DHAWAN, SAHIL, M.A. Analyzing the Topological Properties of 3D STL Files. (2024)
Directed by Dr. Thomas Weighill. 35 pp.

Part features in 3D objects that are created for the purpose of rapid prototyping and fabrication using methods such as 3D printing or CNC milling often do not match the measurements of the finished product's features due to tolerancing and clearance design errors. A solution to measuring the range of potential error that the features of a part can have can be found through the topology and persistent homology of the chosen truth mesh of that part. Variations of the part feature(s) to be measured, such as different diameters of a hole, can be created with a sequence of 3D object files that change the measurements of the selected feature(s) while keeping other features the same. We use FreeCAD to generate these part variations and export them as ASCII STL files. A new mesh is created for each of these files to convert them to Delaunay triangulations. We use the Delaunay triangulation mesh in tandem with the filtered simpliicial complex generated by the Alpha complex. The Alpha complex is modified so all its matching simplicies that were present in the Delaunay triangulation mesh have a birth time of 0 for accurate analysis using persistent homology. We then generate a persistence diagram for each variation of the 3D part feature to see the progression of change in topology for the chosen feature with respect to the chosen truth 3D STL file. Our results show the potential errors of features in a 3D STL file can be calculated through the topology of part variations which represent different measurements in tolerance and clearance.

ANALYZING THE TOPOLOGICAL PROPERTIES OF 3D STL FILES

by

Sahil Dhawan

A Thesis Submitted to
the Faculty of The Graduate School at
The University of North Carolina at Greensboro
in Partial Fulfillment
of the Requirements for the Degree
Master of Arts

Greensboro
2024

Approved by

____
Committee Chair

Dedicated to my cats, Piccolo and Gohan, and my dogs, Bebe and Tilly:
the solution to, and cause of, all my problems.

APPROVAL PAGE

This thesis written by Sahil Dhawan has been approved by the following committee of the Faculty of The Graduate School at The University of North Carolina at Greensboro.

Committee Chair	______________________ Thomas Weighill
Committee Members	______________________ Michael Hull
	______________________ Thomas Lewis

____
Date of Acceptance by Committee

____
Date of Final Oral Examination

ACKNOWLEDGMENTS

Thanks to Dr. Weighill for all your support, I owe the confidence I have gained throughout my Master's degree in my coding and mathematical ability to your courses and guidance. Thanks to my committee members, Dr. Lewis and Dr. Hull, for your time and support. I would also like to thank all the faculty in the Mathematics Department who have been involved in the Mathematics MA and PhD programs at UNC Greensboro. I have gone through great personal, academic, and professional growth thanks to being a part of the Mathematics MA program and I hope many students after me will be able to do the same.

Contents

List of Figures

List of Tables

Chapter 1: Introduction

The goal of this paper is to apply topological data analysis (TDA) methods to topological features of 3-dimensional data from sterolithography (STL) files generated by a computer-aided design (CAD) program to measure potential errors in tolerances and clearances which can frequently arise in manufacturing methods such as 3D printing. A part's topology can be analyzed to obtain metrics that can help identify the margin of error in its features. The possible tolerances of a part can be represented by alternate 3D files which are variations of the part's features, such as a hole or extrusion. While there are many possible simple and complex features a 3D part can have, we will observe differences in the topology of fundamental part features through hole diameters, distances between two separate objects, and two different features that come together which are attached to the same object. These variations in tolerances of features will result in different birth and death values in the persistent homology of the part which can be observed through a sequence of persistence diagrams.

TDA is a set of methods in applied mathematics that use topology to understand the shape of data through transformations into simplicial complexes which allow us to visualize features in the data such as possible connected components, holes, and voids. A standard approach in TDA is to transform data into a filtered simplicial complex which will allow us to observe its persistent homology. Persistent Homology tracks changes in homology over "time", or, as changes in filtration values. For the purposes of this paper, we will only be focusing on connected components and holes.

The sample STL file data used for this paper was generated in FreeCAD and then exported to ASCII STL files, but the methods of this paper can be applied to 3D files generated by any program. The `MeshPy` Python library was used to create Delaunay meshes of each STL file, and the `Gudhi` Python library was used to create filtered simplicial complexes and to generate the persistence points of the Alpha complex of the meshes. We then used the change in filtration values of the simplicial complex for individual meshes at scale r to visualize connected components and holes represented by the H_0 and H_1 homology. Finally, for the sequence as a whole, we compared the persistent homologies of each 3D part variation to see how the changes to part features affected the topology of the part.

1.1 Related Works

Although TDA is relatively new as a field of study, it has applications in many areas. We will review related works in the fields of geospatial data, image data, medical/neuroscience, deep learning, and computational geometry.

Geospatial Data

Persistent homology can be used to detect patterns in voting data by margin of victory for presidential candidate votes using geospatial data [4]. This paper takes voting data from 24,626 California precincts and uses the geographic locations of these precincts to create an adjacency graph. The analysis is done with new methods of creating filtered simplicial complexes: the adjacency complex and the level-set complex.

Geospatial data and persistent homology can also be used to visualize changes in the movement of artic ice [7]. This paper uses artic ice binary image data from NASA from 1999 to 2009 with metadata for geographic information. The images were assigned weights by the amount of ice per image. Persistence diagrams were created from the image data to visualize increases and decreases in the amount of artic ice through the death times of connected components, or points in H_0.

Neuroscience

In the field of Neuroscience, persistent homology can be used to detect differences in cognition and personality with MRI data [1]. Resting state fMRI data of nodes in a brain network was mapped to a point cloud. Each point was assigned a value based on its level of association to other nodes in the network. A distance measure of associativity was mapped to each node in order to have larger correlations create smaller distances. The Vietoris-Rips complex was applied to the metric space of the point cloud to generate persistence barcodes in H_0 for connected components in the dataset. The persistence barcode showed differences in cognition and personality over the spatial domain.

Deep Learning

Machine learning can incorporate topology through trainable neural network units [10]. This paper uses persistence diagram dissimilarity functions, such as the Wasserstein and bottleneck distances, to act as the activation function of the unit which analyzes the shape of data input to the functional unit. The unit was appplied to classify signals and learn space encodings.

Topology can be applied to the fields of deep learning and cybersecurity to model dependencies in neural networks and detect trojaned neural networks [15]. A trojaned neural network performs a trojan attack by injecting data into a dataset. The paper uses the neural network architecture as the space to be analyzed with TDA by converting nodes and connections to vertices and edges.

Computational Geometry, Finite Element Modeling (FEM), and Meshing

Applications of topology and homology exist in meshing and finite element method applications. Homology is a fundamental building block of TDA with applications in meshing and analyzing simplicial data. Homology has many possible applications itself, especially in the field of computational geometry. Homology can be used with finite element meshing algorithms such as Gmsh, a meshing component of FreeCAD [12]; algorithms for creating meshes more efficiently [8]; and used to simplify high-dimensional simplicial shapes for easier analysis [3] [6].

A paper using methods similar to the methods of this thesis is "Computing Mitered Offset Curves Based on Straight Skeletons", which creates triangulations of polygons similar to our surface triangulations of polyhedrons [11]. A mitered offset curve is created by forming offset parallel edges to an original polygon which are offset inward from the outer edges of the polygon. A straight skeleton is created by forming lines which trace the vertices of respective edges as they shrink to the new polygon within the original. The triangulation method this paper uses to compute the mitered offset curves is kinetic triangulation, which is dynamic in accordance with moving vertices. As the mitered offset curve algorithm creates new edges, the topology of the new polygons changes per offset level, so the kinetic triangulation must adapt to these changes by taking this topology into account.

Chapter 2: Background

Homology is used to detect the holes present in a shape through the use of simplicial complexes. Before defining a simplex and simplicial complex, fundamental preliminary definitions need to be recalled to understand simplicial homology and persistent homology [9].

Definition 2.1 (Field)**.** A field is a set $\mathbb{F}$ endowed with operations addition, $+$, and multiplication, $\bullet$, respectively satisfying the following axioms for all $a, b, c \in \mathbb{F}$:

1. (Identity) $\exists$ an *additive identity*, $0_{\mathbb{F}}$ s.t. $a + 0_{\mathbb{F}} = a$ and a *multiplicative identity* denoted $1_{\mathbb{F}}$ s.t. $1_{\mathbb{F}} \bullet a = a$.
2. (Associativity) Addition and multiplication are associative:
    - $(a + b) + c = a + (b + c)$
    - $(a \bullet b) \bullet c = a \bullet (b \bullet c)$
3. (Commutativity) Addition and multiplication are commutative:
    - $a + b = b + a$
    - $a \bullet b = b \bullet a$
4. (Inverses) Each $a \in \mathbb{F}$ has *additive inverse* denoted $-a$ s.t. $a + (-a) = 0_{\mathbb{F}}$ and, excluding $0_{\mathbb{F}}$, a *multiplicative inverse* denoted a^{-1} s.t. $a \bullet a^{-1} = 1_{\mathbb{F}}$.
5. (Distributivity) Multiplication distributes over addition:
    - $a \bullet (b + c) = (a \bullet b) + (a \bullet c)$

Definition 2.2 (Field over $\mathbb{F}_2$)**.** $\mathbb{F}_2$ is a field with two elements, 0 and 1, with modulo 2 applied to addition and multiplication.

Definition 2.3 (Free Vector Space)**.** Let $\mathbb{F}$ be a field and let S be a finite set. The *free vector space* over $\mathbb{F}$ on the set S is the vector space $V_{\mathbb{F}}(S)$ (or $V(S)$ when the field is fixed) with underyling set consisting of functions $\phi : S \to F$.

Definition 2.4 (Convex)**.** A subset of a set S of $\mathbb{R}^m$ is *convex* if for any points $x, y \in S$, each point $(1-t)x + ty, t \in [0,1]$, along the interpolation between x and y is also contained in S.

Definition 2.5 (Convex Hull)**.** The *convex hull* of a set S is the smallest convex subset of $\mathbb{R}^m$ which contains S and is denoted $cvx(S)$:

$$cvx(S) = \bigcap\{C \mid S \subset C \subset \mathbb{R}^m \text{and } C \text{ is convex}\}$$

Definition 2.6 (Affine Subspace)**.** An *affine subspace* of $\mathbb{R}^m$ is a set of the form $x + V = \{x + v \mid v \in V\}$ where $x \in \mathbb{R}^m$, $V \subset \mathbb{R}^m$ is a vector subspace.

Definition 2.7 (General Position)**.** Let $S = \{x_0, x_1, ..., x_n\}$ be a finite subset of $\mathbb{R}^m$. S is in *general position* if its points are not contained in any affine subspace of $\mathbb{R}^m$ of dimension less than n where $n \leq m$.

Definition 2.8 (Open Metric Ball)**.** An *open metric ball* in a metric space (X, d), notated $B(x, r)$, is defined for a center point $x \in X$ and a radius $r > 0$ to be the set:

$$B(x, r) = \{y \in X \mid d(x, y) < r\}$$

Definition 2.9 (Voronoi Cell)**.** For each point in x in a set of points $X \subset \mathbb{R}^m$ with distance d, the *Voronoi Cell* of x is the set of points in $\mathbb{R}^m$ closest to x:

$$\mathsf{Vor}(x) = \{s \in \mathbb{R}^m \mid d(x, s) \leq d(x, t) \forall t \in \mathcal{X}\}$$

2.1 Simplicial Homology

Simplicial homology analyzes shapes of dimension m through combinations of fundamental components such as points, edges, triangles, tetrahedra, and so on.

Definition 2.10 (Simplex)**.** For a set $\mathcal{X}$ of n points in general position, an $(n-1)$-dimensional *simplex*, $\sigma(\mathcal{X})$, associated to $\mathcal{X}$ is formed by the convex hull of $\mathcal{X}$, $cvx(\mathcal{X})$.

For our purposes, we will only be working over $\mathbb{F}_2$ with 0, 1, 2, and 3-simplices. A 0-simplex is a point; a 1-simplex is an edge; a 2-simplex is a "filled-in" triangle, or the convex hull of a triangle; and a 3-simplex is a "filled-in/solid" tetrahedron, or the convex hull of a tetrahedron.

Definition 2.11 (Geometric Simplicial Complex)**.** A *geometric simplicial complex* is a collection of simplicies $\mathcal{X}$ in $\mathbb{R}^m$ satisfying:

1. for any simplex $\sigma \in \mathcal{X}$, all faces of σ are also contained in $\mathcal{X}$.

2. for any two simplices $\sigma, \tau \in \mathcal{X}$, the non-empty intersection $\sigma \cap \tau$ is also a simplex and a face of both σ and τ.

Definition 2.12 (Abstract Simplicial Complex)**.** An *abstract simplicial complex* is a pair $\mathcal{X} = (V(\mathcal{X}), \Sigma(\mathcal{X}))$, also notated $\mathcal{X} = (V, \Sigma)$, where $V(\mathcal{X})$, the vertices of $\mathcal{X}$, is a finite set and $\Sigma(\mathcal{X})$, the simplices (or faces) of $\mathcal{X}$, is a collection of subsets of $V(\mathcal{X})$, such that for any $\sigma \in \Sigma(\mathcal{X})$ and any nonempty $\tau \subset \sigma, \tau \in \Sigma(\mathcal{X})$. Faces containing exactly two vertices are *edges*. Faces containing exactly $(k+1)$-vertices are *k-dimensional faces* (or *k-faces*).

Definition 2.13 (Chain Group)**.** The *k-th chain group* $C_k(\mathcal{X})$ of $\mathcal{X}$ over F_2, or C_k when the simplicial complex $\mathcal{X}$ is fixed, is the free vector space over F_2 generated by the set of *k-dimensional* simplices of $\mathcal{X}$.

Definition 2.14 (Boundary Map)**.** The *boundary map*, ∂_k, of a simplicial complex, X, defined on basis elements σ is:

$$\partial_k(\sigma) = \sum_{j=0}^{k} \partial_k^j(\sigma),$$

where ∂_k^j is a $(k-1)$-*dimensional* face of *sigma*. ∂_k takes a k-simplex to the sum of $(k-1)$-simplices which lie along its boundary. It can also be expressed as:

$$\partial_k(\sigma) = \sum_{j=0}^{k} \Big\{ \tau \mid \tau \text{ is a } (k-1)\text{-dimensional face of } \sigma \Big\}$$

Definition 2.15 (Cycle)**.** The *cycle* group, $Z_k(\mathcal{X})$ or Z_k, is the kernel of the boundary map ∂_k. The elements of Z_k are referred to as *k-cycles*.

Definition 2.16 (Boundary)**.** The *boundary* group, $B_k(\mathcal{X})$ or B_k, is the image of the boundary map ∂_{k+1}. The elements of B_k are reffered to as *k-boundaries*.

Through building on the previous definitions, we are finally ready to define homology groups through the use of cycles and boundaries.

Definition 2.17 (Homology Group)**.** The *k-th homology group* of $\mathcal{X}$ is the quotient space $H_k(\mathcal{X}) = Z_k(\mathcal{X})/B_k(\mathcal{X})$.

Different homology groups give us information about different features in $\mathcal{X}$. H_0 describes the number of connected components, H_1 describes the number of holes, and H_2 describes the number of voids.

2.2 Persistent Homology

Persistent homology measures the change in the homology of features over "time", or marked events used to keep track of the changes. Persistence diagrams keep track of the persistence points that arise over the course of these changes through recording birth and death moments of simplicies in the complex.

Definition 2.18 (Filtered Simplicial Complex)**.** A *filtered simplicial complex* is a collection of $\mathsf{K} = \{K_r\}_{r\geq 0}, r \in \mathbb{R}$, of (finite) simplicial complexes, $K_{r_1} \subset K_{r_2} \subset K_{r_3} \subset ... \subset K_{r_n}$, and a family of inclusion maps $f_{r,r'} : V(K_r) \to V(K_{r'})$ for each pair (r, r') with $r \leq r'$. The inclusion maps must satisfy the compatibility condition: for all $r \leq r' \leq r''$, we have $f_{r,r''} = f_{r'',r'} \circ f_{r',r}$. The inclusion maps induce maps on homology $i_{m_*} : H_k(K_{r_m}) \to H_k(K_{r_{m+1}})$ for each k.

Definition 2.19 (Persistence Diagram)**.** The Fundamental Theorem of Persistent Homology [16], states that an input of a filtered simplicial complex, $K_{r_1} \subseteq K_{r_2} \subseteq ... \subseteq K_{r_t}$ gives a basis of homology groups $H_k(K_{r_1}) \to H_k(K_{r_2}) \to ... \to H_k(K_{r_t})$ with induced maps on homology $i_{1*}, ..., i_{t*}$ such that each basis element has a well-defined birth time, b, and death time, d.

The *persistence diagram* is obtained from the multiset of these births and deaths, $\{(b_j, d_j)\}_{j=1}^{M}$, by plotting each point in the xy-plane (recording multiplicity as necessary). Since $b_j < d_j$ for each j, the points all appear above the diagonal line $x = y$.

There are many different ways to create a filtered simplicial complex. We will define three methods we can use in order to create the complex. The Čech complex is the base complex we will build off of.

Definition 2.20 (Čech Complex)**.** Let X be a finite metric space. The *Čech complex*, C_r, can be applied to X, where open metric balls grow around points of the metric space. With each intersection of $k + 1$ balls, add a *k-dimensional* simplex to the filtered simplicial complex.

$$\mathsf{C}_r(X) = \{\sigma \subseteq X \mid \cap_{x\in\sigma} B_r(X) \neq \emptyset\}$$

The Vietoris-Rips complex expands upon the Čech complex by introducing a fixed radius.

Definition 2.21 (Vietoris-Rips Complex)**.** Let (X, d) be a finite metric space. The *Vietoris-Rips complex* at parameter r, $\mathsf{VR}(X, r)$, for each real number $r \geq 0$, can be applied to X, where:

$$\mathsf{VR}(X, r) = (V_r, \Sigma_r),\ V_r = X \text{ for all } r,\ \Sigma_r = \{\sigma \subset X \mid d(x, y) \leq r\ \forall x, y \in \sigma\}$$

Example 2.22 (Vietoris-Rips Complex applied to a Point Cloud)**.** The Vietoris-Rips complex is created from connections between points which form when the distance of radii between points intersect. When the radii of two points intersect, a connection appears as a 1-simplex, or an edge. When three radii intersect, a 2-simplex, or a "filled-in" triangle, appears. These connections keep forming until a chosen final radius, r. In Fig 2.1, the Vietoris-Rips complex grows until a radius of 1.0.

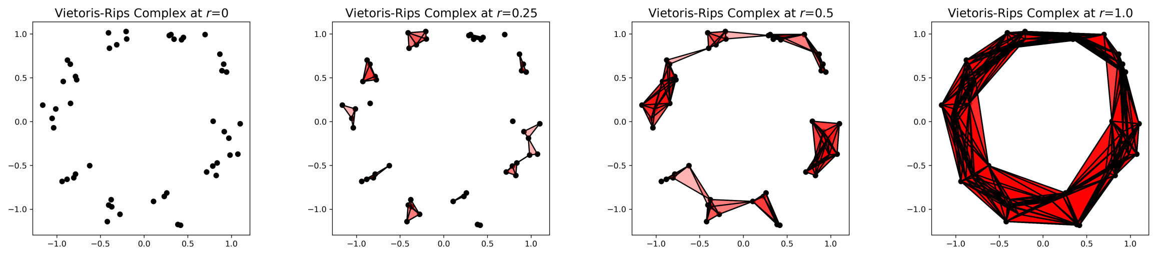


Figure 2.1. Vietoris-Rips complex of a point cloud as radius r increases.

Example 2.23 (Persistence Diagram of a Vietoris-Rips Complex of a Point Cloud)**.** When $r = 1.0$, the persistence diagram shows one point at $d = \infty$ for H_0. This corresponds to the connected component in the filtered simplicial complex. The points below infinity on the y-axis show the death times of points as they get added to the connected component. Notice how there are only points on the y-axis for H_0: as Figure 2.1 shows, all points are present when $r = 0$, this means each point in the complex has a birth time of 0. There is also a point in H_1 point which represents the hole that is formed at a radius of approximately 0.6 and dies at a radius of approximately 1.45.

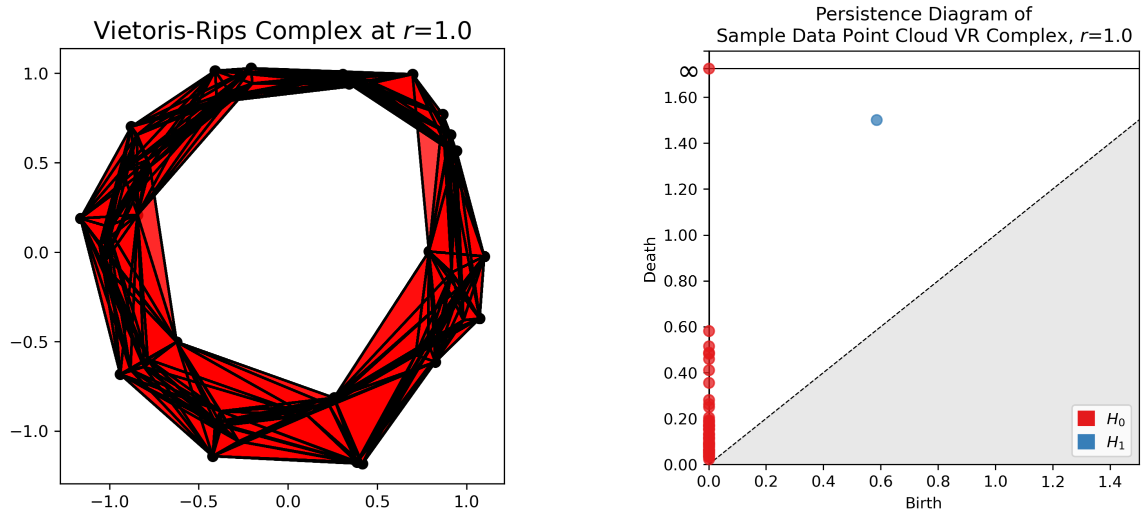


Figure 2.2. Plot of a point cloud and persistence diagram of its Vietoris-Rips complex.

The Čech and Vietoris-Rips complexes can be very computationally expensive, so an alternative filtered simplicial complex, the Alpha Complex, will need to be applied to the STL file data. Before we can define the Alpha Complex, we need to define an essential component of it, the Delaunay Triangulation.

Definition 2.24 (Delaunay)**.** We say a k-simplex whose vertices are in a set of points, V is *Delaunay* if there exists a circumsphere in that k-simplex such that no vertex of V lies inside it [13].

Definition 2.25 (Delaunay Triangulation)**.** A *Delaunay triangulation* is a unique triangulation on a vector space which is formed by creating a connection between points whenever their Voronoi cells (Def 2.9) intersect such that all simplices in the space are Delaunay. This triangulation is the dual of its Voronoi cells [2][13].

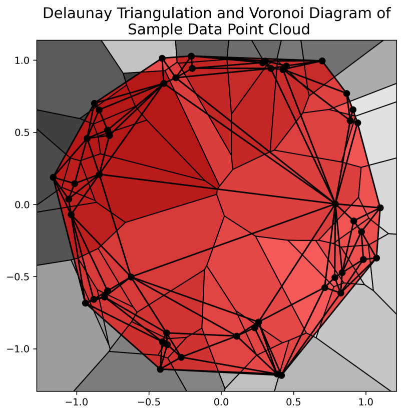


Figure 2.3. Plot of a Delaunay Complex and Voronoi Cells applied to a point cloud.

Definition 2.26 (Gabriel)**.** When a mesh is created with a Delaunay Triangulation, certain edges in the triangulation can be considered *Gabriel* when the open ball of the triangulated edge is empty of points [5]. However, not every edge in a triangulation needs to be considered Gabriel for it to be considered Delaunay. In $\mathbb{R}^2$, the figure below shows that every open ball, or circumcircle, of an edge contains no inner points.

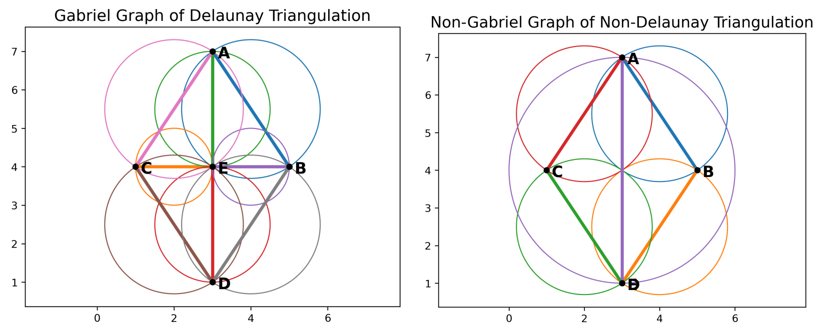

Gabriel Graph of Delaunay Triangulation
Non-Gabriel Graph of Non-Delaunay Triangulation
A
B
C
D
E
0
2
4
6
1
2
3
4
5
6
7

Figure 2.4. A Gabriel graph compared to a graph which cannot be Gabriel.

Definition 2.27 (Alpha Complexes)**.** The Alpha complex (or α-complex) exists within a finite metric space with Euclidean distance. The alpha complex builds upon the Delaunay Triangulation with its use of Voronoi cells by introducing a radius parameter for an open metric ball.

$$\alpha_r(X) = \{\sigma \subseteq X \mid \cap_{x\in\sigma}(B_r(X) \cap \mathsf{Vor}_x) \neq \emptyset\}$$

By convention we report birth and death times of points with r^2. At $r^2 = 0$, the filtered simplicial complex consists only of the set of points in the metric space as individual connected components. As r^2 increases, the balls grow up to the boundary of or beyond their Voronoi cell. When two balls intersect, a connected component between those two points is formed. This continues until r^2 grows to create a simplex equivalent to the Delaunay triangulation.

The Alpha complex is the ideal filtered simplicial complex to use when analyzing the homology of 3D objects as it can show us when connected components and holes are born and die in the persistent homology of the object while the growth of r^2 is self-contained to the maximum diameter of the object. The end growth point being the object's Delaunay triangulation also allows for the Alpha complex to be the least computationally expensive persistent homology method of the three we have defined.

Example 2.28 (Alpha Complex applied to a Point Cloud)**.** Fig 2.5 shows an Alpha complex created with the python library `Gudhi`. To understand events at certain radii, we denote plots with r^2 for birth and death times, and r for Euclidean distance.

Two holes were created with birth times of $r^2 = 0.2308$ and $r^2 = 0.3068$. The first hole dies at $r^2 = 0.8045$. To understand why the second hole dies relatively quickly at $r^2 = 0.3069$, we can compare it to the filled-in triangle formed between $r^2 = 0.2308$ and $r^2 = 0.2900$. The points of the triangle which make up the second hole have voronoi cells which intersect close to or exactly at the intersection point of all three voronoi cells. As one of the three balls grows slightly, the hole gets filled in $r^2 = 0.0001$ later. Three voronoi cells of points from the filled-in triangle all intersect at different times, creating the filled-in triangle instantly upon the third intersection.

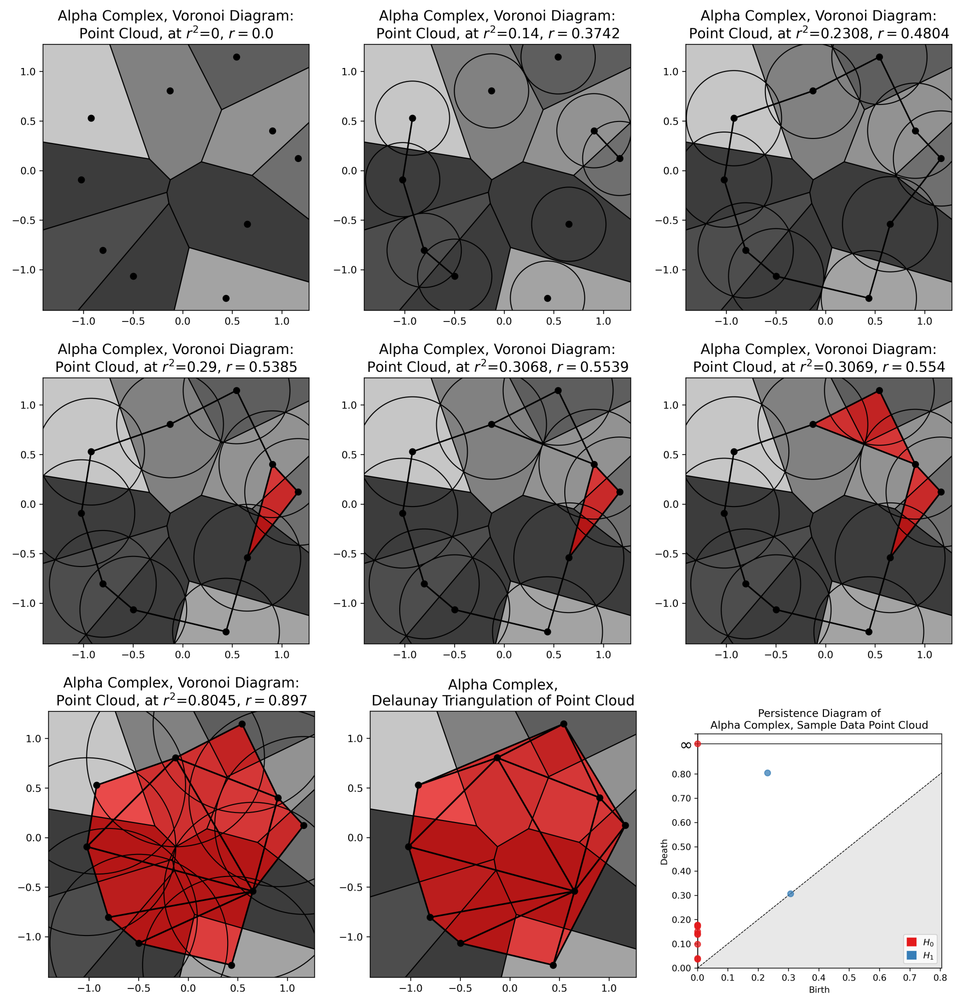


Figure 2.5. Progression of the Alpha complex on a point cloud and the persistence diagram of the completed complex.

Example 2.29 (Persistence Diagram of an Image)**.** Fig 2.6 uses a grayscale image of a white square with a black circle in the center to create a persistence diagram from an Alpha complex. Unlike previous examples, the dataset that creates the image is not a point cloud but is image data as a 50×50 pixel array of decimal values. The white pixels in the image contain values of 1.0 and the black pixels contain no data, or values of 0.0. This causes the black circle to appear as a hole in the data.

This hole can be seen as a point in H_1 with a birth time of 0 and a radius of 10 pixels. As r increases to a Euclidean distance of 10, the hole gets filled in because balls on the circumference of the black hole finally intersect. The intersection at $r = 10$ is shown as a death time of $r^2 = 100$.

On a persistence diagram, in most cases, the points along or relatively close to the diagonal line which are not on the y-axis can be considered noise and disregarded.

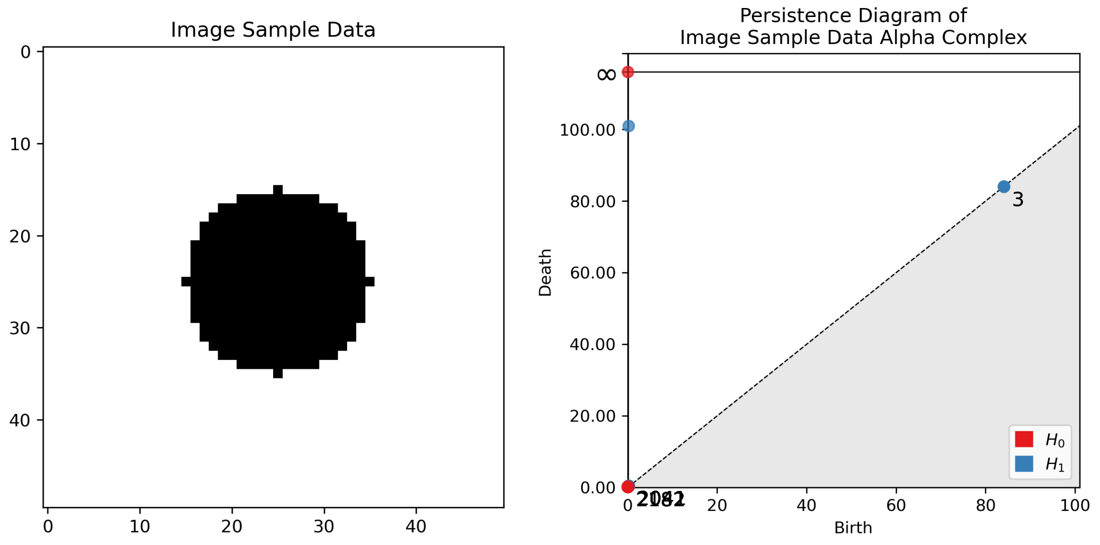


Figure 2.6. Plot of a white square with a hole and persistence diagram of its Alpha Complex.

2.3 Background on the STL Filetype

An STL file is a filetype created for use with 3D data generated by computer-aided design (CAD) programs for 3D printing and other rapid-prototyping applications. STL is acronym which can be an abbreviation for Stereolithography, Standard Triangle Language, or Standard Tesselation Language. These files are built through triangulation: every set of points creates a triangle which acts as a facet of the object. These triangles are "filled-in" 2-simplices, and will be fundamental in analyzing the topology of STL files. In terms of the contents of an ASCII STL file, a facet should not be confused with a face: a facet must be a triangle, while a face of a polyhedron can be subdivided into multiple facets to achieve triangulation. The information in an STL file can be encoded using binary or in ASCII as a `.ast` (ASCII STL) file. For our purposes, we will be using `.ast` files to extract and analyze 3D object data [14].

```
solid Mesh
  facet normal 0.000000 0.000000 -1.000000
    outer loop
      vertex -0.500000 -0.288675 0.000000
      vertex 0.000000 0.577350 0.000000
      vertex 0.500000 -0.288675 0.000000
    endloop
  endfacet
  facet normal -0.816497 0.471405 0.333333
    outer loop
      vertex 0.000000 0.000000 0.816497
      vertex 0.000000 0.577350 0.000000
      vertex -0.500000 -0.288675 0.000000
    endloop
  endfacet
  facet normal 0.000000 -0.942809 0.333333
    outer loop
      vertex 0.000000 0.000000 0.816497
      vertex -0.500000 -0.288675 0.000000
      vertex 0.500000 -0.288675 0.000000
    endloop
  endfacet
  facet normal 0.816497 0.471405 0.333333
    outer loop
      vertex 0.000000 0.000000 0.816497
      vertex 0.500000 -0.288675 0.000000
      vertex 0.000000 0.577350 0.000000
    endloop
  endfacet
endsolid Mesh
```

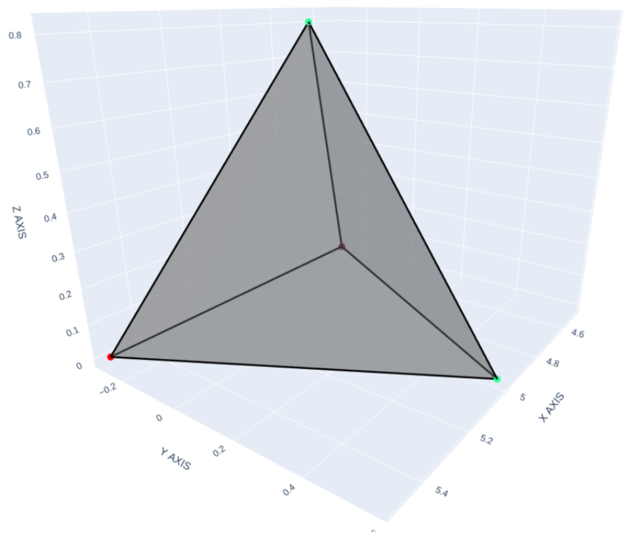


Figure 2.7. The contents and 3D plot of an ASCII STL file for a tetrahedron.

STL files contain descriptions of facets which make an object (or objects). Fig 2.7 shows these descriptions contain the facet normal vectors and the three vertices used to create it. The contents of an STL file do not specify topological information, such as connections to other vertices, which vertices make up a face, etc. Instead, STL files describe each vertex as many times as it appears in the file until the overall shape is built. To analyze the topological data within STL files, we can parse the `.ast` files for four strings which are used to denote the beginning and end of the description of facets and vertices: ‘facet normal’, ‘end facet’, ‘outer loop’, and ‘end loop’, respectively.

Chapter 3: Methods

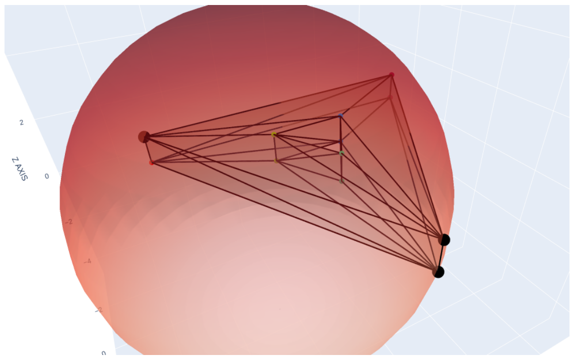
3.1 Main Method

3.1.1 Creating a Mesh from an STL file

To compute the persistence of the variations of the "truth" of an STL file, the data must be processed and transformed with meshing in order to be properly analyzed. Although STL files contain a mesh upon creation, this mesh is not adequate for our purposes as the triangulated mesh an STL file contains is not necessarily a subcomplex of the Delaunay triangulation. We will call a simplex a *Delaunay simplex* if it is contained in the Delaunay triangulation of the vertices of the mesh. A simplicial complex is called a *sub-Delaunay complex* if every simplex in it is Delaunay. The completed Alpha complex of a mesh is its Delaunay triangulation, so we require the 3D object's original mesh to be sub-Delaunay because in the next step of our method we will alter the filtered simplicial complex based on the data of the initial simplicial complex, K_0. K_0 will be compared with the Alpha complexes as r^2 grows to include new simplices. Without K_0 being sub-Delaunay, our filtration and subsequent persistence analysis may show incorrect results.

Example 3.1 (Comparison of ASCII STL mesh and Delaunay Triangulation Mesh)**.** Fig 3.1a shows the open circumsphere of a facet, emphasized by three large black points, and its circumsphere on the original triangulation for an ASCII STL file. This facet's circumsphere does contain all three vertices of the facet on its surface boundary but it also contains every vertex which makes up the hole in the center of the object within the interior of the circumsphere. This inclusion of vertices within the circumsphere means the entire object's triangulation is not Delaunay.

Fig 3.1b shows the open cirumspheres of three facets of a Delaunay triangulation of the same object. The surface boundary of each facet's circumsphere contains the three vertices of its respective facet while containing other facet vertices on its surface boundary as well. The circumsphere only containing vertices on its surface boundary but not within its interior means the object's new mesh is a Delaunay triangulation.


(a) Original mesh from ASCII STL.

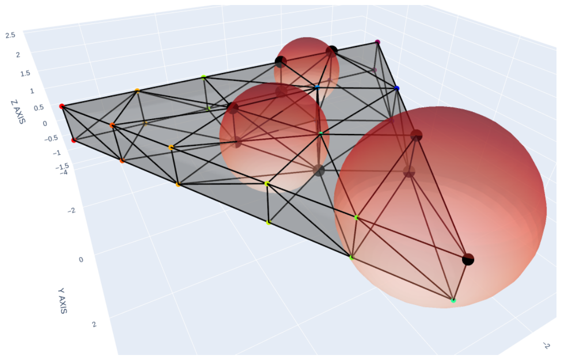

(b) Delaunay triangulation mesh.

Figure 3.1. The original input mesh of the data from an ASCII STL file and its Delaunay triangulation mesh.

3.1.2 Creating and modifying an Alpha Complex

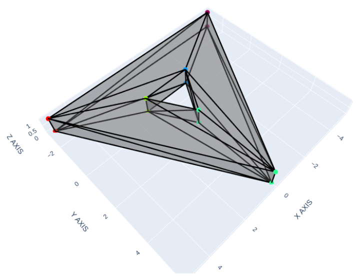
Now we are ready to generate an Alpha complex. The sub-Delaunay complex mesh K_0, created by the Delaunay triangulation, contains every edge and triangle present in the original 3D object mesh while retaining its boundaries, so the topology is not altered as it will be with the Alpha complex. The sub-Delaunay K_0 is a subset of the completed Alpha complex, therefore every simplex present in the K_0 complex is present as a simplex in the Alpha complex of the vertices for some scale parameter r.

The Alpha complex grows to include simplices which were not originally in the K_0 sub-Delaunay complex. These Alpha complex simplices are checked against the K_0 simplices to determine which simplices were present originally. All simplices that were present originally will be given a new filtration value of 0 to denote they appear in the K_0 complex. Replacing the filtration values in this way is done to ensure the original simplices from K_0 have a corresponding birth time of $b = 0$, while all new simplices created by the Alpha complex will retain their assigned r^2 filtration.

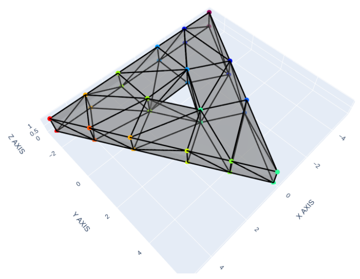
3.1.3 Computing a Persistence Diagram

Now that we have created and reassigned the appropriate filtration values that make up our filtered simplicial complex, we compute a persistence diagram using Def 2.19. We then create persistence diagrams of each variation of the 3D object to see the progression of its homology over the changes to a feature or features of a part. The death axes on the sequence of persistence diagrams are constrained to the maximum death time within the sequence to keep the plots consistent and monitor its change over the variations.

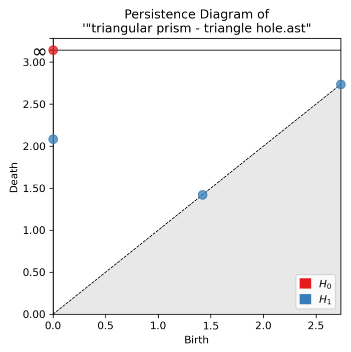
Example 3.2 (Example of Methods with a Single 3D Object)**.** Fig 3.2 shows the progression of transformations to the STL file data. Initially, the object is loaded in and parsed to create the 3D plot of its original mesh in Fig 3.2a. Then, the data is meshed to create a Delaunay Triangulation (additional points are added to the mesh which will be expanded on in Sec 3.2.3) in Fig 3.2b. Finally, we compare the simplices in 3.2b to the simplices in the alpha complex of the meshed 3D object and plot its persistence in 3.2c to show 1 connected component in H_0 and 1 hole in H_1.



(a)



(b)



(c)

Figure 3.2. A triangular prism with a triangular hole, its Delaunay triangulated mesh, and the persistence diagram of the Delaunay triangulated mesh

3.2 Implementation

The methods of this paper are implemented in Python using many libraries, most notably `MeshPy` for mesh creation, `Gudhi` for TDA, `Plotly` to visualize 3D plots, and `Matplotlib` to visualize persistence diagrams.

3.2.1 Creating STL Files with FreeCAD

STL files from online sources such as Thingiverse can be processed through the program, but these files are often too complex and therefore very computationally expensive. For this reason, we create our own dataset to test our methods using FreeCAD to keep complexity to a minimum while covering basic test cases that demonstrate potential tolerancing errors. The parts are all exported to an ASCII STL (`.ast`) which encodes the data as text to allow for the data to be easily read in by Python.

3.2.2 Parsing the STL File Data

We will discuss two methods to load in the data an ASCII STL file. We can parse data directly from the `.ast` file and load it into lists and tuples, or we can use the `load_stl()` attribute of the `MeshPy` `MeshInfo` object.

For "simplicity", we will use the `load_stl()` attribute. To use this method with a `.ast` file, we will pass in the full file path of the `.ast` file, but first we must use Python to create a temporary copy with the extension renamed to `.stl` even though the content of the file will stil be ASCII text. Without creating a temporary copy that renames the `.ast` to `.stl`, `load_stl()` will not work. This temporary `.stl` file is then passed in to `load_stl()` to create our `MeshInfo` object which will be used to build the new Delaunay triangulation mesh.

Although we use `MeshPy` to load in the data, the following paragraph is an example of how ASCII STL file data could be loaded in if we wanted to do it manually. Loading in the code manually can be advantageous if we want to directly alter the ASCII STL data of the chosen "truth" `.ast` file to create variations of part features by manipulating the facets and vertices and saving these manipulations as new `.ast` files to be analyzed sequentially.

As shown in Fig 2.7, an ASCII STL file is delimited by the formatting of its facets and vertices. We use the strings ‘facet normal’, ‘end facet’, ‘outer loop’, ‘end loop’, and newline character delimiters to identify the faces and vertices of the 3D object. For each facet, the ASCII data gives us the normal vector of the facet and the coordinates of each of the three points that makes up the facet. At this point, we can use the parsed data to add or manipulate points, edges, or facets if we choose to do so. We can then use the parsed or manipulated data to create a list of points where each unique point is numbered numerically. All point names would be 1-indexed, or begin naming at 1, to match the indexing of `Gudhi`. The list of unique edges can be extracted from the list of facets and matched with their vertex names to create a list of edges described by two vertex names instead of coordinates. The same can be done to describe each facet by the three vertex names that create it. This relational point name data can then be passed in to `Gudhi` if the triangulation is Delaunay.

3.2.3 Creating a Delaunay Triangulation with `MeshPy`

`MeshPy` is a library that creates meshes by adapting the TetGen mesh generation program to python. As per the TetGen documentation [13], Tetgen creates Delaunay tetrahedralizations (triangulations for our purposes) by default. Additional switches can be added to the code to enhance or alter the mesh. While there are dozens of possible switches to use, for simplicity, we will only be using `-p` and `-q`.

Switch `-p`: Tetrahedralize a Piecewise Linear Complex

The switch `-p` Tetrahedralizes a piecewise linear complex (PLC). According to the documentation, a PLC is defined as the following:

Definition 3.3 (Piecewise Linear Complexes)**.** A 3D *piecewise linear complex* $\mathcal{X}$ is a set of cells that satisfies the following properties:

1. The boundary of each cell in $\mathcal{X}$ is a union of cells in $\mathcal{X}$
2. If two distinct cells $f, g \in \mathcal{X}$ intersect, their intersection is a union of cells in $\mathcal{X}$.

The boundary of a 3D PLC is the set of cells whose dimensions are ≤ 2:t

1. A 0-dimensional cell is a vertex
2. A 1-dimensional cell is a segment, or an edge
3. A 2-dimensional cell is a facet

The `-p` is how Fig 3.1 retains its original shape's boundary but consists of new points and edges to allow for the mesh to be Delaunay. However, the specific Delaunay triangulation used with this switch is a Constrained Delaunay Tetrahedralization (CDT - again, triangulation for our purposes). A CDT differs slightly from a Delaunay triangulation in that open circumspheres of facets on the outer surface of a mesh are allowed to contain other facet's vertices on its surface boundary, as seen in Fig 3.1b.

Switch `-q`: Refine Mesh

The `-q` switch can be used in combination with `-p` to improve the quality of the mesh by inserting additional points into the mesh. This is done with two constraints. The first constraint is a *maximum radius-edge ratio bound*. This ratio is found by dividing the radius of the circumsphere of a tetrahedron (facet, in our case) by the length of the shortest edge of that facet. This constraint limits the maximum ratio up to a default value of 2.0. A smaller ratio means more triangulations will be made. The second constraint is a *minimum dihedral angle bound*. A dihedral angle is the angle between two planes. This constraint has a default value of 0 degrees. Choosing a larger minimum dihedral angle bound will cause the facets to be larger than they would be otherwise.

3.2.4 Creating and modifying an Alpha Complex with `Gudhi`

`Gudhi` is a python library which contains functions to analyze the topology of a dataset. We will be using it to do this through generating an Alpha complex and modifying it as outlined in Sec 3.1.2. By comparing the simplices of the Delaunay triangulation mesh generated by `MeshPy` and the simplices of the alpha complex generated by `Gudhi`, we change the filtration values so that every original simplex from the Delaunay triangulation has a filtration value of 0 and appears in K_0 while the remaining simplices generated from the Alpha complex will retain their assigned filtration.

3.2.5 Persistence Diagram Construction

Using the `Matplotlib` python library in tandem with the persistence points generated by `Gudhi`, we create a persistence diagram, like the persistence diagram of the tetrahedron from Fig 2.7. We will use a sequence of diagrams to show the progression of change in the topology of variations of 3D object features.

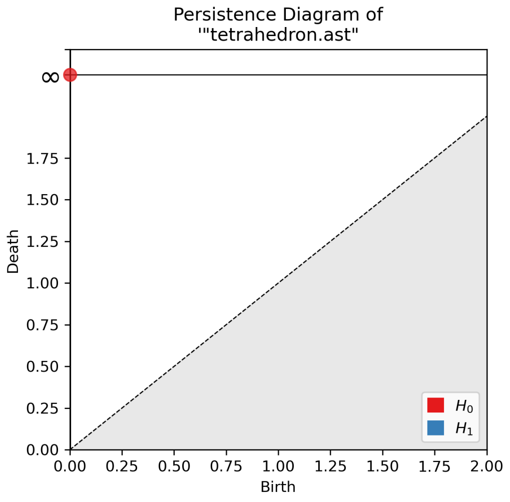


Figure 3.3. Persistence diagram of a tetrahedron.

Chapter 4: Results

The datasets used for these experiments was created with the CAD software FreeCAD, then exported to an ASCII STL as an input to `MeshPy`. The images of the 3D objects are displayed with the Python library `Plotly`. For each experiement, a feature of a file was chosen to be manipulated by changing a measurement over a series of alternate version files. This was done to show how our method lets us observe the change in homology as these measurements change across the file variations. We will dicuss the results in Chapter 5.

4.1 Two Cubes with Three Pockets Moving Closer

A CAD pocket operation subtracts an area from a face by some depth. This can also create a hole which goes through an object's face. This operation was done to a cube on the top, front, and right faces to create three intersecting pockets through each face of the cube, as shown in Fig 4.1.

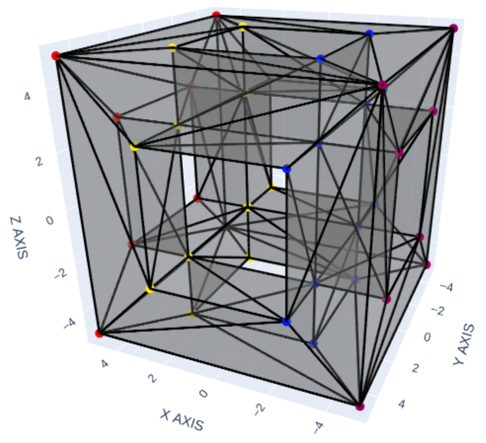


Figure 4.1. Cube with three pockets.

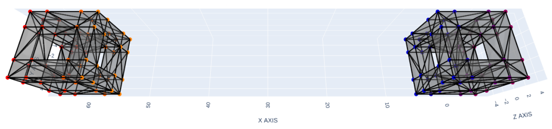
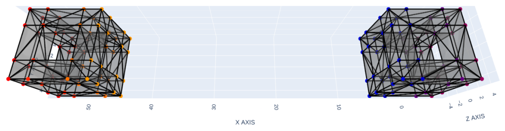
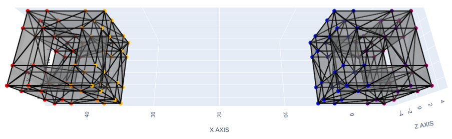
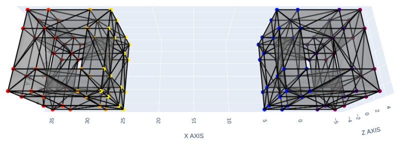
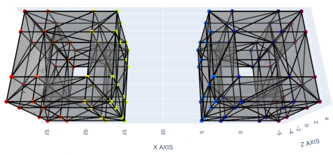
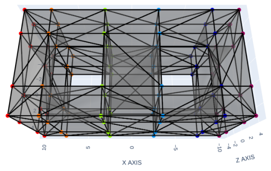
 50 mm apart	 40 mm apart
 30 mm apart	 20 mm apart
 10 mm apart	 00 mm apart

Figure 4.2. MeshPy plots displayed with Plotly of two cubes with three CAD pocket operations that move closer together until combining.

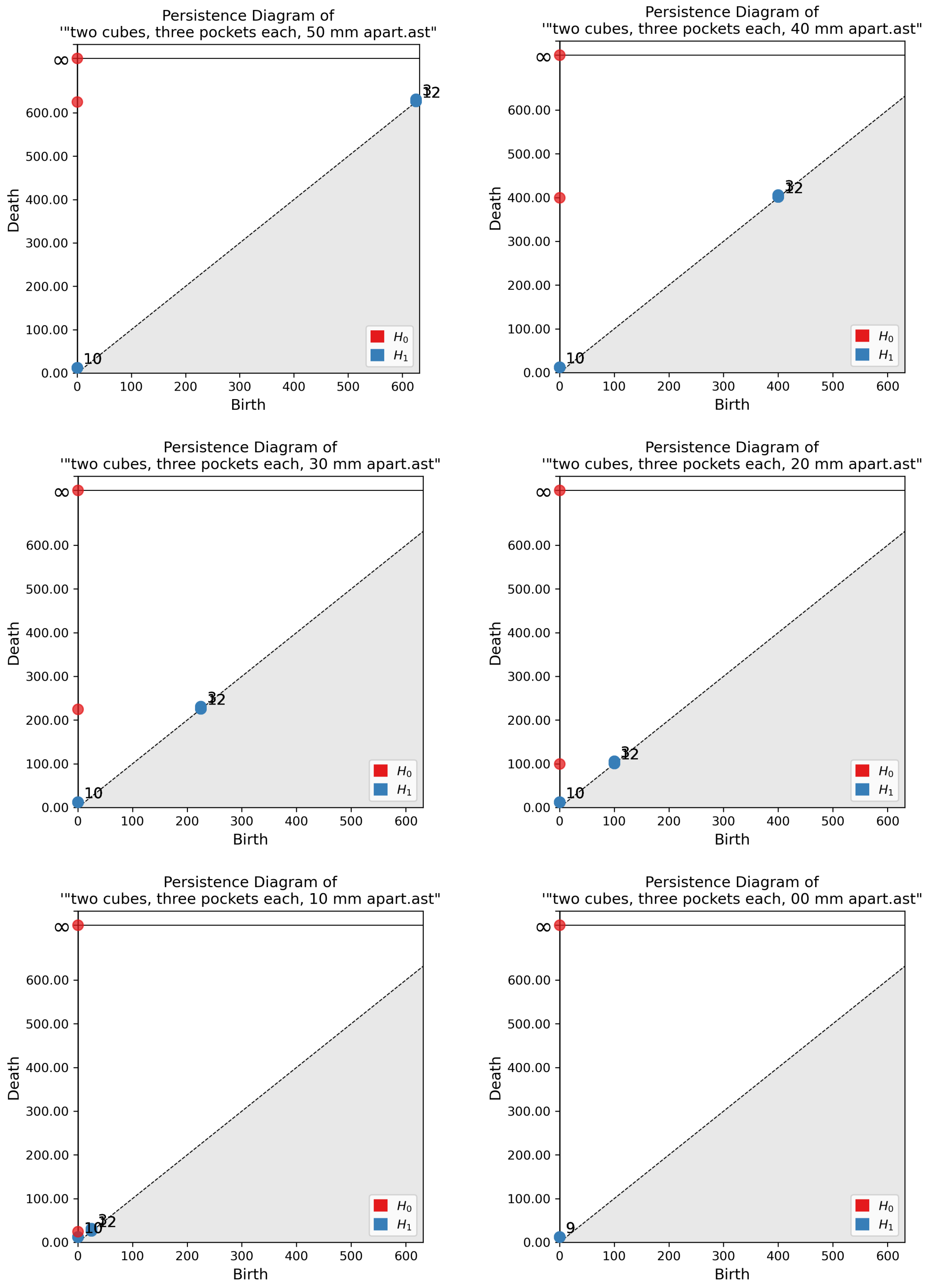
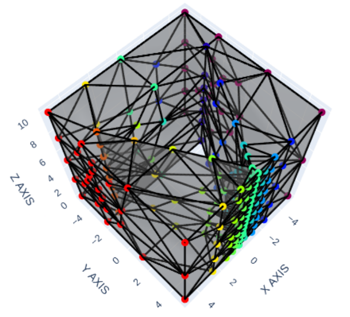


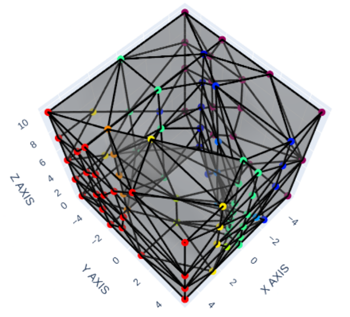
Figure 4.3. Persistence Diagrams of two cubes with three CAD pocket operations that move closer together until combining.

4.2 Cube with Equilateral Triangle Hole

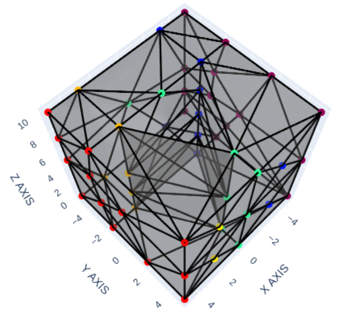
An equilateral triangle was sketched onto the top face of a cube and a pocket operation was done through the cube. The equilateral triangle hole starts with an edge length of 8mm until it shrinks completely and dissapears.



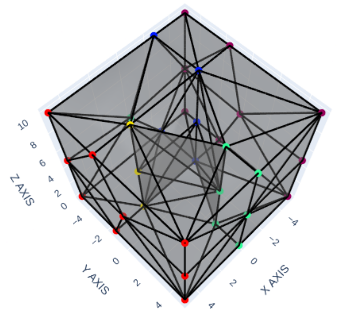
Equilateral Triangle Hole, 8 mm



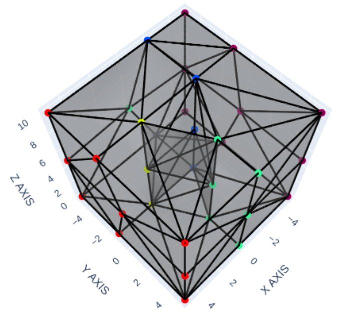
Equilateral Triangle Hole, 7 mm



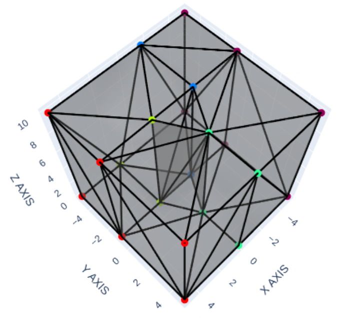
Equilateral Triangle Hole, 6 mm



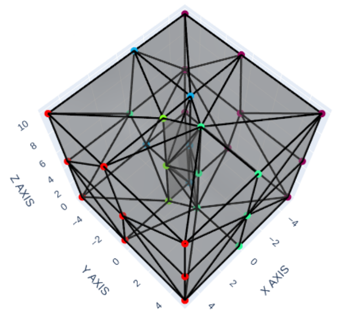
Equilateral Triangle Hole, 5 mm



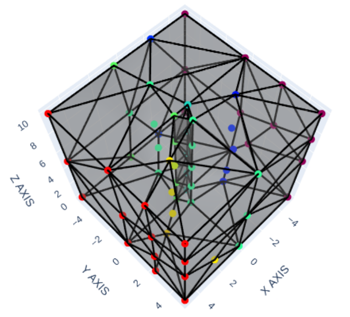
Equilateral Triangle Hole, 4 mm



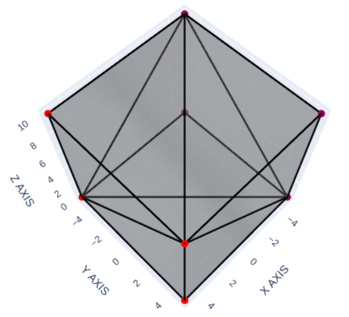
Equilateral Triangle Hole, 3 mm



Equilateral Triangle Hole, 2 mm



Equilateral Triangle Hole, 1 mm



Equilateral Triangle Hole, 0 mm

Figure 4.4. MeshPy plots displayed with Plotly of a cube with an equilateral triangle hole that decreases in size to the original shape.

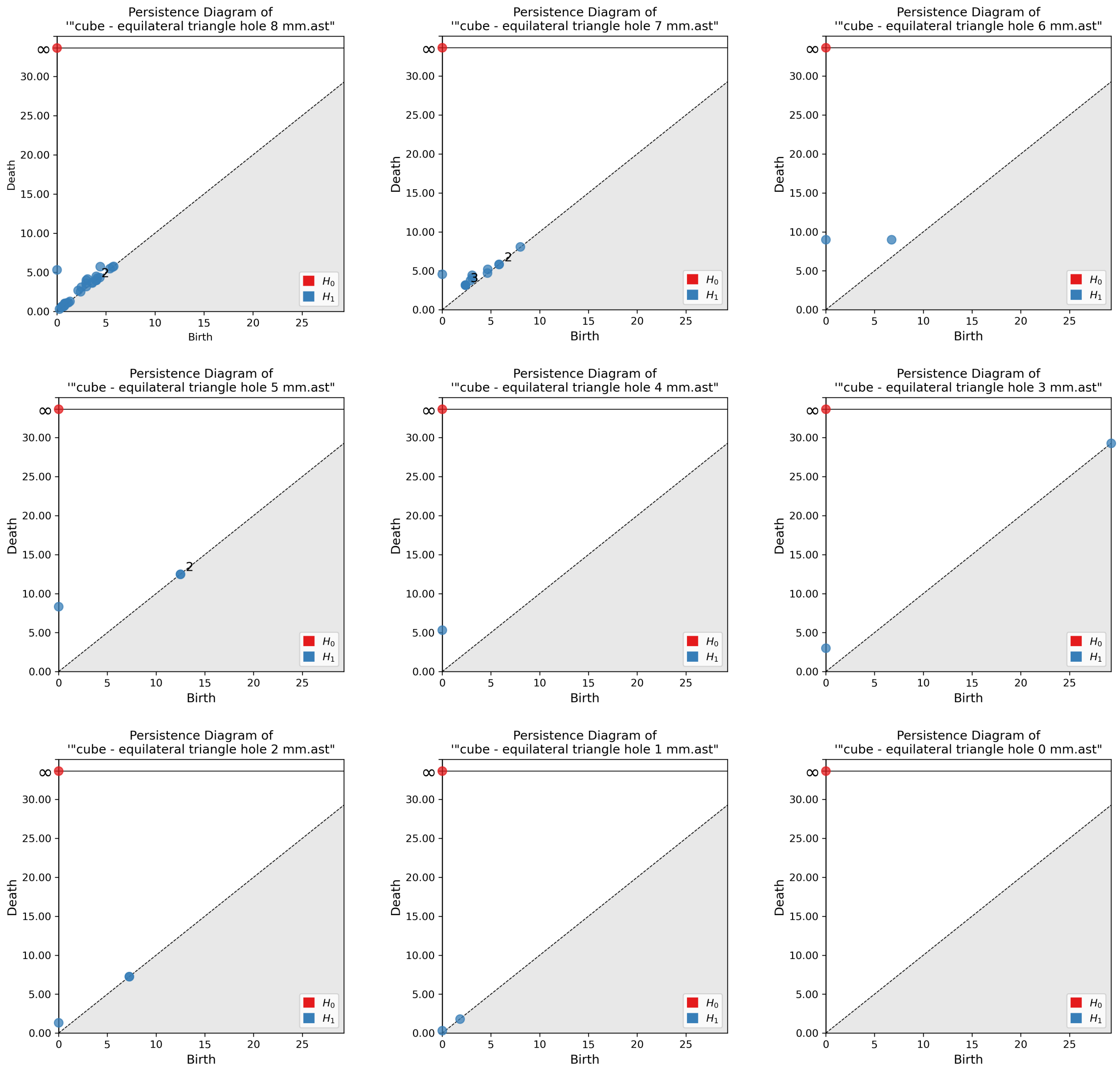


Figure 4.5. Persistence Diagrams of a cube with an equilateral triangle hole that decreases in size to the original shape.

4.3 Rectangular Prism Ring with Cut

A rectangular prism ring was created as a topologically equivalent model of a solid torus. This was done to allow for faster compute times from a simpler triangulation with less facets. The initial state of the object has a 40mm cut which shrinks and dissapears.

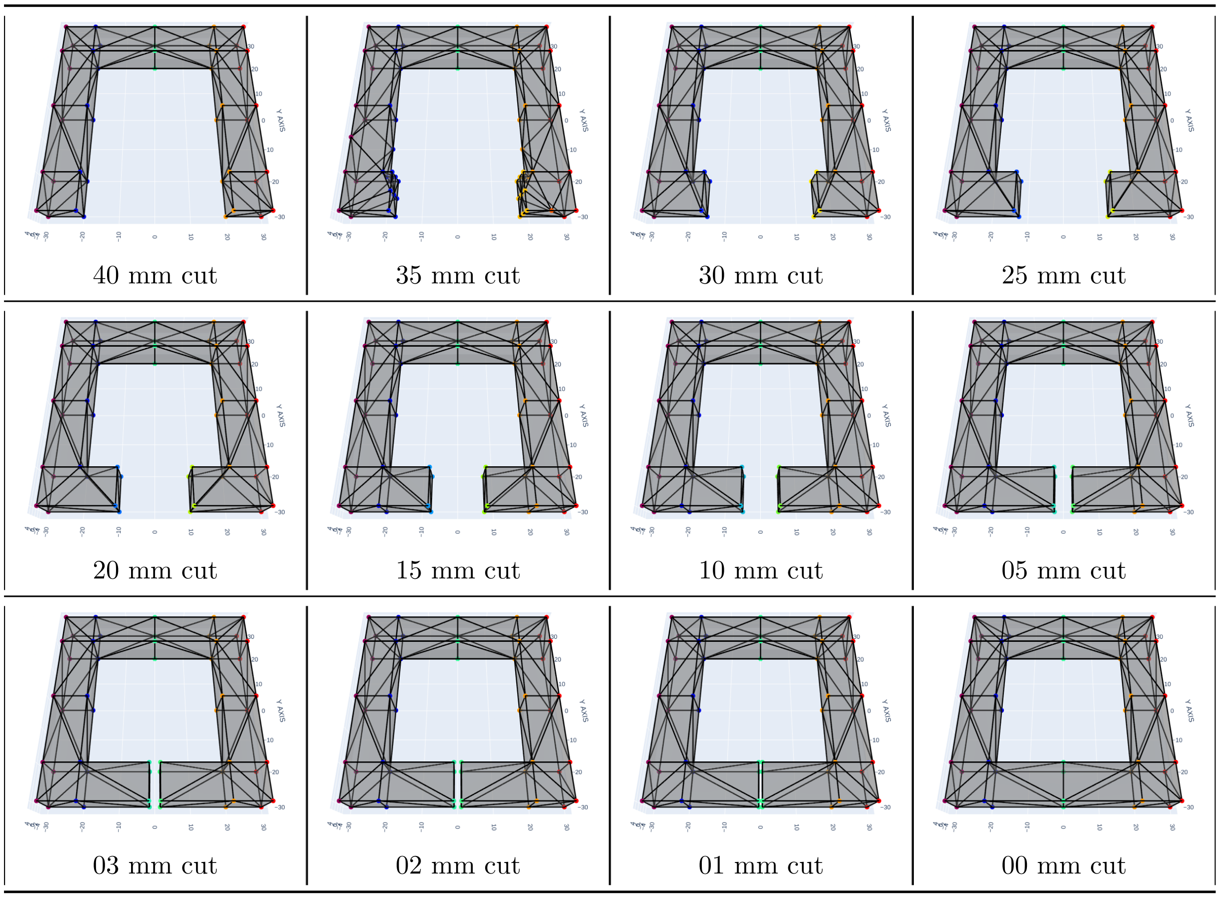


Figure 4.6. MeshPy plots displayed with Plotly of a rectangular prism ring with a cut that decreases to the original shape.

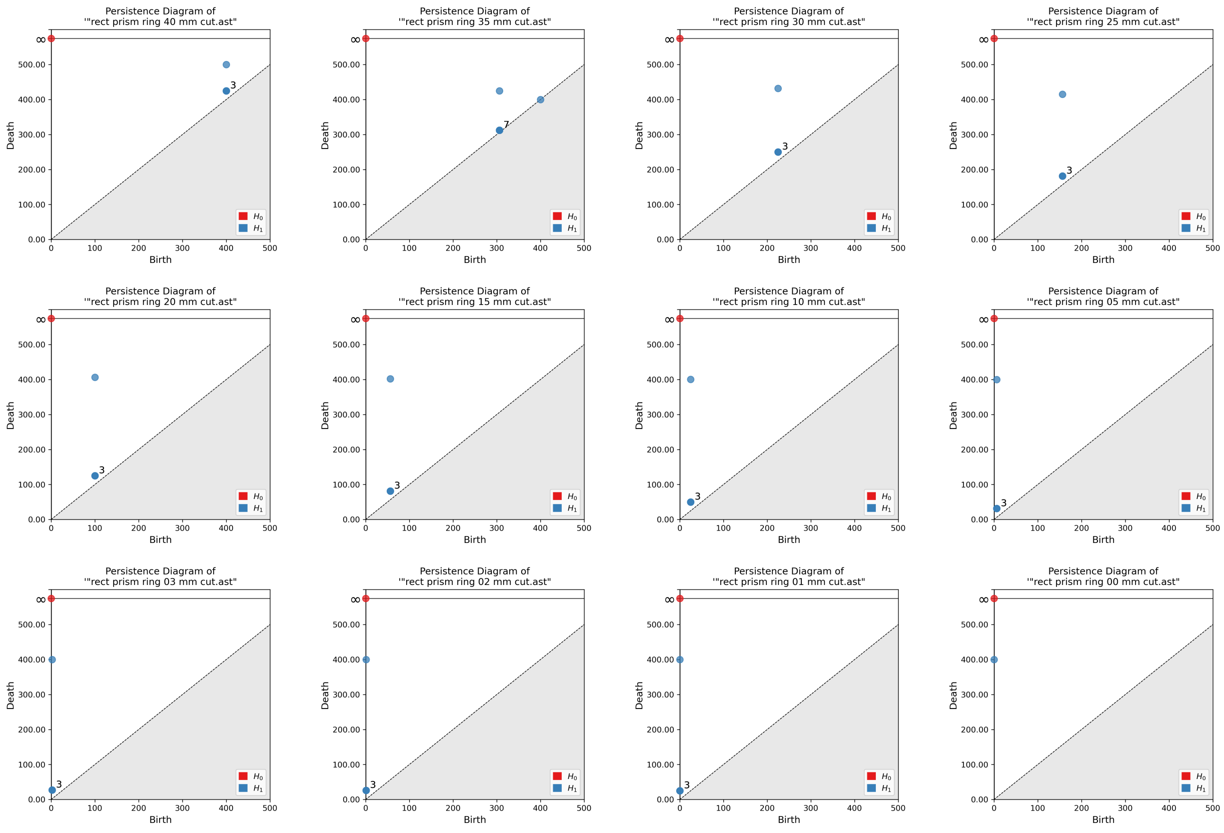


Figure 4.7. Persistence Diagrams of a rectangular prism ring with a cut that decreases to the original shape.

Chapter 5: Discussion

5.1 Two Cubes with Three Pockets Moving Closer

Section 4.1 features a test case of two cubes with three pocket operations on three orthogonal faces. The cubes move closer together until combining to form one object, as shown in Fig 4.2. The three pocket operations on the cube may appear to create 6 holes, but similar to how a non-filled-in tetrahedron has three holes, each cube only has five holes. This can be seen in the figure below of topologically equivalent graphs for a tetrahedron and the cube with three pocket operations (or a non-filled-in cube).

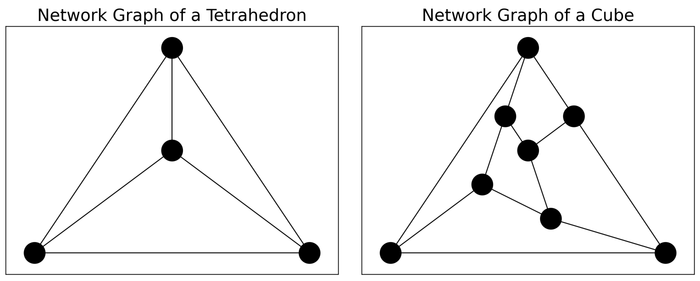


Figure 5.1. Network graphs of a non-filled-in tetrahedron and cube.

The decreasing distance between the objects and the change in topology which occurs when the objects combine are represented in Fig 4.3 as a sequence of persistence diagrams. Initially the two cubes appear as two connected components in H_0 with birth-death points at (0, 625) and (0, ∞). One connected component dies at $r^2 = 625$. This is equivalent to a Euclidean distance of $r = 25$mm which means the open balls of each connected component intersect at the halfway point of the distance, 50mm, between each object. The other connected component was chosen by `Gudhi` to be the first point in the complex, so it does not die and has a death time of $r^2 = \infty$. As the two cubes move closer together, we can convert their distance apart to an H_0 death time by taking the square of their circumradius, which in this case is the shortest distance of two vertices between the two inward faces of each cube: $\mathsf{Death} = (\frac{\mathsf{Dist}}{2})^2$. Conversely, the H_0 death time of a cube can be used to find the distance between two cubes: $\mathsf{Dist} = 2\sqrt{\mathsf{Death}}$. This also explains the value for the death times of H_1 points that represent the holes of the cubes. Each hole is a 5mm square and has a death time of $r^2 = 12.5$. This is equivalent to $r^2 = (\frac{5\sqrt{2}}{2})^2$, where $5\sqrt{2}$ is the distance of the diagonal of the squares. The additional H_1 points on the diagonal are noise and appear when the open balls of vertices between the two inward faces of each cube intersect.

Distance (mm)	Connected Component Death Time
50	625
40	400
30	225
20	100
10	25
0	∞

Table 5.1. Distances between cubes and the corresponding H_0 death times.

In Table 5.1, as the distance shrinks between the two cubes, we can see the death time of the second connected component move closer to 0. When the two cubes combine, there is only one point in H_0 with a death time of ∞ because there is only one connected component. The merger of cubes also shows one of the holes no longer exists as there are 9 H_1 points instead of 10.

In terms of tolerancing between two 3D parts, we can use the base formulas of $d = r^2$ and $r = \sqrt{d}$, where r is euclidean radius and d is death time, along with intuition about how to choose the radii for the formula (set radius equal to half the distance apart in this case) to determine which death time would create the optimal tolerance in fabrication and vice versa. For example, if a fabrication method such as 3D printing had a tolerance of 3mm (this is a realistic tolerance for large-scale printers which print car prototyping parts), we can use $d = (\frac{3}{2})^2$ to find that the H_0 death time of a part feature would need to have a margin of error of ± 2.25 to ensure a good fit with other part features in close proximity to it.

5.2 Cube with Equilateral Triangle Hole

Section 4.2 features a test case of a cube with a pocket of an equilateral triangle through the entire object, creating an equilateral triangle hole. The equilateral triangle hole begins with an edge length of 8mm and decreases in length until the hole dissapears. As expected, there is only one connected component as a point in H_0 with a death time of ∞.

The nature of how the Delauanay triangulation mesh was generated from the original STL mesh affects the resulting persistence diagrams. The differences between hole death times in Table 5.2 depends on the fineness of the mesh generated by `MeshPy`. To understand the reasons why the differences occurred, we will use the following formula to find the circumradius of an equilateral triangle:

$$r = \frac{l}{\sqrt{3}}, \text{ where } l \text{ is the edge length.}$$

We will also use the radius of a triangle's inscribed circle. The inscribed circle of an equilateral triangle exists within the triangle and is tangent to each edge in the triangle. The formula to find the inscribed radius is the following:

$$r = \frac{l}{2\sqrt{3}}, \text{ where } l \text{ is the edge length.}$$

Edge Length (mm)	Inscribed radius (mm), r_i	Circum-radius (mm), r_c	$d = r_i^2$ (Lower Bound)	$d = r_c^2$ (Upper Bound)	Figure 4.5 H_1 Death Time	Original STL H_1 Death Time
8	2.309	4.619	$5.\overline{333}$	$21.\overline{333}$	$5.\overline{333}$	$21.\overline{333}$
7	2.021	4.041	$4.08\overline{3}$	$16.\overline{333}$	4.567	$16.\overline{333}$
6	1.732	3.464	3.000	12.000	9.000	12.000
5	1.443	2.887	$2.08\overline{3}$	$8.\overline{333}$	$8.\overline{333}$	$8.\overline{333}$
4	1.155	2.309	$1.\overline{333}$	$5.\overline{333}$	$5.\overline{333}$	$5.\overline{333}$
3	0.866	1.732	0.750	3.000	$2.\overline{999}$	$2.\overline{999}$
2	0.577	1.155	$0.\overline{333}$	$1.\overline{333}$	$1.\overline{333}$	$1.\overline{333}$
1	0.289	0.577	$0.08\overline{3}$	$0.\overline{333}$	$0.\overline{333}$	$0.\overline{333}$

Table 5.2. Equilateral triangle hole edge lengths and their corresponding death time comparisons between predictions calculated from triangle circumradius and inradius with observed death times from the `MeshPy` Delaunay triangulation mesh and the original STL file mesh.

The death times of an equilateral triangle for a more coarse mesh are expected to be found from using the square of the circumradius, r_c. The expanding balls from each of three vertices in the plane of an equilateral triangle will meet at the intersection of the three perpendicular bisectors of each triangle edge, and the radius of these balls at the intersection will be equal to the the radius of the triangle's circumsphere. Thus, taking the square of r_c will give us the death time of this triangle if it is not "filled-in".

The column of H_1 death times from persistence diagrams in Fig 4.5 shows the death times for edge lengths 5mm through 1mm all match the death times calculated with r_c. We can see the reason for this correlation in the sequence of meshes created with `MeshPy` in Fig 4.4: meshes for edge lengths 5mm through 1mm all consist of only three points to create the triangle.

However, for edge lengths 8mm through 6mm, the `MeshPy` switch `-q` caused the mesh to include many new points in close proximity to each other due to the relatively smaller distances between the vertices of the triangle hole and the edges of the top and bottom faces of the cube. The additional points were also added to the triangle hole edges for these lengths, which caused the Alpha complex to form a hole sooner in r^2 with smaller open metric balls due to the triangles containing more than the minimum three points. This results in the death times of the H_1 points being much smaller than the calculated death time with r_c.

The death times from Fig 4.5 for edge lengths ranging 8mm through 6mm all either match the death times calculated from the inscribed radius r_i (8mm), are close to the death times from r_i (7mm), or are in between the death times from r_i or r_c (6mm and 7mm). The square of the inscribed radius is the lower bound of possible radii to compute the death time of an equilateral triangle. This is becuase it is possible for the smallest distance from points on an equilateral triangle edge to the intersection points of perpendicular bisectors of its three edges to be the distance from the midpoint to that intersection point, or $\frac{l}{2\sqrt{3}}$. Therefore the lower bound death time for an equilateral triangle is $r^2 = (\frac{l}{2\sqrt{3}})^2$.

In most cases, more points need to be added to the mesh in order to make the original STL mesh a Delaunay triangulation. However, as we have seen this can affect the persistence points in H_1. Table 5.2 also shows the death times of triangle holes using the original STL mesh as the input for the Alpha complex. In cases such as these, the desired feature of analysis, the triangle hole, is already a Delaunay triangulation, so increasing the fineness of the mesh through `MeshPy` is not necessary because the death times of the ASCII STL mesh all match (taking into account possible floating-point error) the death times calculated with r_c.

5.3 Rectangular Prism Ring with Cut

Section 4.3 features a rectangular prism ring, meant to be a simplified model of a solid torus, which has a portion cut out of it. The cut starts at 40mm wide and decreases in length until the cut dissappears to the final shape of a ring.

The three H_1 points of holes slightly above the diagonal can be considered noise for the same reason as the noise in Fig 4.3 as we discussed in Section 5.1. However, the H_1 point which is not noise represents the time it takes for the hole to form once the gap of open metric balls in the Alpha complex intersect.

Length of Cut (mm)	Birth Time	Death Time
40	400.0000	500.0000
35	306.2500	425.0000
30	225.0000	431.6406
25	156.2500	415.2588
20	100.0000	406.2500
15	56.2500	401.9775
10	25.0000	400.3906
5	6.2500	400.0244
3	2.2500	400.0032
2	1.0000	400.0006
1	0.2500	400.0001
0	0.0000	400.0000

Table 5.3. Cut lengths and the birth and death times of holes, from Fig 4.7.

The birth times of the holes are formed through the same type of motion as in Section 5.1 where two features move closer together until combining. This causes the birth times to follow the same pattern of $\mathsf{Dist} = 2\sqrt{\mathsf{Death}}$. Aside from the cut length of 30mm, the death times also follow a pattern of decreasing from 500 to converge to 400. The death times converge to 400 because the width of the square hole in the rectangular prism ring is 40mm, so we can calculate that 400 should be the death time by $\mathsf{Death} = (\frac{\mathsf{Dist}}{2})^2$.

While the birth time or death time could be used for choosing a tolerance or clearance value for a part's design, it would be easier to choose these values based on birth time. The death times also correspond to the length of the cuts, but in order to choose a tolerance value closer to 0.5mm or lower, floating-point errors could begin to be a problem with the difference between death time and $r^2 = 400$.

Chapter 6: Conclusion

6.1 Findings

In this paper we outlined a method for analyzing 3D ASCII STL files to determine metrics for when a feature of a part may have a margin of error in its tolerance or clearance which could cause issues in fabrication. We first pass in the STL data to `MeshPy` to generate a Delaunay triangulation of the surface of the object. This is done to ensure features of parts are accurately represented in H_0 and H_1 by the Alpha complex filtered simplices. However, depending on the part feature to be analyzed, meshing can cause irregular death times to occur (see Sec 5.2). We then compare the simplices of the Alpha complex to the simplices of the Delaunay triangulation mesh to assign a birth time of 0 to the Alpha complex simplices which were originally in the Delaunay triangulation mesh. Using these modified persistence point values, we visualize the persistent homology of a part with a persistence diagram. We then repeat these steps for a sequence of variations of altered part feature measurements as separate STL files to determine potential patterns in the birth and death times of the feature.

6.2 Future Work

For computational efficiency, the implementation could be improved by checking if the mesh originally contained within a 3D object file is Delaunay or not upon parsing to determine if it is necessary to create a new mesh with meshpy. Potentially eliminating this step would reduce the overall computational expense of the program.

The variations of the three cases of 3D objects were created manually in FreeCAD, but creating the variations could be automated. This can happen either through using the FreeCAD python library and editing the original FreeCAD .FCStd file or by editing the text of the exported ASCII STL file to manipulate facets and vertices to change part features to simulate different tolerances.

The methods of this paper can be improved upon by computing a measure of topological stability to determine the error from the "truth" of an STL file through analyzing the different persistence diagrams of the different versions of the file. The "truth" value of the 3D object would be the persistence vector of the preselected object and the error would be computed through the differences in the persistence diagrams of the variations.

This paper only analyzes connected components and holes. When the `MeshPy` meshes of 3D objects were created, it only triangulated the outer surfaces of objects. So when an object with a void was given to the program, it created a mesh of only the outer surface and inner surface of the void, but it did not tetrahedralize the space between the two surfaces. This led to voids having a birth time not equal to $r^2 = 0$ after the step of modifying filtration values. The methods of this paper can also be improved through finding the correct meshing algorithm to tetrahedralize the space between surfaces in an object containing a void, as well as fully enclosed objects.

Bibliography

[1] Keri L. Anderson, Jeffrey S. Anderson, Sourabh Palande, and Bei Wang. Topological data analysis of functional mri connectivity in time and space domains. In Guorong Wu, Islem Rekik, Markus D. Schirmer, Ai Wern Chung, and Brent Munsell, editors, *Connectomics in NeuroImaging*, pages 67–77, Cham, 2018. Springer International Publishing. `10.1007/978-3-030-00755-3_8`.

[2] Boris Delaunay et al. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7(793-800):1–2, 1934.

[3] Riccardo Fellegara, Federico Iuricich, Leila De Floriani, and Ulderico Fugacci. Efficient homology-preserving simplification of high-dimensional simplicial shapes. *Computer Graphics Forum*, 39(1):244–259, Aug 2019.

[4] Michelle Feng and Mason A. Porter. Persistent homology of geospatial data: A case study with voting. *SIAM Review*, 63(1):67–99, 2021. `https://doi.org/10.1137/19M1241519`.

[5] K Ruben Gabriel and Robert R Sokal. A new statistical approach to geographic variation analysis. *Systematic zoology*, 18(3):259–278, 1969.

[6] Benoıt Hudson, Gary L Miller, Steve Y Oudot, and Donald R Sheehy. Mesh enhanced persistent homology. *Computer Science Department*, 1122, 2009.

[7] Jocelyn Ilagor. *Visualizing Arctic Ice Data with Optimal Transport.* MA Thesis, The University of North Carolina at Greensboro, 2023.

[8] T. Maquart, Y. Wenfeng, T. Elguedj, A. Gravouil, and M. Rochette. 3D volumetric isotopological meshing for finite element and isogeometric based reduced order modeling. *Computer Methods in Applied Mechanics and Engineering*, 362:112809, 2020. `https://www.sciencedirect.com/science/article/pii/S0045782519307017`.

[9] Tom Needham. Introduction to applied algebraic topology, 2019. `https://sites.google.com/site/tneedhammath/research`.

[10] Christopher Oballe, David Boothe, Piotr J Franaszczuk, and Vasileios Maroulas. Tofu: Topology functional units for deep learning. *Foundations of Data Science*, 2021.

[11] Peter Palfrader and Martin Held. Computing mitered offset curves based on straight skeletons. *Computer-Aided Design and Applications*, 12(4):414–424, 2015.

[12] M. Pellikka, S. Suuriniemi, L. Kettunen, and C. Geuzaine. Homology and cohomology computation in finite element modeling. *SIAM Journal on Scientific Computing*, 35(5):B1195–B1214, 2013. `https://doi.org/10.1137/130906556`.

[13] Hang Si. *TetGen, version 1.5, user's Manual.* Weierstrass Institute, 2013. `https://wias-berlin.de/software/tetgen/1.5/doc/manual/index.html`.

[14] M. Szilvśi-Nagy and Gy. Mátyási. Analysis of stl files. *Mathematical and Computer Modelling*, 38(7):945–960, 2003. `https://www.sciencedirect.com/science/article/pii/S0895717703900793`.

[15] Songzhu Zheng, Yikai Zhang, Hubert Wagner, Mayank Goswami, and Chao Chen. Topological detection of trojaned neural networks. *Advances in Neural Information Processing Systems*, 34:17258–17272, 2021.

[16] Afra Zomorodian and Gunnar Carlsson. Computing persistent homology. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 347–356, 2004.