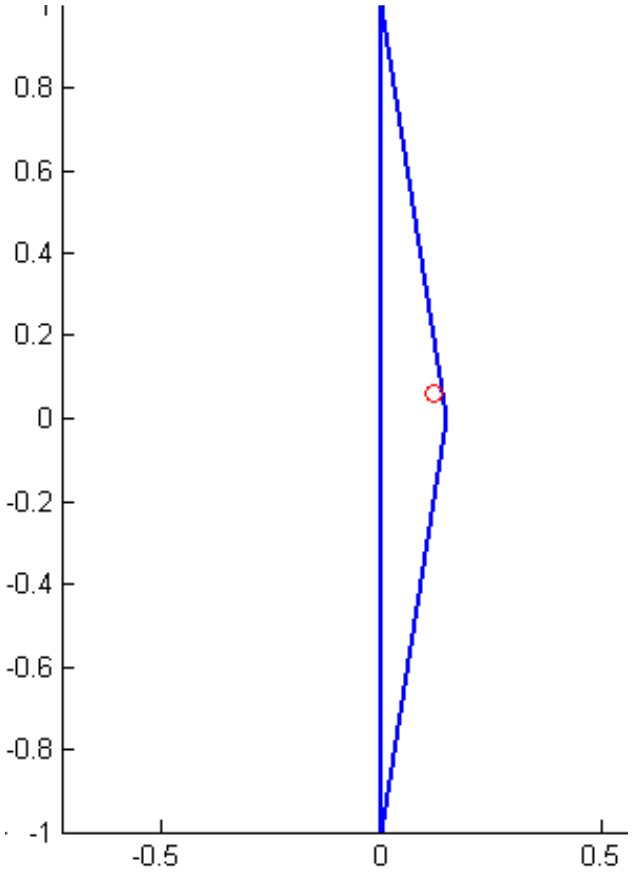

Figure 19: Map of the point p

Using the described technique we may produce data points from a triangulation of any object. Below we present the MATLAB routine [$xunif$ $yunif$] = unifInTriangle(X, Y, n) which implements this algorithm.

```
% define the unit triangle coordinates
Xunit = [0 1 0];
Yunit = [0 0 1];
% generate n random points in a basic triangle
t = 0:1/(n-1):1;
[x y] = meshgrid(t,t);
in = inpolygon(x,y,Xunit,Yunit);
x = x(in);
y = y(in);
S = 2; % 1/polyarea(Xunit,Yunit)
n = length(x);
% initialize vectors for the barycentric values
xunif = []; yunif = [];
% processing every interior point
for j = 1:n
   p = x(j); q = y(j);
   % calculate barycentric coordinates of the point
   u = polyarea([0 p 0],[0 q 1])*S;
   v = polyarea([0 p 1],[0 q 0])*S;
   w = polyarea([0 p 1],[1 q 0])*S;
   if (u ~= 0) && (v ~= 0) && (w ~= 0)
      xunif = [xunif dot([u v w],X)];
      yunif = [yunif dot([u v w],Y)];
   end
end
```

We apply this function to the triangulation of the letter B which we got from the implementation of the routine [C X] = B2D. We store indices of the triangles and then collect

all generated points to yield data for making an experiment with the routine `plex(`$D < \epsilon$`)`. We choose a number of points to be generated in such a triangle accordingly to its area. This is made with the purpose to get data with more or less equal density in the whole area. Thus, the code for assembling the data looks as follows:

```
[C X] = B2D;
% get the indices of the triangles
s = simplices(C,2);
% initialize vectors for data
x = [];
y = [];
% number of points to be used for every triangle
n = 120;
% initializations for every triangle
X1 = zeros(1,3);
X2 = zeros(1,3);
% processing every triangle
for j=1:size(s,2)
   t = s(:,j);
   X1 = X(1,t);
   X2 = X(2,t);
   % generating the points inside of the triangle
   [x1 y1] = unifInTriangle(X1,X2,round(n*polyarea(X1,X2)));
   x = [x x1];
   y = [y y1];
end
```

On the next step we assemble all points in $X = [x;\ y]$ and launch the function D `= px_euclid(`X`)` to determine the distances between the vertices. The parameter α (or ϵ) is set to 0.2. A skeleton of a complex is being constructed by applying the routine C `= plex(`$D < \alpha$`)`. After that we use the PLEX routine

C = `expand`$(C, 2)$ to expand the complex to the full one. The generated vertices and the final triangulation can be seen in Figure 20. The routines `betti`$(C, 0)$ and `betti`$(C, 1)$ give the numbers 1 and 2 which are corresponding to the number of objects we have (one letter) and the number of holes of the target.

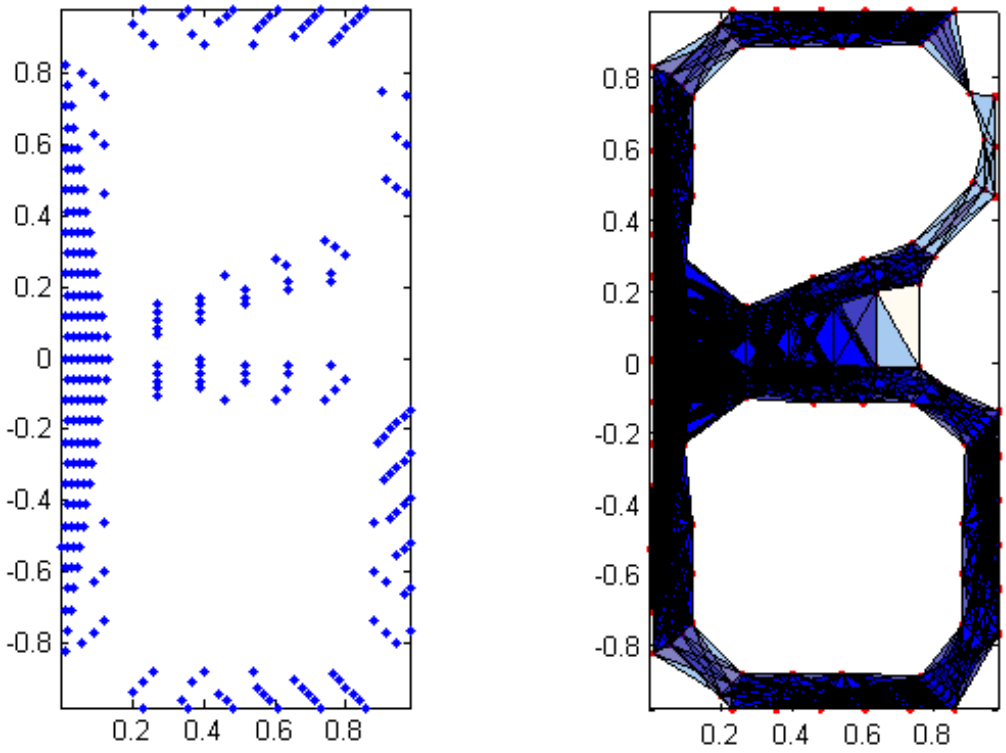

Figure 20: The generated data and the complex for the letter B

However, in this case the computation of Betti numbers took around 150 seconds. It obviously has too many triangles according to the picture. Later we shall describe how to manage this problem (see p. 47).

The toolbox allows us to compute a simplicial complex for three dimensions as well and even for higher dimensions. Nevertheless, we take only the 3D case into account for practical

reasons. And one of the first PLEX authors' tasks for the three-dimensional Euclidean space is to build a simplicial complex to produce a torus. We have chosen to make an object which is homeomorphic to the torus and it is shown in Figure 21. As you can see there are no tetrahedra, hence it is called a triangulation.

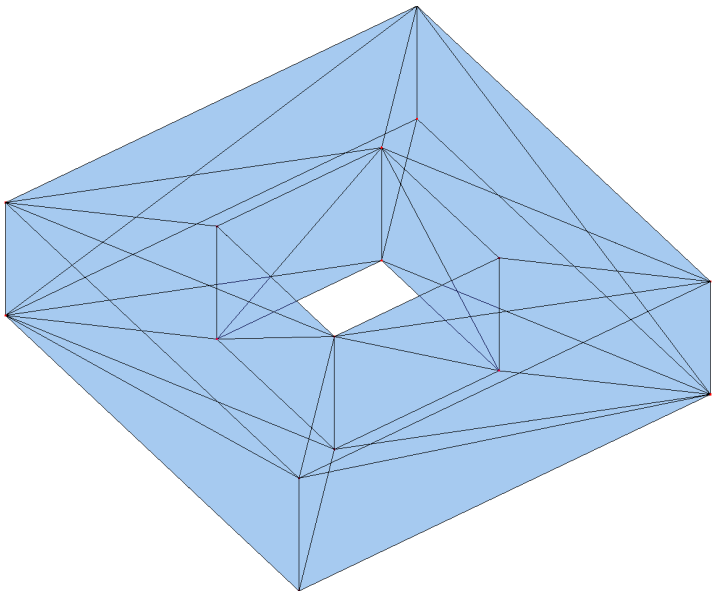

Figure 21: The torus triangulated

The complex was built in MATLAB due to the following code:

```
c = [1 2 3; 1 3 4; 2 3 6; 3 6 7; 5 6 8;
   6 7 8; 1 4 8; 1 5 8]';
c = [c c+8];
c1 = [3 4 11; 3 7 11; 7 11 15; 7 15 16;
   7 8 16; 4 8 16; 4 12 16; 4 11 12]';
c2 = zeros(3,size(c1,2));
c2(mod(c1,2) == 1) = c1(mod(c1,2) == 1)-1;
c2(mod(c1,2) == 0) = c1(mod(c1,2) == 0)-3;
C = attach(plex, [c c1 c2]);
```

We defined the torus coordinates using the next representation:

```
x = [1 0 0 1];
y = [0 0 0 0 1 1 1 1];
z = [0 0 0.3 0.3];
X = [repmat(x,1,2) repmat(abs(x-.3),1,2)
     y abs(y-.3)
     repmat(z,1,4)];
```

Eventually, the picture was plotted, as usual, by the routine `plot3complex`(C, X). The Betti values are correct and equal to 1, 2, 1 for the number of objects, the first Betti number and the number of the voids of the torus, respectively. Computing the persistent homology in this case is very fast, because we have only 16 vertices. To check ability of PLEX to maintain 3-dimensional data we have coded a function which extends a 2-dimensional simplicial complex into a 3D one. The function

$$\texttt{[}C3\ X3\texttt{]} = \texttt{conv3D}(C, X, d)$$

works as explained below:

First, it expands the vertices by adding required indices into the simplicial structure and appends necessary coordinates. The variable d in the function plays a role of the depth, that is how far we go in the axis z direction. Hence, on the first stage the routine also produces lines which connect the vertices with the same 2D coordinates and the different depths.

```
s = simplices(C,0);
if ~isempty(s)
   n = length(s);
   C3 = attach(plex, [s s+max(max(s))]);
   s = simplices(C3,0);
   for k=1:n
```

```
      C3 = attach(C3, [s(k);s(k+n)]);
   end
   if size(X,1) == 1
      X3 = [repmat(X,1,2); repmat(zeros(1,n),1,2)
         zeros(1,n) d*ones(1,n)];
   else
      X3 = [repmat(X(1,:),1,2); repmat(X(2,:),1,2)
         zeros(1,n) d*ones(1,n)];
   end
else
   error('The simplicial complex C is empty')
end
```

On the next step we expand the segments of the complex. It makes a 2D rectangle and provides it with two triangles. Here we start using the PLEX routine C = `union`$(C1, C2)$ which helps us to combine two simplicial complexes into one.

```
s = simplices(C,1);
if ~isempty(s)
   n = size(s,2);
   m = max(max(s));
   C1 = attach(plex, [s s+max(max(s))]);
   s = simplices(C1,1);
   for k=1:n
      for j=1:2
         C1 = attach(C1, [s(j:2,k); s(1:j,k+n)]);
      end
   end
   C3 = union(C3, C1);
end
```

And the final stage expands the triangles to prisms which are represented by three tetrahedra.

```
s = simplices(C,2);
if ~isempty(s)
   n = size(s,2);
   C2 = attach(plex, [s s+m]);
   s = simplices(C2,2);
   for k = 1:n
      for j=1:3
         C2 = attach(C2, [s(j:3,k); s(1:j,k+n)]);
      end
   end
   C3 = union(C3, C2);
end
```

To illustrate this algorithm we made the function

$$\texttt{[}C\ X\texttt{] = word3DComplex}(W, d)$$

which works very similarly to the function

$$\texttt{[}C\ X\texttt{] = word2DComplex}(W).$$

One of the differences is the parameter d to specify a depth of the word W. For visual representation we have chosen to launch this routine with randomly chosen word and depth `[C X] = word3DComplex('MATYLDA', 0.37)` and apply the PLEX function `plot3omplex(C, X)` (see Figure 22).

We would like to know how many letters and holes the word has. To display them we again invoke the function `betti(C, k)`. However, this time we use three k's: $0 \leqslant k \leqslant 2$. The results are: $\beta_0 = 7$, $\beta_1 = 3$, $\beta_2 = 0$, that is the word MATYLDA has 7 letters, 3 holes and no cavities. We are glad that we got the correct result in around two seconds and now we shall test the toolbox PLEX for 3-dimensional not handmade data but generated from one of the letters. For this purpose we take the letter O in 2 dimensions first and generate data in the same way as we did for the letter

B. Then we expand the data into the 3D space, take the vertices only and construct a simplicial complex using a bunch of Plex's routines.

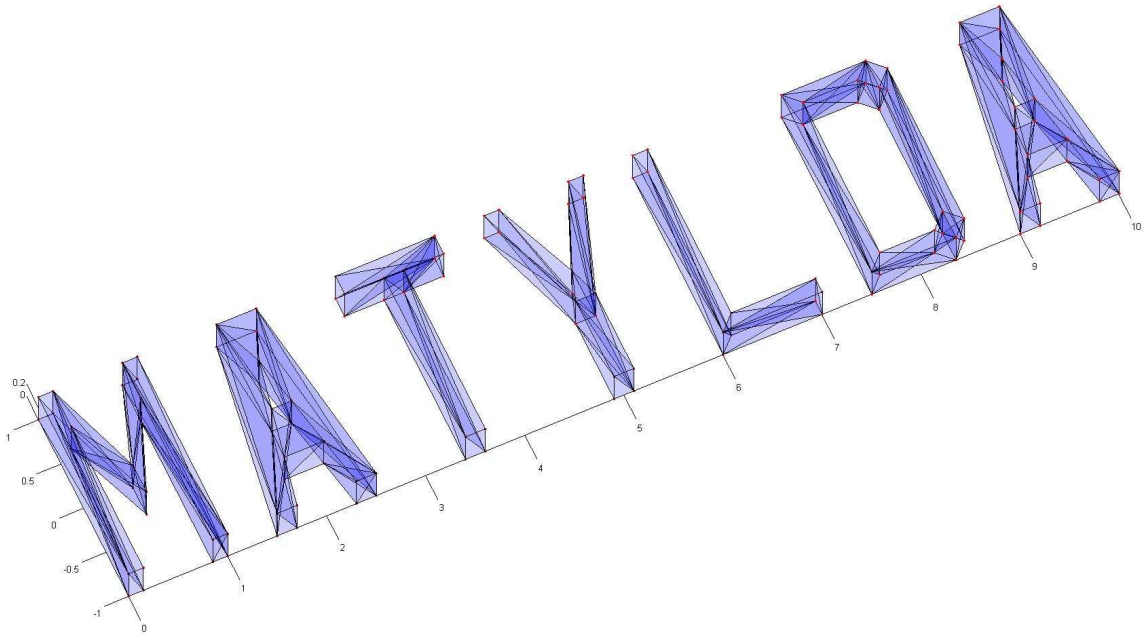

Figure 22: The word MATYLDA tetrahedralized

Thus, we use 35 points to generate vertices from the triangulation of the letter O and increase this amount twice taking the depth parameter to equal 0.2. After that we construct a simplicial complex with an alpha value 0.75 and compute its Betti numbers. The graphical representation is demonstrated in Figure 23 and the zeroth Betti value is 1, the second one equals 1 and there are no cavities. The procedure
C `= plex(px_euclid(`X`) < 0.75)` returns the complex C with 40 vertices, 280 segments, 968 triangles and 2148 tetrahedra. The Betti numbers were computed within around 2.34 seconds. However, one problem occurred when we repeated the same algorithm using 40 points to generate data. We have built a tetrahedralization and computed the zeroth and the first Betti numbers but the routine was not able to compute the second Betti

number, because MATLAB returned the error “Out of memory”. It happened because the procedure
C `= plex(px_euclid(`X`) < 0.75)` returned the complex C consisting of 68 vertices, 790 segments, 4868 triangles and 20118 tetrahedra. Too many elements to maintain the computation.

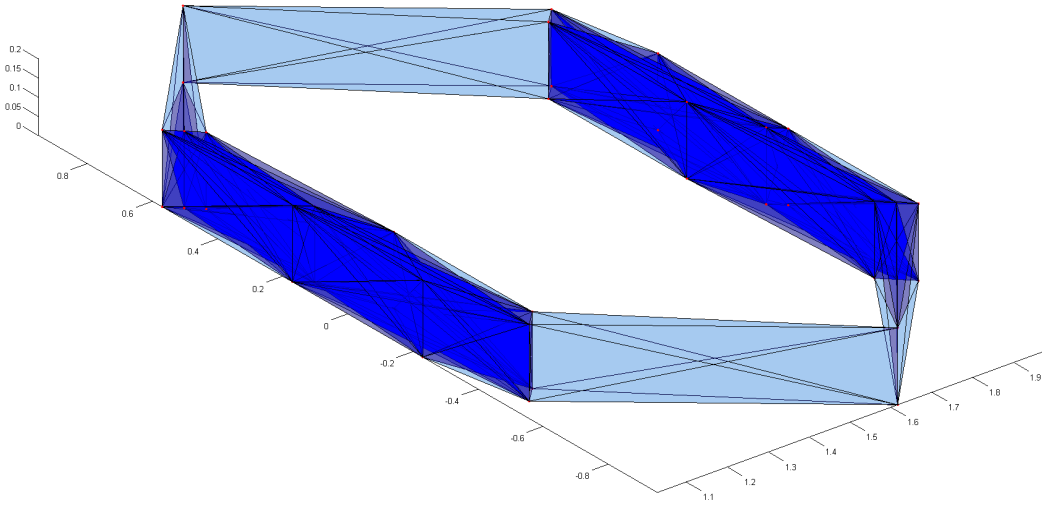

Figure 23: The letter O tetrahedralized

If you want to get more information about PLEX you are welcome to take a look at its web page [9].

7 Results

In this section we present our approaches to solve a problem of the computation of the persistent homology of an object. We show how our routines work with artificial and real data.

7.1 The persistent homology of artificial data

After testing the PLEX routines we started to think how to get rid of unnecessary simplices and the first idea was to compute a Delaunay triangulation. Implementing the MATLAB built-in algorithm `delaunay(`x, y`)` we got the indices of the triangles of the simplicial structure in two dimensions. For the 3D Euclidean space there is a function `delaunay3(`x, y, z`)`. However, we had here another problem — the Delaunay triangulation does not consider an alpha parameter at all, therefore we should have coded an algorithm to build a triangulation with taking the alpha value into account. The concept is to erase those segments of the simplicial complex whose vertices have a distance between them no less than alpha.

Let us first maintain this case in two dimensions. The MATLAB Delaunay function returns an m-by-3 matrix where m is the number of triangles. If we find a line whose vertices are not sufficiently close to each other, then we can not erase the whole row because the triangle may have an acceptable segment. Therefore, we store all segments in the certain variable that we denote by $s1$ to specify that it contains 1-simplices. Eventually, besides $s1$ we have two more "storages" of such type: $s0$ for vertices and $s2$ for triangles.

Since our task is to compute homology and the persistent homology then it would be nice to take care of betti values in advance. And that is what we do, and we construct the standard matrix representation for the boundary operator and count how many vertices, lines and triangles the complex has. Remind that in the 2D case we need only the matrix that connects vertices

and segments. Below we demonstrate parts of a routine we have coded in MATLAB. We called this routine

$$\texttt{function [}C, b\texttt{] = alphaBetti}(x, y, Alpha)$$

and it returns the simplicial complex C and the Betti number in the vector b.

First, we make some simple preparation: compute a Delaunay triangulation, define vertices of a future simplicial complex, add an additional parameter to make the computation faster and put null values to necessary elements. In order to define 0-simplices correctly we should go through the Delaunay indices and memorize them. We need this procedure because there might be some points which are not in the general position in the point cloud. For instance, we may find three or more points on the same line or no less than four points on the same circumcircle. In the end we shall have a set of vertices whose size is not necessarily the same as the size of the data.

```
% make a general triangulation
d = delaunay(x,y);
% initiliaze parameters
% define a square radius of an alpha circle
a = Alpha^2;
s0 = []; % 0-simplices (vertices)
s1 = []; % 1-simplices (lines)
s2 = []; % 2-simplices (triangles)
D1 = []; % matrix connecting vertices with lines
i0 = 0; % number of vertices
i1 = 0; % number of lines
i2 = 0; % number of triangles
% assemble all vertices of the delaunay triangulation
for j = 1:length(x)
   if ismember(j,d)
      s0 = [s0; j];
```

```
      i0 = i0 + 1;
   end
end
```

Now we go through all triangles made after the Delaunay procedure. We add some accessorial parameters for our usual purpose — to speed up the computation. We take every separate row of d, which represents the Delaunay structure, and process each segment of it. After checking the distance between vertices of a segment we decide whether to add this line to $s1$ or not. If we add then we increment the number of lines and add two entries into the matrix $D1$. If all segments were accepted to be put into $s1$ then we put the current triangle into $s2$ as well and increment the number of triangles.

```
for i = 1:size(d,1)
   % some additional parameters
   d1 = d(i,:);
   xx = x(d1); yy = y(d1);
   b = true; % the points make a triangle
   % processing every triangle
   for j = 1:2
      % auxiliary elements
      xcurr = xx(j); ycurr = yy(j);
      d1j = d1(j);
      for k = j+1:3
         d1k = d1(k);
         d2 = [d1j d1k];
         % points should be close enough to each other
         if(xcurr-xx(k))^2 + (ycurr-yy(k))^2 < a
            % do we have this segment?
            if ~ismember(d2,s1,'rows') &&...
               ~ismember([d1k d1j],s1,'rows')
            s1 = [s1; sort(d2)];
```

```
            i1 = i1 + 1;
            % store proper values in the matrix D1
            D1(d1j,i1) = -1;
            D1(d1k,i1) = 1;
         end
      else
         b = false;
      end
   end
   if b
      s2 = [s2; d1];
      i2 = i2 + 1;
   end
end
```

Following this technique for every triangle we build the standard matrix representation and, eventually, compute its rank. Finally, we assemble all simplices in one structure and compute Betti numbers using the formulas we have derived earlier (see p. 22).

```
r1 = rank(D1);
C.s2 = s2;
C.s1 = s1;
C.s0 = s0;
b = [i0-r1, i1-r1-i2];
```

Let us demonstrate the algorithm on the example with the teapot points. In distinction from the Ashape case we use the original points only, which are presented in Figure 24.

After several attempts we found out that the alpha parameter 0.4 is a very good value to make a proper triangulation C. The graphical result of applying the described routine is shown in Figure 25.

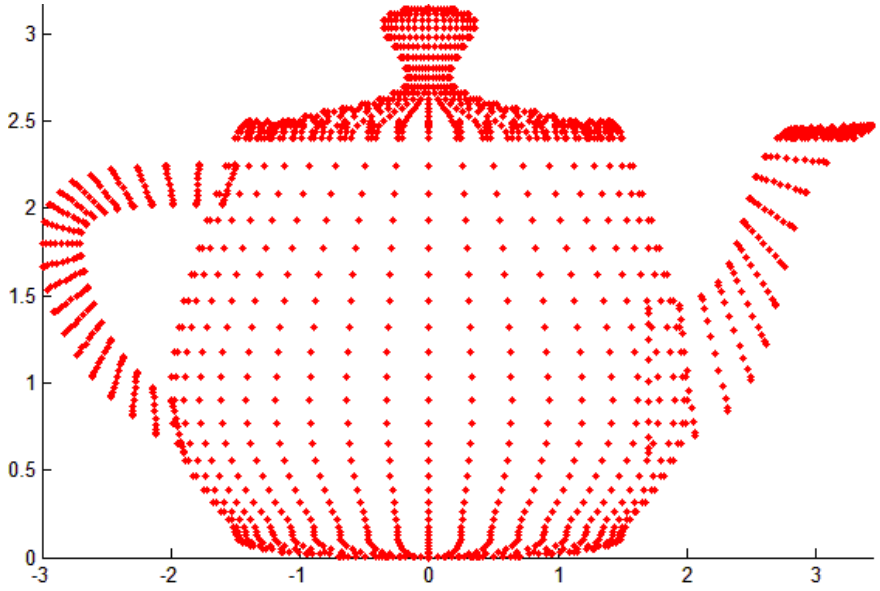

Figure 24: The kettle points

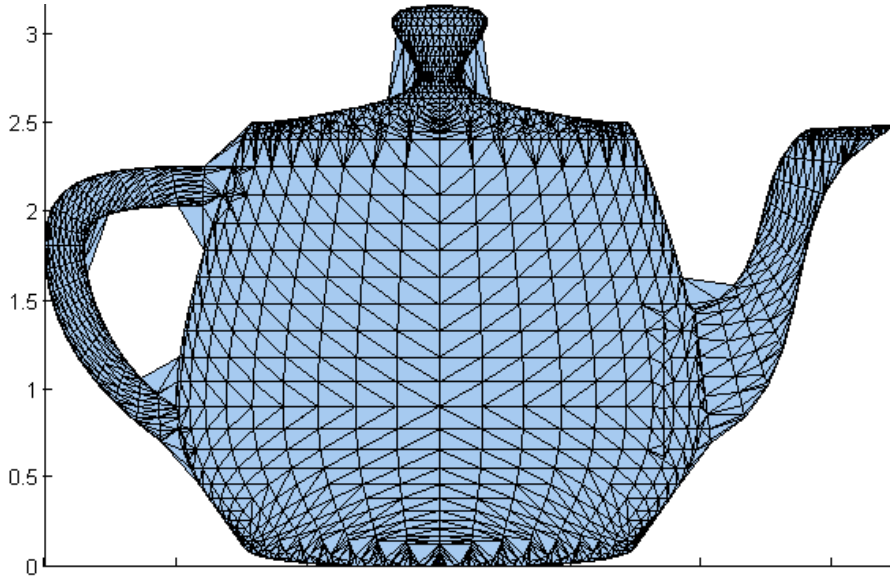

Figure 25: The triangulation of the teapot vertices

The Betti numbers we get, are equal to 1 and 1 for the number of elements and the number of holes, respectively. The coded algorithm has spent around 240 seconds (in other words,

four minutes) and produced the simplicial complex C consisting of 1724 vertices, 3302 segments and 5026 triangles. To compare it with the PLEX routines we convert every k-chain (that is, all k-simplices) of the complex C to a PLEX simplicial structure denoted by PC. We applied the PLEX routine `betti` to compute the values, however, MATLAB returned the message that it was out of memory:

```
>> tic; b = [betti(PC,0) betti(PC,1) betti(PC,2)];
t =toc;
???  Error using ==> svd
Out of memory
```

7.2 The persistent homology of real data

Now we consider a real problem. Let us say we have a 3D point cloud which represents a tower of power line. And the structure of the target is interesting for us and, therefore, we are to compute the persistent homology of the object. First we would like to test the data in two dimensions. Thus, we erase the superfluous coordinate and apply our procedure for the 2-dimensional points (see Figure 26).

On the next step we utilize our technique with an alpha parameter equals 3.5. We choose this value from the picture of the vertices to avoid searching an alpha parameter since we want to present one of the results and finding the correct alpha value using some algorithm would take quite much time. It might also return a wrong value. As a result we get the Betti numbers 1 and 2, the number of connected elements and the number of holes, respectively. The triangulation can be seen in Figure 27.

From these two plots we can see that for the case of the power line tower this is clearly an underestimate, but the correct one for alpha equals 3.5. The result shows that we have only one object which is correct because the power line tower does not consist of

disconnected components.

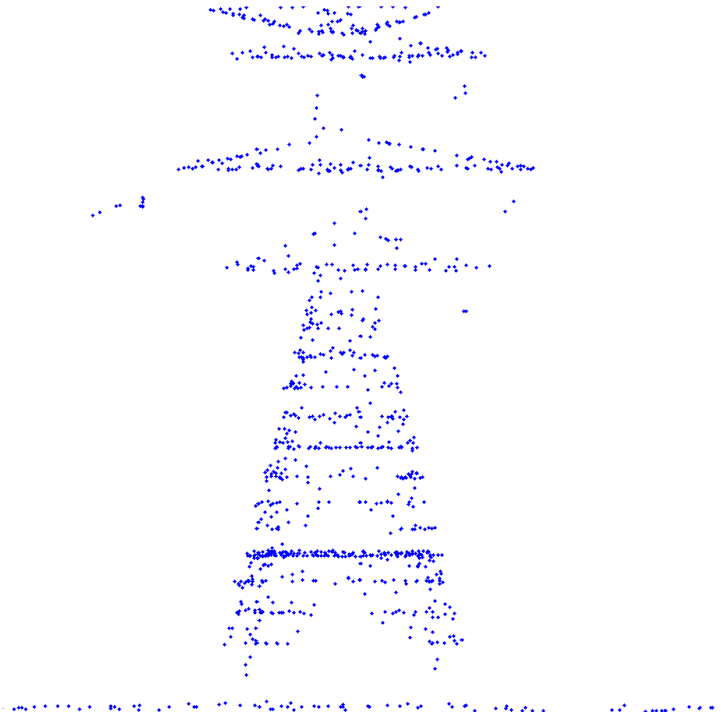

Figure 26: The tower's vertices

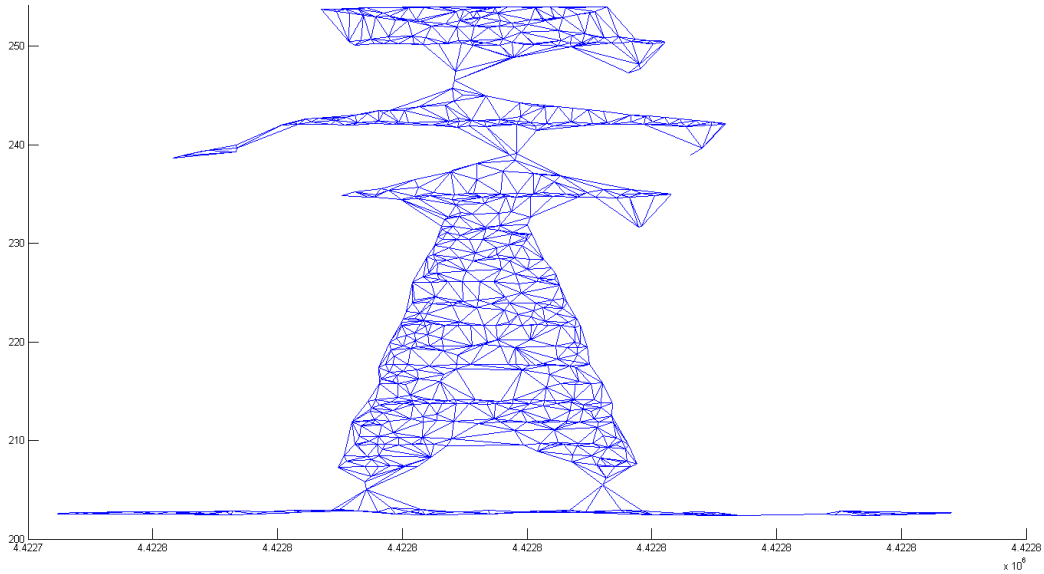

Figure 27: The tower's triangulation

In order to deal with 3D points we have coded the routine `function [`C, b`] = alphaBetti3(`$x, y, z, Alpha$`)` whose structure is similar to the 2D case function, however we have to construct more matrices. Here we need to add an additional option in the `delaunay3` function. The options `Qt, Qbb, Qc` are the default ones and the option `Qz` is to allow the Delaunay triangulation of cospherical sites. For example, it helps when we are computing the persistent homology of a sphere. It also reduces precision errors for nearly cospherical sites.

```
% make a general triangulation
d = delaunay3(x,y,z,{'Qt', 'Qbb', 'Qc', 'Qz'});
% initiliaze parameters
% define a square radius of an alpha circle
a = Alpha^2;
s0 = []; % 0-simplices (vertices)
s1 = []; % 1-simplices (lines)
s2 = []; % 2-simplices (triangles)
s3 = []; % 3-simplices (tetrahedra)
D1 = []; % matrix connecting vertices with lines
D2 = []; % matrix connecting lines with triangles
M1 = []; % an auxiliary matrix
i0 = 0; % number of vertices
i1 = 0; % number of lines
i2 = 0; % number of triangles
i3 = 0; % number of tetrahedra
% assemble all vertices of the delaunay triangulation
for j = 1:length(x)
   if ismember(j,d)
      s0 = [s0; j];
      i0 = i0 + 1;
   end
end
```

Here we introduce the matrix $M1$ which will store indices of the segments using the indices of the vertices. Then we go through all tetrahedra and consider every triangle of the 3-simplex. For each 1-simplex we repeat the same procedure as in two dimensions, however there is a difference that for the triangles the matrix $D2$ is being built. And in the end of the exterior loop we add the tetrahedron in our variable $s3$ that contains all segments that have been accepted.

```
for i = 1:size(d,1)
   d1 = d(i,:); % some additional parameters
   xx = x(d1); yy = y(d1); zz = z(d1);
   t = true; % the points make a tetrahedron
   % processing every tetrahedron
   ind = [1 2 3; 1 2 4; 1 3 4; 2 3 4];
   for w = 1:4
      b = true; % the points make a triangle
      for j = 1:2
         % auxiliary elements
         p = ind(w,j);
         xcurr = xx(p); ycurr = yy(p); zcurr = zz(p);
         d1p = d1(p);
         for k = j+1:3
            q = ind(w,k);
            d1q = d1(q);
            d2 = [d1p d1q];
            % points should be close enough
            % to each other
            if (xcurr-xx(q))^2 + (ycurr-yy(q))^2 +...
               (zcurr-zz(q))^2 < a
               % do we have this line?
               if ~ismember(d2,s1,'rows') &&...
                     ~ismember([d1q d1p],s1,'rows')
                  s1 = [s1; sort(d2)];
```

```
            i1 = i1 + 1;
            % the auxiliary matrix connects the
            % lines and the triangles
            M1(d1p,d1q) = i1;
            M1(d1q,d1p) = i1;
            % store proper values in the matrix
            D1(d1p,i1) = -1;
            D1(d1q,i1) = 1;
         end
      else % the distance between the points
           % is too big
         b = false;
         t = false;
      end
   end
end % the triangle's lines have been processed
if b
   d3 = sort(d1(ind(w,:)));
   dperm = perms(d3);
   bool = true;
   % do we have this triangle?
   for j = 1:6
      if ismember(dperm(j,:),s2,'rows')
         bool = false;
         break
      end
   end
   if bool
      s2 = [s2; d3];
      i2 = i2 + 1;
      D2(M1(d3(1),d3(2)),i2) = 1;
      D2(M1(d3(1),d3(3)),i2) = -1;
      D2(M1(d3(2),d3(3)),i2) = 1;
   end
```

```
      end % the triangle has been processed
   end % the tetrahedron's triangles
        % have been processed
   if t
      s3 = [s3; d1];
      i3 = i3 + 1;
   end
end % all tetrahedra have been processed
```

The same last stage: computing the rank of the built standard matrix representation, assembling all simplices in one structure and computing Betti numbers using the derived formulas (see p. 22).

```
r1 = rank(D1);
r2 = rank(D2);
C.s3 = s3;
C.s2 = s2;
C.s1 = s1;
C.s0 = s0;
b = [i0-r1, i1-r1-r2, i2-r2-i3];
```

To demonstrate this algorithm we have applied it to the real 3D points of the scanned power tower (see Figure 28). We have empirically chosen an alpha parameter equal to 3.5, the same as for the two-dimensional case. The routine has spent around 1400 seconds (23 minutes, 20 seconds) and produced 868 vertices, 4506 segments, 6566 triangles and 2912 tetrahedra. The Betti values, which have been computed, denote that we have got 18 connected elements, six holes and four cavities in the power tower. The simplicial complex, which geometrically appoximates our target, is shown in Figure 29 and it is visible that the tower has been presented as one object.

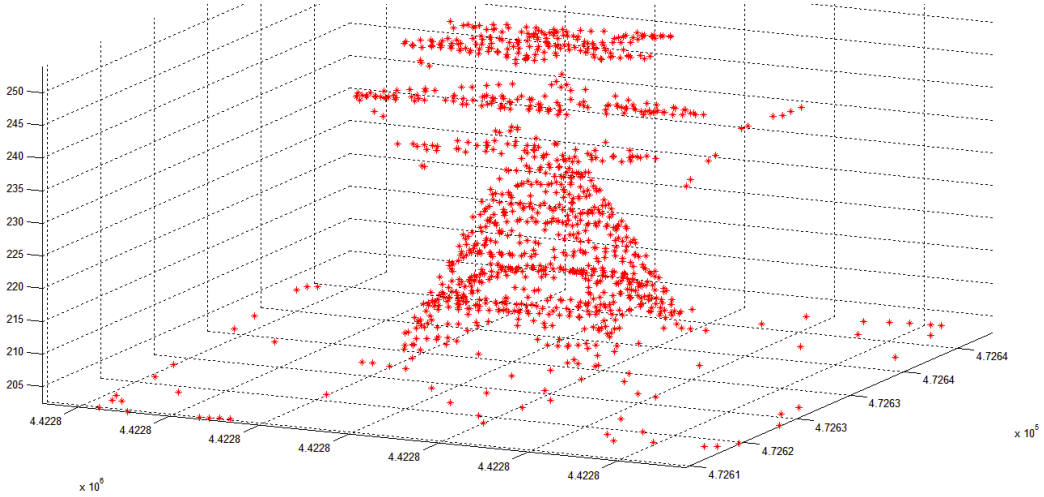

Figure 28: 3D points

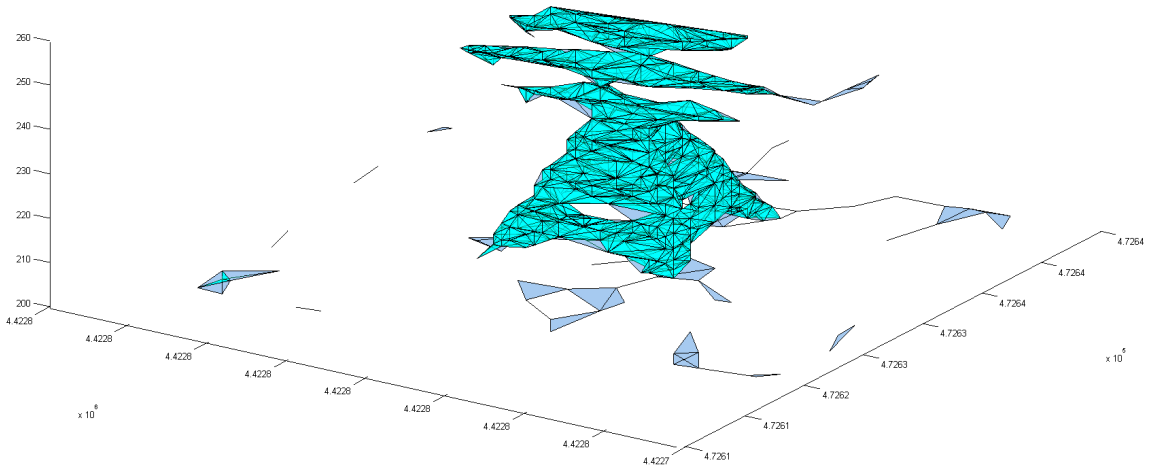

Figure 29: Tetrahedralization

Let us assume that we have scanned points of wires which connect power line towers. Then we would take considerably large alpha to handle these vertices, since every wire is very long. This idea may help us to distinguish one cable from another and

count their number. However, here we may get a problem with the tower itself. With a very big alpha number we get one solid object and we are not able to identify bars and beams of the tower. When we try to find an appropriate value of alpha to label the bars, then we can easily lose our simplicial structure and come to a discrete point cloud.

In order to solve such a task one may try to apply a concept of using different alpha values for different regions of the vertices. For example, use smaller alpha for a more dense area and vice versa.

Another solution is to use the same alpha parameter, but a more dense set of points. To get this real data we used for the illustration, the method of Airborne Laser Scanning (ALS) was applied. However, it does not produce a very dense point cloud. In order to make plenty of vertices it is better to use Terrestrial Laser Scanning technology (TLS).

In our algorithm we may get one more problem when the matrices are too big and we are not able to compute the persistent homology because MATLAB runs out of memory then. However, we have made the routines C = `alphaDelaunay`$(x, y, Alpha)$ and C = `alphaDelaunay3`$(x, y, z, Alpha)$ which allow us to construct simplicial complexes of a point cloud in two and three dimensions, respectively. For example, with the original 3D points of the kettle (see Figure 30) we are able to construct a tetrahedralization only and in order to implement this the routine C = `alphaDelaunay3`$(x, y, z, Alpha)$ is applied. The code of the function is very similar to the structure of `alphaBetti3` function except that here we do not build the standard matrix representations.

The procedure has built a simplicial complex for this 3D point cloud within around 790 seconds (a bit more than 13 minutes). The complex contains 3241 vertices, 10273 lines, 24512 triangles and 171300 tetrahedra. The tetrahedralized teapot presentation is shown in Figure 31. One can easily recognize the handle, the

beak and the lid.

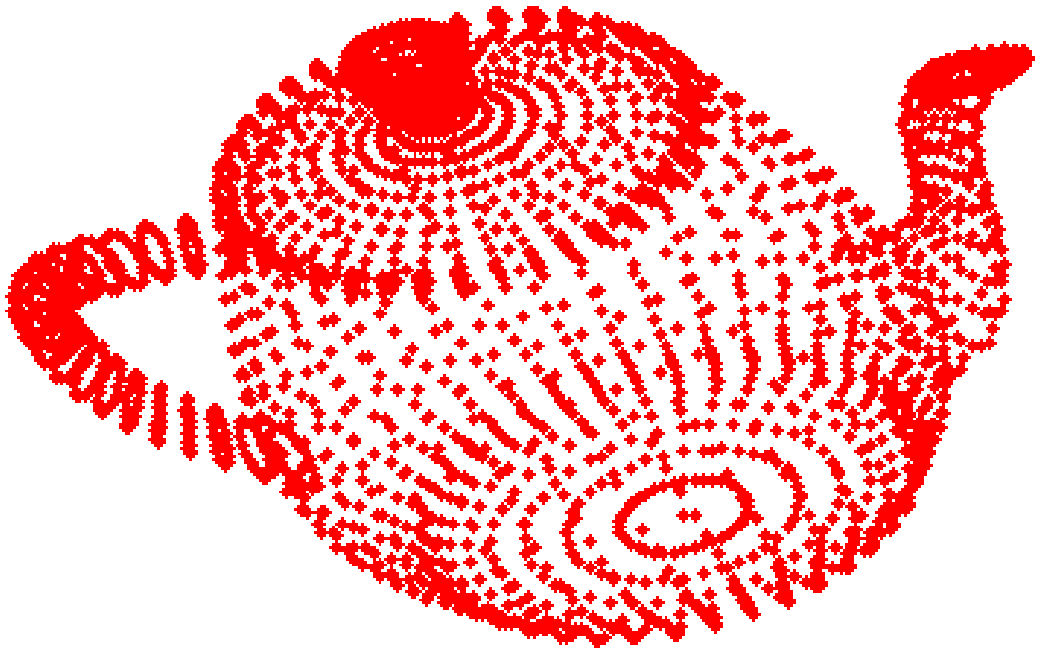

Figure 30: The vertices of the kettle

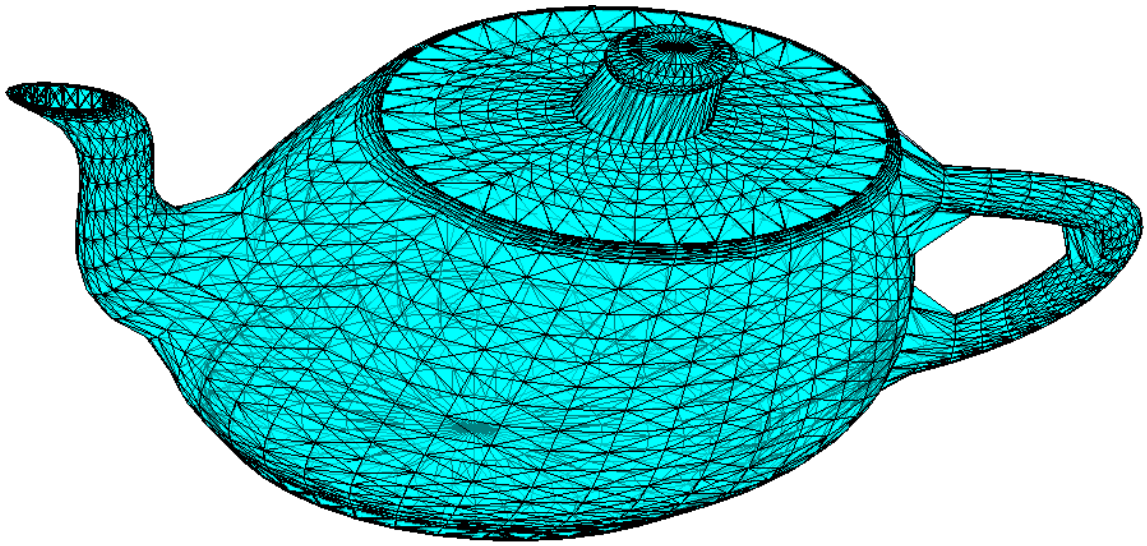

Figure 31: The tetrahedralization of the teapot

Now let us speak about the problem of choosing alpha. The most reliable way to choose it is to look at the vertices and roughly define a starting value. Then we need a step of changing this number. One solution is to decrease the alpha parameter twice on each step. We stop the computation when either the Betti numbers do not change anymore or the alpha value is less than the minimal distance between the vertices. The code representing this idea in two dimensions is shown below. At every step we plot triangulations of the object, thus you can see at which step the value satisfies you better.

```
distData = dist([x y]');
minDist = min(min(distData(distData > 0)));
[C, b] = alphaBetti(x,y,Alpha);
bnew = inf;
while (~isequal(b,bnew)) && (Alpha > minDist)
   Alpha = Alpha*0.5;
   bnew = b;
   [C, b] = alphaBetti(x,y,Alpha);
   figure
   hold on
   if ~isempty(C.s1)
      if ~isempty(C.s2)
         fill(x(C.s2'), y(C.s2'), 'b'); alpha(0.5)
      end
      line(x(C.s1'), y(C.s1'), 'color','k');
   end
   plot(x(C.s0), y(C.s0), 'r.');
end
```

Another solution is leaving the alpha value as it is and accept the result it produces. If it does not satisfy us, then choose another alpha parameter empirically. For honesty, we confess that during all experiments, we have conducted, we were almost

always choosing parameters empirically. Except the last one we present here. We take the 2-dimensional points of the teapot and initialize the first alpha value as the following
`Alpha = max(max(distData(distData > 0)))/6`.

After several steps we have got the result which is illustrated in Figure 32. As you can see the described algorithm brought us come back to the vertices of the kettle. Most likely, in order to use this technique efficiently we shall need a plenty of points like in the Ashape's case.

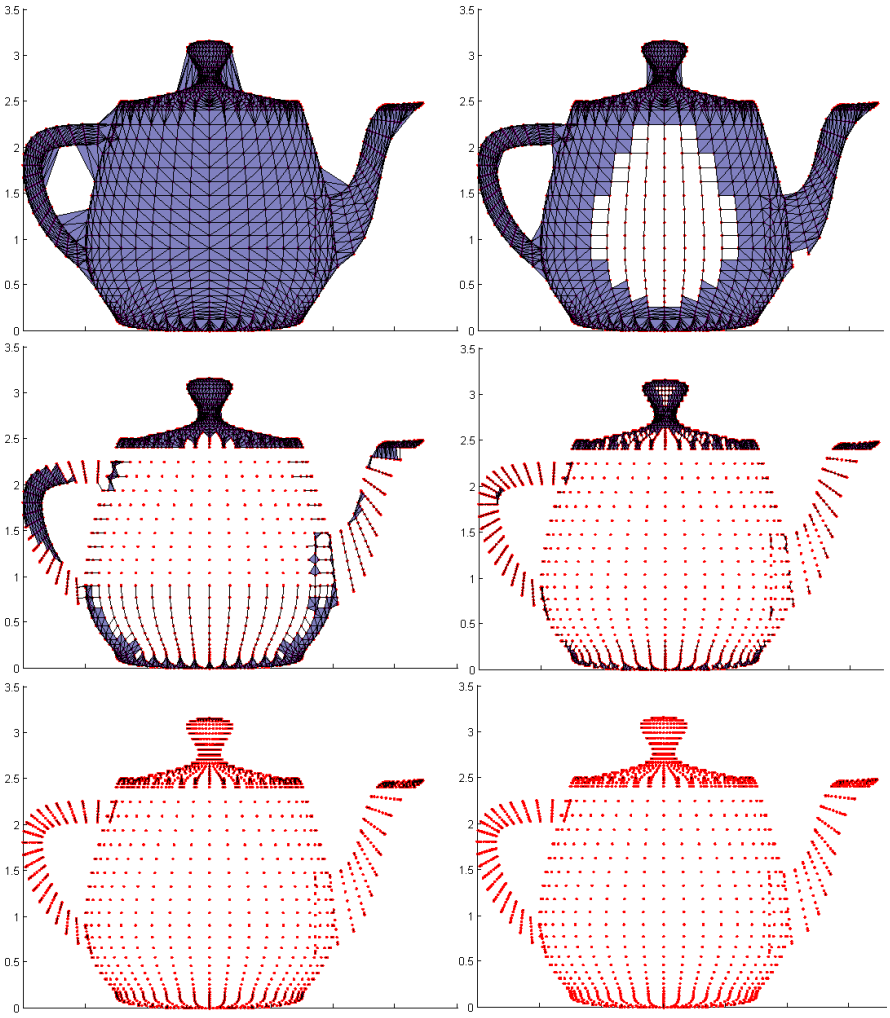

Figure 32: The stages of the triangulation of the teapot

8 Conclusions and future perspectives

In the paper we have presented ideas and implemented methods to compute the persistent homology of a target. In most cases, the persistent homology is represented by Betti numbers which are topological invariants and, basically, are unique for homeomorphic objects. The tools such as Ashape and PLEX have made first steps in the direction of development and implementation of the ideas using programming languages.

Most likely, the Ashape MATLAB routines can be expanded into three dimensions. They may also be improved in the sense of using a prefiltered set of points: probably, we do not need a very dense point cloud to demonstrate the target sufficiently well. Thus, it is one direction where we can look to find refinement.

Another point is to develop the PLEX C++ and MATLAB routines in such a way that they do not need to make so many simplices. One can assume that our routines might be useful for this purpose. The Delaunay triangulation technique is a very helpful one to be used in the tools. The routines may also provide an option for building a simplicial complex: to build the complex only or to calculate the Betti numbers during the process. Sometimes we need the complex only and it is certainly faster to make it than add the computation of Betti numbers in the algorithm. However, we are to be sure that we shall not need them in future because calculating the Betti values within building the simplicial structure takes much less time than constructing the complex and computing them only afterwards.

Our results are not perfect and the biggest problems are nowadays the fact that we use matrices which might be too big, and the choice of the alpha parameter. One can think about using sparse matrices, however we shall have to connect the ranks of the sparse matrices (which are not the same as ones for the original matrices) and the formulas for the Betti numbers. One of ideas of choosing the alpha value is changing it according to the local

density of the point cloud. That is, take small regions of the points and use different alpha numbers for each region. For more details we refer the reader to [13].

The area of studying the problem of computing the persistent homology is not old and is nowadays under rapid development. There is a nice idea of using barcodes with a combination of filtered simplicial complexes and this concept helps us to identify Betti numbers of the simplicial structure with regard to the alpha value. You can read more about this approach in [5] and [7]. As you can see there is a lot of things to do and, likely, some techniques might be assembled and refined in order to get better results.

Index

References

[1] Alexandrov, P. S., *"Introduction to dimension theory"*, Moscow: Nauka, 1973, 191–211 (in russian)

[2] Barber, C. B., Dobkin, D. P., and Huhdanpaa, H. T., "The Quickhull algorithm for convex hulls", *ACM Transactions on Mathematical Software*, 22(4): 469–483, Dec 1996, http://www.qhull.org/

[3] Edelsbrunner, H., and J. Harer, "Persistent homology — a survey", *Contemporary Mathematics* 453 (2008): 257-282.

[4] Edelsbrunner, H., and E. P. Mücke, "Three-dimensional alpha shapes", *ACM Transactions on Graphics* 13 (1994): 43–72.

[5] Carlsson G. and T. Ishkhanov, *"A topological analysis of the space of natural images"*, preprint (2008), http://comptop.stanford.edu/preprints/

[6] Carlsson, G., and V. de Silva, "Computational topology using Plex", *TMSCSCS: Plex*, 10 Jul 2006, <http://comptop.stanford.edu/programs/Plexercises2.pdf>

[7] Chris R., "Barcodes: The persistent topology of data", *Bulletin (New Series) Of The American Mathematical Society* Volume 45, Number 1, Jan 2008: 61–75.

[8] De Silva, V., "Plex – a MATLAB library for studying simplicial homology", *TMSCSCS: Plex*, 24 Oct 2003, <http://comptop.stanford.edu/programs/plex/plexintro.pdf>

[9] De Silva, V., *TMSCSCS: Plex*, 7 Sep 2006, <http://comptop.stanford.edu/programs/plex.html>

[10] Ishkhanov T., *"Topological method for shape comparison"*, preprint (2008), http://comptop.stanford.edu/preprints/

[11] *MATLAB Central File Exchange — ashape: a pedestrian alpha shape extractor*, 24 Apr 2007, <http://www.mathworks.com/.../loadFile.do?objectId=6760>

[12] Tamal K. Dey, and Guha S., "Computing homology groups of simplicial complexes in $\mathbb{R}^3$". *Journal of the ACM* 45 (1998): 266–287.

[13] Tenenbaum, J. B., V. de Silva, and J. C. Langford, *Isomap Homepage*, 22 Dec 2000, <http://isomap.stanford.edu/>

[14] Zomorodian, A., and G. Carlsson, *"Computing persistent homology"*, Proceedings of the twentieth annual symposium on Computational geometry, New York, NY, USA: ACM, 2004, 347–356.

[15] Zomorodian, A., and G. Carlsson, "Localized Homology", *Computational Geometry* (2008), doi:10.1016/j.comgeo.2008.02.003