TOPOLOGICAL AND FEATURE BASED IDENTIFICATION OF HOLE BOUNDARIES IN POINT CLOUD DATA AND DIFFERENTIATION BETWEEN SURFACE AND PHYSICAL HOLES

by

AAQIF MUHTASIM

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2018

Acknowledgements

First and foremost I would like to convey my most sincere of gratitude to Dr. Manfred Huber for being my supervising professor and providing support and direction to my research whenever I required. His vast knowledge and insight have guided me quite a few times when I was at a loss about which direction to navigate my research towards.

Secondly, I would like to thank Dr. Beksi for being on my committee and more importantly for sharing his research with me. His assistance has provided a very necessary acceleration to my research and have played a major role in its conclusion.

I would also like to thank Dr. Kamangar for agreeing to be on my committee.

I would also like to thank Sandeep Chahal and Sreesha Kashyap my friends and lab mates who have provided moral support during times of despair.

I would like to thank my parents for supporting me in my endeavors as I journeyed through both undergraduate and graduate school. My academic career would have been drastically different without it.

Finally, I would like to thank Dr. Roger Walker, my first supervisor. He has provided a lot of support at the start of my graduate program and it is unfortunate that he is absent during its completion.

Nov 19, 2018

Abstract

TOPOLOGICAL AND FEATURE BASED IDENTIFICATION OF HOLE BOUNDARIES IN POINT CLOUD DATA AND DIFFERENTIATION BETWEEN SURFACE AND PHYSICAL HOLES

Aaqif Muhtasim, MS

The University of Texas at Arlington, 2018

Supervising Professor: Manfred Huber

With the advent of autonomous agents becoming prominent in everyday lives, the importance of processing the surroundings into understandable features becomes more and more important. 3D point clouds play a major role in the perception of such agents and thus having the ability to correctly decipher features from point clouds is crucial to the planning of actions that the agent would need to undertake.

This thesis analyzes holes found in point clouds. Based on two approaches that center around topological data analysis and local point set features respectively. It studies how each of the methods works and how a combination of the two can be used to ascertain important information that may not have been obtainable from just one of them. Moreover, it studies how distinctions between different types of holes in point clouds can be made. The thesis contributes in two ways in the feature extraction from point cloud holes.

The first contribution is the constriction of the minimal 1-cycle generated by the addition of edges to the minimum spanning graph generated. These edges are detected using local surface geometry for the points and allow elimination of vertices from the hole boundary thus providing a tighter hole boundary.

The second contribution is the classification of the type of hole whose boundary has been detected. This involves calculating a normal to the surface approximated by the boundary and detecting a chain of vertices on the boundary whose surface normal are either orthogonal or parallel to the normal of the boundary points.

This thesis approaches the abstract notion of a hole and tries to provide a boundary in order to allow for planning of actions that might involve it, such as determination of further sensing actions or determination of interaction points for object manipulation. We have provided algorithms that calculate the necessary features and have provided results that show their effectiveness in real-world scenarios.

Table of Contents

List of Illustrations

List of Algorithms

Chapter 1

INTRODUCTION

Modern society is inundated with technologies that utilize sensors in order to gather information about the world around them. These sensors usually gather data about the intensity of light, noise, temperature etc. in order to get a sense of their environment. One of the common ways to represent and store sensory information about the environment is in a spatial form in terms of Cartesian coordinates within a 3-dimensional (3D) map. As the number of technologies dependent on a mapping of their 3D environment grows so does the number of sensors that are dedicated to the 3D representation of the world. This influx of sensors bringing with them a large amount of 3D data thus requires more algorithms that can process such data and ascertain meaningful features from them in order to make the data useful.

Point Cloud Data

Point clouds are a common way to represent 3D data for further processing in robot applications. Point cloud data is data collected by sensors such as LIDAR (Light Detection and Ranging), stereo, structured light, and ToF (Time-of-Flight) cameras. This data by providing 3D coordinates of points on a surface allows for the accurate mapping of shapes of objects. Depending on the sensor source, 3D points in a point cloud can also contain additional information such as signal intensity, color or surface normal at the point. However, many sensors do not naturally provide this information, leaving frequently only the actual location of the surface points. Point Cloud Data generally contains no point connectivity information and is usually unorganized, as it can be merged from multiple sensors as well as from multiple measurements taken by these sensors, and thus lacks the easy detection of the neighborhood. This is in contrast to 2D images or single depth

images from 3D cameras where the layout of the imager provides a natural structure for the pixel positions. Sample 2D snapshots of point cloud data of various objects are provided in

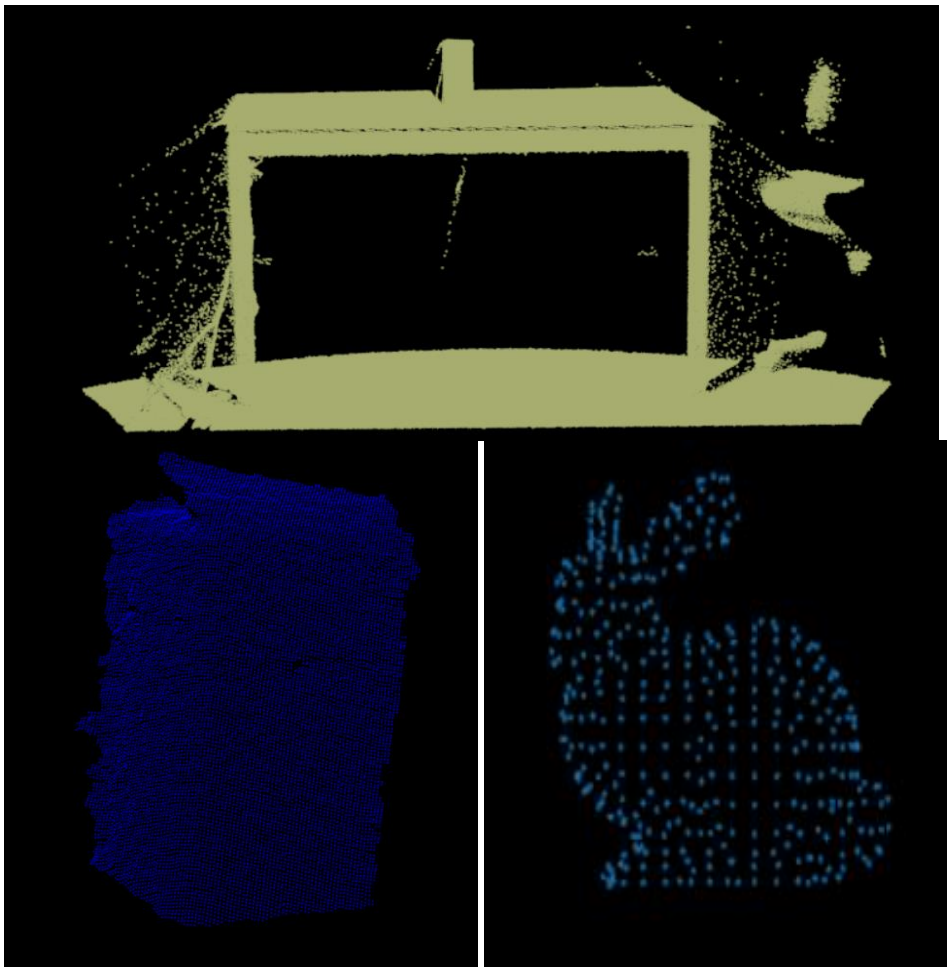


Figure 1-1 Point Cloud Examples

Point Cloud Features

A point cloud with n-points will contain at a minimum nx3 numbers with each row of 3 numbers indicating a point in the 3D coordinate system. As indicated above, points in a point cloud can contain additional information (e.g. color) depending on the sensor that

generated the data but these additional features can be heterogeneous across data points and absent, e.g. in the case of many LIDAR sensors. Extracting meaning through combination and manipulation of these numbers in order to allow for other systems to use the data is known as feature extraction. For point cloud data this may come from segmenting a given point cloud into different objects in order to assign the resulting objects to their respective labels. One significant issue when extracting features or objects from point clouds is that while the point cloud represents points on the surface of the object, the points generally do not cover the entire surface either due to occlusions, reflections or surface properties that prevented the sensor from observing a part of the surface, or because no sensor has been in a position from which particular parts of the object surface would have been visible. This leads to holes in the point cloud data which might either correspond to holes present in the object surface or to the absence of information at a location. This significantly complicates the identification of surfaces and the segmentation problem, making the detection of these itself a feature detection problem in point clouds.

Hole Definition and Boundary Accuracy

A hole in the physical sense is defined as “a hollow place in a solid body or surface”. The problem arises when transferring this definition to point cloud data and trying to identify a hole boundary feature. A common representation for a hole boundary feature is a chain of points that would indicate the boundary of said hole. This, however, can become problematic in point cloud data. For a hole in a truly 2D surface (i.e. a surface – and thus a hole - that has no depth), this issue is relatively simple since the opening is on a 2D surface and is thus consistent from multiple perspectives. For a hole in a 3D object, the fact that the hole has depth and is surrounded by a 2D surface makes the problem much more prominent since the boundary would appear different from different

perspectives and thus there would be multiple correct answers. Due to this fact, no easy accuracy measure for surface holes has been developed and thus the quality of the feature detector results is generally judged solely by visual inspection.

Approach

The aspect of hole features and their classification in terms of physical, 3-dimensional holes, 2D surface holes, or data holes resulting from missing observations, has not been a well-defined topic in algorithms that process point cloud data. Even though detection of surface holes has been an active area of research, accurate identification of the presence and boundary of physical holes surrounded by a surface has still not been sufficiently explored.

The main approach for this thesis extends on and combines previous approaches for hole identification in 3D point clouds and revolves around applying the topological calculation of the minimal 1-cycle with curated parameters to get a base boundary for the hole. To improve the compactness of the resulting boundary feature, the final boundary is created by adding edges based on local geometry to other points inside the initial boundary. Based on the resulting feature, a type classification is defined by finding a chain of points along the boundary whose surface normal are consistently orthogonal or parallel to the surface normal of the hole and have a degree of parallelism with each other.

Contributions

The contributions made by this thesis in understanding the features of a hole are:

- Providing an algorithm that calculates the minimal hole boundary and thus produces a series of connected vertices that mark the inside of a hole.

- Classifying the type of hole whose boundary has been detected thus allowing for better characterization of the point cloud object.

Thesis Organization

In the remainder of the Thesis, Chapter 2 first introduces the fundamental mathematical background for hole identification before discussing the related work in the area of hole identification in 3D point cloud data. In Chapter 3 we talk about two previously explored approaches to hole boundary detection and try to combine them to achieve better boundary extraction as well as provide some metrics which might be used to evaluate the quality of a given hole boundary. Chapter 4 introduces a surface normal based approach to classify between surface and physical holes. We conclude with mentions of future work that can be performed in Chapter 5.

Chapter 2

BACKGROUND

Topological and Geometric Properties in Point Clouds

Since points in a point cloud do not naturally form surfaces and do not contain connectivity information, one essential task when analyzing the data, and thus when trying to identify hole features, is to establish connectivity and surface characteristics within the data. For this, topological and geometric properties are generally used.

Simplex and Simplicial Complex

A k-simplex σ where $0 \leq k \leq n$ is defined as the complex hull of a set of k+1 independent points in $\mathbb{R}^n$. The convex hull is the solid polyhedron determined by the k+1 points. Some examples of Simplices in different dimensions are the vertex, the edge, the triangle, and the tetrahedron as shown in Figure 2-1.

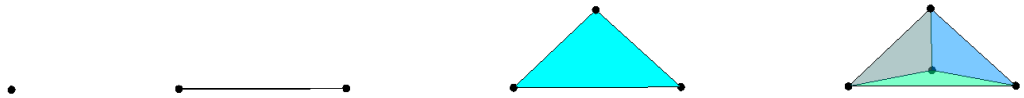


Figure 2-1 Simplices in Different Dimensions, From 0 (left) to 3 (right)

A simplicial complex is a collection of simplexes such that if σ and σ' are Simplices in a simplicial complex K, then $\sigma \cap \sigma'$ is either null or a face τ such that $\tau \in K$, $\tau \in \sigma'$ and $\tau \in \sigma$. A face of a k-simplex is defined as the set of k-1 simplices obtained by removing each of the points in the k-simplex one at a time. An example can be seen by removing each of the points from the tetrahedron (3-dimensional simplex) in Figure 2-1 to obtain a set of 4-face triangles. An example of a simplicial complex is seen in Figure 2-2.

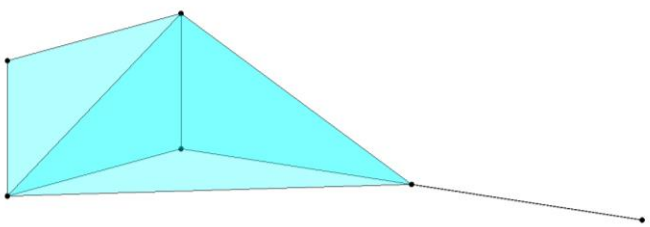


Figure 2-2 Simplicial Complex

Vietoris-Rips Complex

A Vietoris-Rips Complex M in $\mathbb{R}^n$ is an abstract simplicial complex such that for each simplex $\sigma \in M$ and for all vertices $v_0,\ldots,v_n \in \sigma$ the distance between each pairwise vertex in σ is less than some predefined radius r. Figure 2-3 shows a sample Vietoris-Rips Complex where the radius for each point is shown in blue.

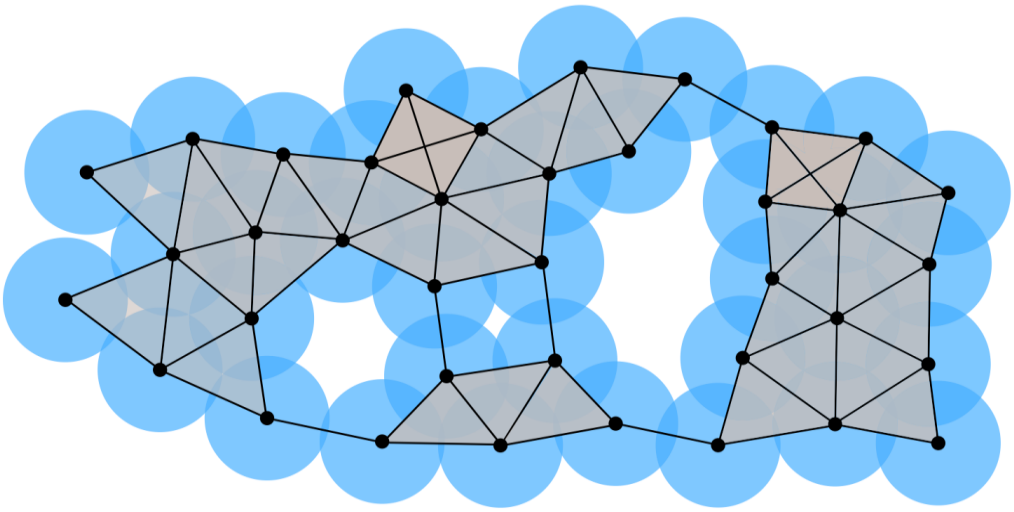


Figure 2-3 Vietoris-Rips Complex [1]

Chains, Cycles, and Boundaries

A k-chain is a subset of k-simplices in K, where K is a simplicial complex in $\mathbb{R}^n$. The sum of two chains a,b is defined by the symmetric difference between the two,

$$a + b = (a \cup b) - (a \cap b)$$

and is commutative. C_k is the group of all k-chains together with addition. For a complex in $\mathbb{R}^3$ the only non-trivial groups are for $0 \leq k \leq 3$.

The boundary $\partial k(\sigma)$ of a k-simplex σ is the set of its (k-1)-faces and is a (k-1)-chain. The boundary of a k-chain is found by summing the boundaries of its simplices,

$\partial k(c) = \sum_{\sigma \in c} \partial k(c)$. The boundary operator is a homomorphism such that ∂k: $C_k \rightarrow C_{k-1}$ and the collection of the operators on the chain groups form a chain complex.

$$\xrightarrow{\partial 3} \quad \xrightarrow{\partial 2} \quad \xrightarrow{\partial 1}$$
$$\cdots \rightarrow \emptyset \rightarrow C3 \rightarrow C2 \rightarrow C1 \rightarrow C0 \rightarrow \emptyset \cdots.$$

The kernel of ∂k is a subgroup of C_k and is defined as the set of k-chains whose boundaries are empty under ∂k and are also known as the k-cycles. The image of $\partial(k+1)$ is also a subgroup of C_k and is defined as the set of k-chains which are boundaries of (k+1)-chains and are also known as k-boundaries.

$$\ker \partial k = \{c \in Ck \mid \partial k(c) = \emptyset\}$$
$$\text{img } \partial k = \{d \in Ck-1 \mid \exists c \in Ck : d = \partial k(c)\}.$$

C_k is formed by the set of k-cycles Z_k and the set of k-boundaries B_k under addition. Since an essential property of the boundary operator is that the boundary of a boundary is empty, $\partial k1 \circ \partial k(c) = \emptyset$, the groups are thus nested $B_k \subset Z_k \subset C_k$. $Z_0 = C_0$ since the boundary of a vertex is empty and thus every 0-chain is also a 0-cycle. $Z_3 = B_3 = \{\emptyset\}$ is a consequence of the complex being in $\mathbb{R}^3$ and thus having no non-empty 3-cycles or 3-boundaries. Figure 2-4 shows the relations between Chains, Cycles, and Boundaries.

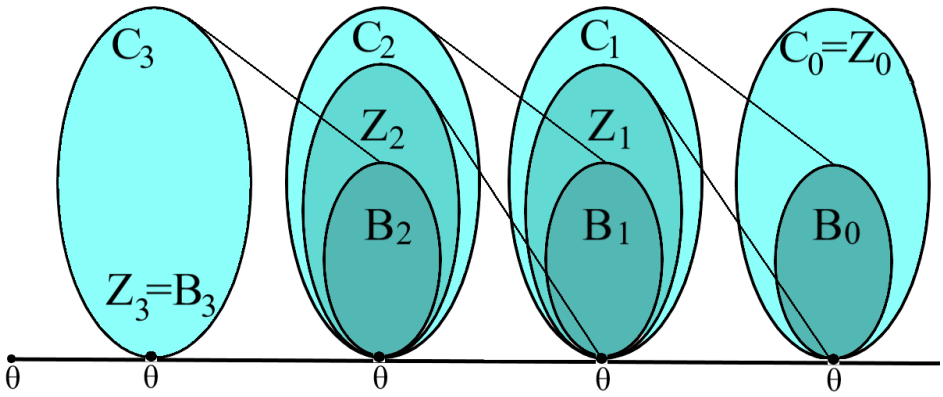


Figure 2-4 Chains, Cycles, and Boundary groups
with their images under the boundary operator

Homology Groups and Betti Numbers

The kth homology group H_k is defined as the quotient of the kth cycle group by the kth boundary group: $H_k = Z_k/B_k$. Homology Groups $H_k(X)$ will have their dimensions correspond to the number of k-dimensional holes in X where X is a simplicial complex.

$H_k(X)$ is a quotient vector space with its dimension equal to its number of generators and whose elements are a linear combination of its generators. Assuming a group with generators {a,b}, a typical element in the group would be $\alpha a+\beta b$ and the dimension of the group would be 2.

This definition allows for all bases to have the same size which corresponds to the rank of the group. Since taking the symmetric differences is similar to adding modulo 2, the size of a group is 2 raised to the power of its rank. An example of adding two 1-cycles is shown in Figure 2-5.

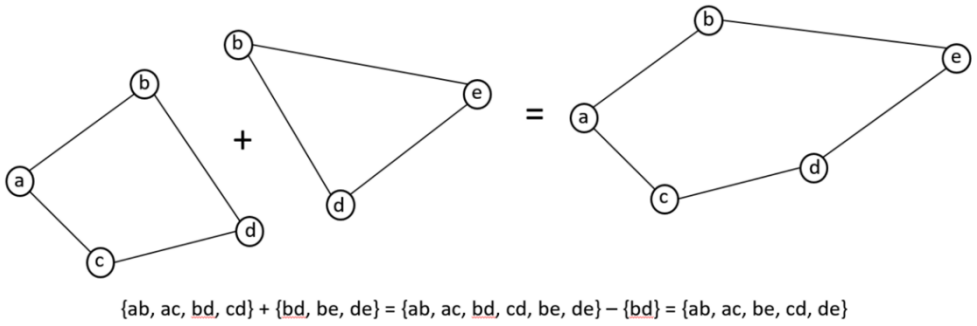


Figure 2-5 1-Cycle Addition

The kth Betti number of X is calculated as the rank of the kth homology group β_k = rank H_k = rank Z_k – rank B_k.

Complexes in $\mathbb{R}^3$ can only have non-trivial Betti numbers for $0 \leq k \leq 2$. B_0 measures the number of connected components in X, β_1 measures the number of holes (tunnels) in X, and β_2 measure the number of voids in X.

Filtration

A filter is defined as an ordering of simplices such that each prefix contains the simplices of a subcomplex. The corresponding filtration is defined by taking successively prefixes. Filtration can be described as the evolution of a complex under the element of growth.

Best Fit Plane and Surface Normal

Given points on a surface, it is sometimes useful to have a model of the underlying surface. Given a point and its neighborhood we can use the data to fit a plane and thus allow us to model the underlying surface. The method used in this thesis uses eigenvalue

decomposition to provide a point and a surface normal when provided with a matrix of point coordinates. Figure 2-6 shows an example of the surface normal representation for a point cloud.

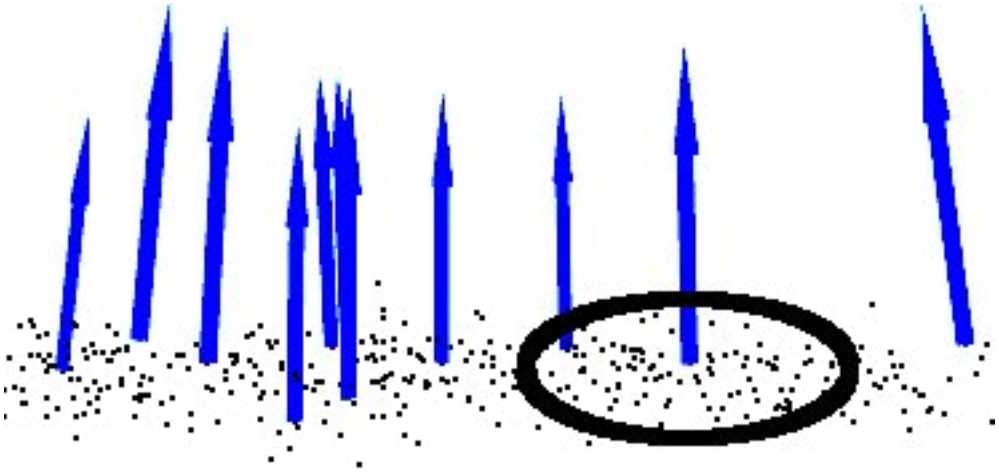


Figure 2-6 Point Surface Normal

Minimum Spanning Tree

A minimum spanning tree is defined by a subset of a weighted, connected graph that connects all of the vertices without cycles such that the sum of edge weights is minimized. A minimum spanning tree also defines a connected component. There are multiple algorithms that can be used to calculate the minimum spanning tree of a graph with some of the more commonly used ones being Prim's algorithm and Kruskal's algorithm. An example of a minimum spanning tree is shown in Figure 2-7.

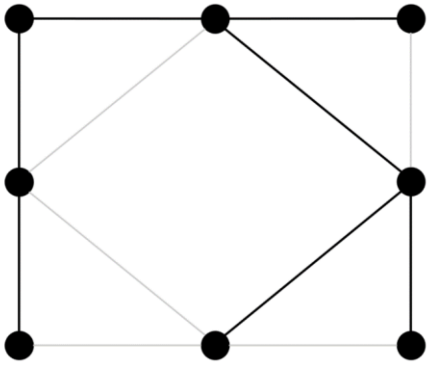


Figure 2-7 Minimum Spanning Tree Example

Ray Casting

A minimal 1-cycle requires determining whether any vertices are present inside the provided boundary. The ray casting algorithm allows us to detect whether a point is inside the boundary by first modeling the boundary as a polygon and then by finding out how many times a ray, starting from the point and going in any fixed direction, intersects the edges of the polygon. The parity of the intersections determines whether the point is inside or outside the polygon (even = inside). A sample ray passing through a polygon is shown in Figure 2-8.

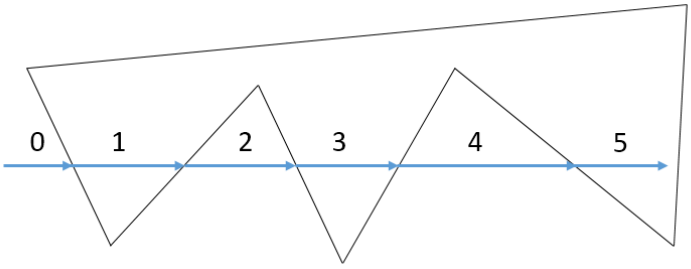


Figure 2-8 Raycasting Example

Chapter 3

HOLE BOUNDARY EXTRACTION

Approach

As explained before, defining the boundary of a hole in point clouds remains as one of the more challenging aspects of point cloud feature detection. This chapter looks first at two existing approaches that calculate the boundary of a hole. The two techniques do so in very different ways with the first [1] taking a topological approach while the second [3] takes a local point feature approach to determine the likelihood of boundary points and then combines them. Considering the different strengths and weaknesses of the methods, this thesis then proposes a way to combine aspects of the two techniques into a method that can achieve tighter hole boundary features. The combination allows for the generation of multiple boundaries each with its benefits and drawbacks can yield a decrease in the time taken to compute one of the boundaries. Finally, we discuss issues in the evaluation of hole boundary features and try to provide heuristics as an initial attempt to define the accuracy of a hole boundary.

.

Related Work

Hole boundary detection is an area of research that has been sparsely explored. NGUYEN et al. [6] used organized point clouds to detect the k-ring neighborhood for each point using an 8-connected neighborhood and extracts a boundary using region growing. Bendels et al. [3] calculates features of points and combines the features to calculate the probability of points being on the boundary. The boundary is then extracted using a graph and Best-First Search (BFS).

A topological approach to hole boundary extraction is used in [1] by Beksi to create a simplicial complex of the point cloud which is then used to calculate a minimum spanning

tree and an edge that represents the hole boundary. The boundary is then extracted by finding the shortest path between the vertices of the edge detected.

Wang et al. [8] uses a triangular mesh created from the input point cloud to find a boundary edge which is an edge belonging to a single triangle. The boundary vertices are found by tracking these boundary edges. A heuristic based on the area of the connected triangles and the number of points is then used to filter out vertices to get a subset of the vertices as boundary vertices.

Mineo et al. [9] uses a two-step method to identify boundary points. If for a query point there exists a circle that passes through it and two of its neighbors such that the radius of the circle is greater than or equal to B, where B is an approximation of the local resolution, and that a sphere with the same radius and center does not contain any other points in the neighborhood of the query point, then we mark the query point as a boundary point. A secondary filtration is done by projecting the query point and its neighbors onto a tangent plane and tries to find a path through all the neighbors such that the path does not encircle the query point. If such a path is found then the query point is identified as a boundary point.

Topological Hole Boundary Extraction

Beksi [1] introduced a method to determine hole boundaries using point cloud topology. Topology can be used to calculate features such as Betti numbers which indicate the number of holes in a simplicial complex with the k-th Betti number representing the number of k-dimensional holes in the complex. A hole boundary can thus be calculated by using the 1-simplex representing the 1-hole in a complex. a hole boundary can be calculated.

Vietoris-Rips Complex Calculation

A Vietoris-Rips complex is calculated by first sorting the input points according to their x, y, and z coordinates respectively. This allows for an initial ordering of the vertices to create the initial 0-simplex ordering for the filter. This sorting carries over to higher dimension simplex ordering since the edges (1-simplices) are processed according to the ordering of the points and the triangles (2-simplices) are processed according to the ordering of the edges. Each simplex thus contains an index specifying their ordering with lower dimensional simplices being ordered earlier.

A KD-tree is created using the input points in order to calculate the neighborhood for each point. A fixed-radius neighborhood which collects the set of neighbors around a point contained in a sphere of radius ε, is used. Each point only accepts neighbors with higher index values from the set created. Figure 3-1 shows the radius used for a point (blue).

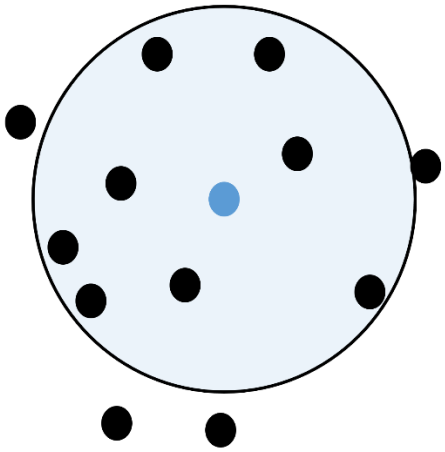


Figure 3-1 Fixed-radius Nearest Neighbors

The calculation of the 0-simplices (points) is trivial since all of the points in the point cloud are 0-simplices. The 1-simplices (edges) are calculated by first creating a list of

edges associated with each point. This is done by adding edges from a point to all of its neighbors. The 2-simplices (triangles) are calculated by going through each pair of points for an edge and finding out the intersection of points in their neighborhood. Each intersection with a higher index point indicates a 2-simplex. This can be further explained by saying that for edge {u, v}, if w is in the neighborhood of u and w is in the neighborhood of v, {u, v, w} is a triangle. Figure 3-2 shows the set of triangles originating from a point.

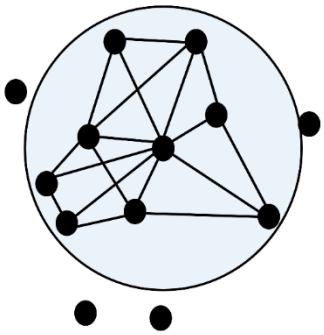


Figure 3-2 A point with all of associated triangles

Betti Number Calculation

The calculation of Betti numbers is an essential part of the extraction of the hole boundary. Betti-1 indicates the number of detected holes in the point cloud based on the input radius. This number also indicates the positive edges that do not form part of a boundary of a triangle and thus gives us the basis to start our search for the boundary of the hole. Since Betti-1 requires the edges to be labeled as either positive or negative, we need to run the algorithm to compute Betti-0 beforehand which labels the edges necessary for Betti-1 calculation.

Betti number B_k is the number of k-cycles that are not part of the boundary of (k+1)-simplex. We call a (k+1)-simplex σ positive if it belongs to a (k+1)-cycle and negative if it

destroys a k-cycle. For Betti-0 this is trivial since every point belongs to a 0-cycle and thus we just have to determine the number of negative edges. An edge belongs to a 1-cycle if both of its endpoints belong to the same component. Thus B_k is defined as:

$$B_k = pos_k - neg_{k+1}$$

Where pos_k = number of positive k-simplices and neg_{k+1} = number of negative (k+1)-simplices.

Betti – 0

The calculation for Betti-0 requires the use of a disjoint-set dataset where the elements are the points in the complex. Each element starts off by being its own parent. The set of edges are then iterated over and for each edge {u, v} the parents of its endpoints are found using the find operation and joined if they are different using the union operation.

The find operation recursively looks at the parent of the given element until an element is detected who is its own parent.

The union operation first finds the element with the higher index according to the initial sorting of the points and selects it as the element to be added if it is not already a part of another set, that is it has a death index of 0. This element is then added to the set by setting its death index to the current edge being processed. Assuming the youngest element is v and the edge connects u to v, the parents are then labeled as u_root and v_root. If v_root is older (has a lower index) we add u_root as the parent of v_root and vice versa.

Betti – 1

The basis for Betti-1 calculation is similar to Betti-0 in the sense that it involves finding the positive edge that is destroyed by a negative triangle. The algorithm thus goes

through each triangle in the complex and finds λ representing the set of positive edges belonging to that triangle.

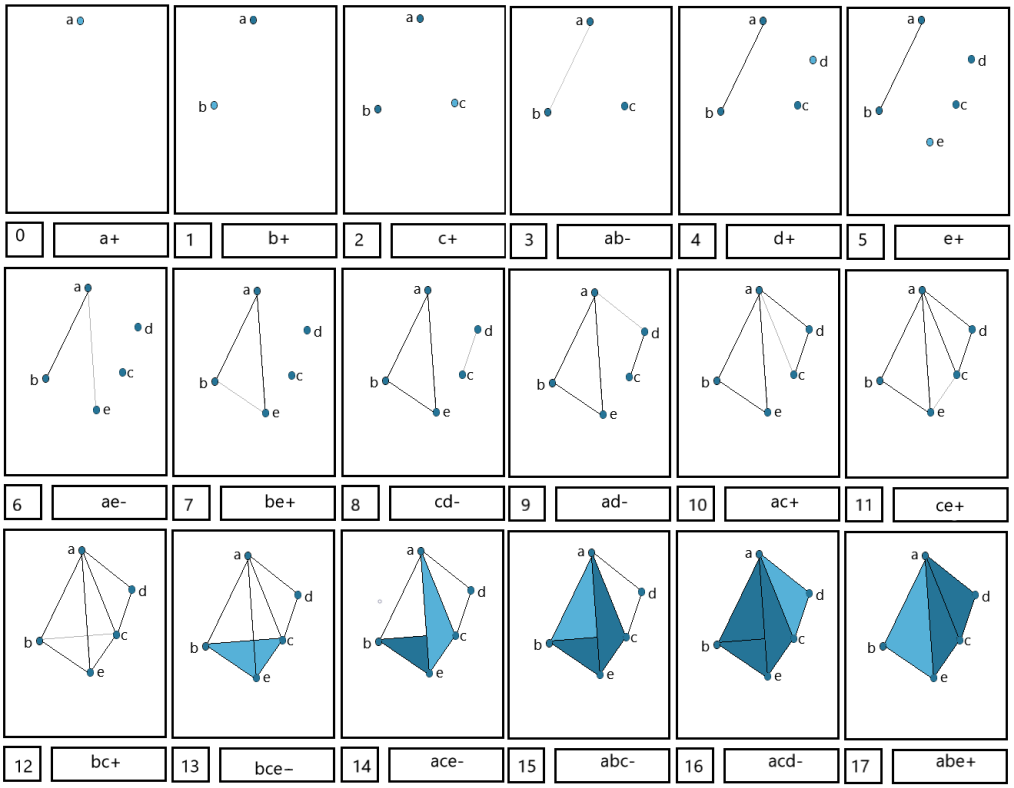


Figure 3-3 Filtration Example

A hash table T is used where T[i] is the index of the i^{th} simplex in the filter and stores the λ^i, which for an edge is the set of positive edges belonging to the triangle that destroyed it. The hash table for the filtration in Figure 3-3 is shown in Figure 3-4 where ∞ indicates a positive k-simplex with no negative (k+1)-simplex that destroys it.

After arriving at a negative simplex σ^j at index j in the hash table the algorithm will create λ and find index i, where i = max(λ) which is the youngest member of λ. If T[i] is empty we store λ and j at T[i]. Else, T[i] already contains a set λ^i that represents a permanently stored 1-cycle. We add λ and λ^i, as defined previously, to get a new λ that produces a k-cycle homologous to the old one and repeat the search with the new λ.

Following the filtration example shown above, when we arrive at j=13 we get λ = {be, ce, bc} and i = max(λ) = 12. We find T[13] empty and store λ and j there. Following the same method for j=14 we get a λ = {ac, ce} stored at T[11]. At j=15 we get λ= {ac, bc} and i = max(λ) = 12. However, we already have λ^i = {be, ce, bc} already stored at T[12]. We

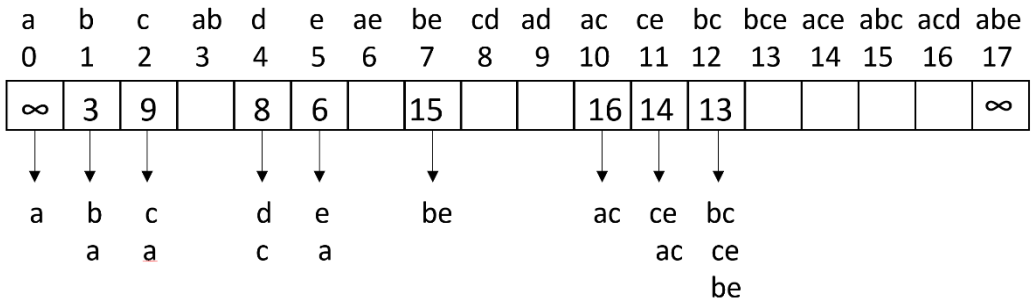


Figure 3-4 HashTable for the Filtration

thus take the symmetric difference between λ and λ^i resulting in the new λ = {be, ac, ce} and get a new i = max(λ) = 11. Again we find T[11] to be occupied and get a new λ = {be} and i = 7. Finally we find T[7] to be unoccupied and store j and λ there.

Minimum Spanning Tree

Since Betti-0 uses the union operation to combine components together, a minimum spanning tree is generated in the process. However, since the union operation does not take into account distances between the vertices when combining them the tree generated does not represent the accurate connectivity between the points. An altered version of the union operation is used such that the parent of v_root is changed from u_root to u and vice versa. This ensures we only add edges to the tree that maintain the radius restriction initially provided to the algorithm. For special cases where the parent of a node is younger than the node, we add a special attribute to the point called parent_conn.

parent_conn is used to define an edge that represents a different pathway for an element to be connected to a set other than the parent attribute in the union operation. The parent_conn of a point is set when connecting two sets of more than one element to each other and is used to represent the connectivity information more accurately whereas the general union operation will sometime create edges longer than the input parameter. The parent_conn of v_root is set to the edge number whenever the parent is set if v_root is older than point u and vice versa. The difference in the minimum spanning trees generated is shown in Figure 3-5.

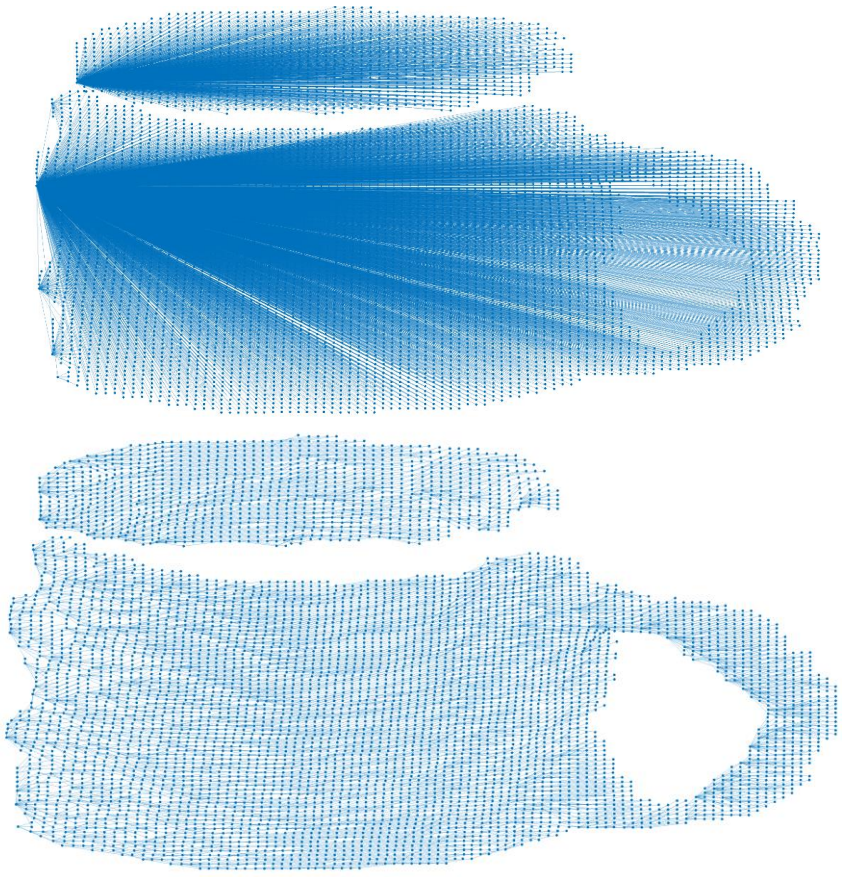


Figure 3-5 Spanning Tree Resulting from Standard Betti-0 Union Operation (Top) and Spanning Tree from Modified Union Operation (Bottom)

Minimal 1-Cycle Extraction

Betti-1 identifies the edge whose addition creates a cycle in the minimum spanning tree which is homeomorphic to the hole boundary. By finding a path between the two vertices of the edge we can identify an initial boundary. This initial boundary is then used to approximate a tangent plane upon which all of the points are projected. The initial boundary is then used to approximate a polygon and the set of points that fall inside the polygon are discovered. Any unpaired positive edge associated with the points on the boundary or interior of the polygon is added to the graph and the shortest path between the edges are found again to calculate a Minimal 1-Cycle surrounding the hole. Figure 3-7 shows an example of the constriction of the hole boundary and Figure 3-6 shows its effects on a physical hole boundary.

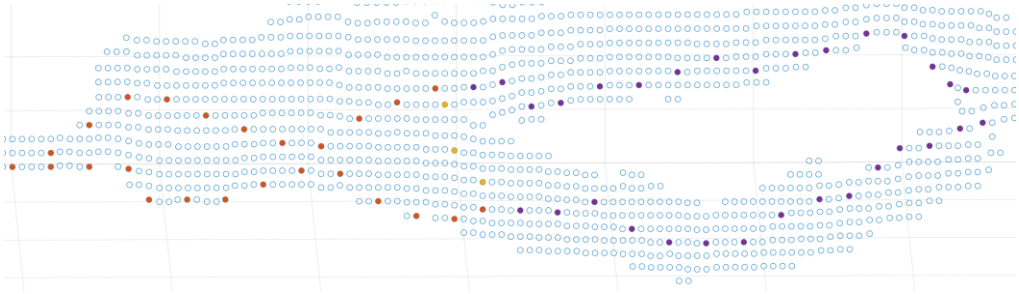


Figure 3-7 Initial Boundary Points (Orange). Minimal 1-Cycle boundary (Yellow). Their intersection (Purple).

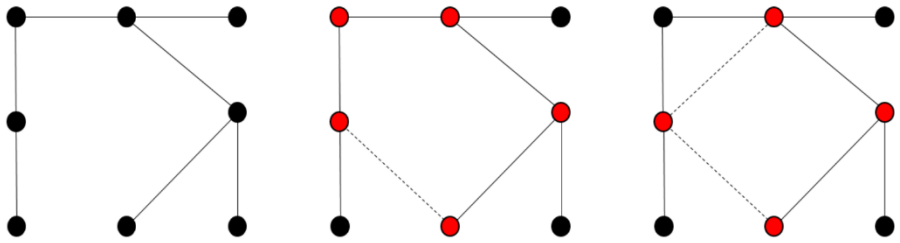


Figure 3-6 Minimal 1-cycle example

Local Feature-based Boundary Extraction

Another method to determine hole boundaries is to consider geometric features of the points in a local neighborhood. Bendels et al. [3] introduce such a method that uses features calculated using the neighborhood of a point to calculate the probability of a point being on a boundary and extract a chain of points to indicate the boundary.

Symmetric kε-Nearest Neighbors

Let P be the set of points found using Fixed-radius Nearest Neighbors and Q be the set of points found using k-Nearest Neighbors, Kε-Nearest Neighbors is then defined as P ∪ Q. Symmetric KENN of point p would then be P ∪ Q ∪ R where R is the set of

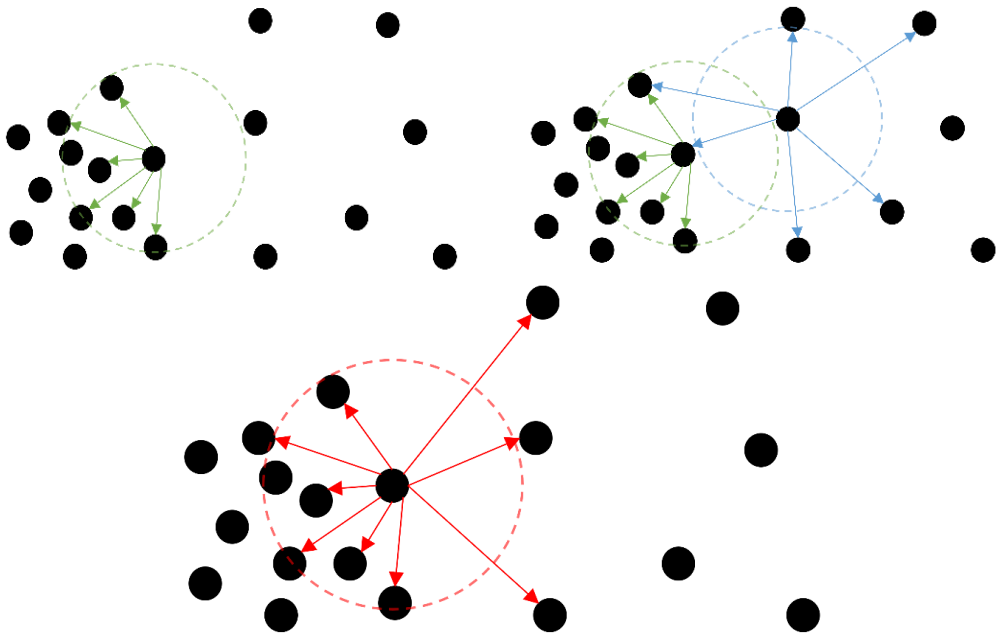


Figure 3-8 Symmetric Neighborhood Example

points whose neighborhood includes p. Figure 3-8 shows the construction of the Symmetric KENN neighborhood for a fixed radius (indicated by the circles) and k = 5.

In the top part of the figure, we can see *p* as the point with green neighborhood and q as a point in R with the blue neighborhood. The bottom part of the figure shows the final KENN neighborhood of p in red, which includes the additional points from all points q that would have p in their neighborhood.

Boundary Probability

Each point is assigned a probability of being on a boundary based on three criteria. All of the neighborhoods used in calculating each of these criteria uses the Symmetric kε-Nearest Neighbors method. Each of the criteria measures local geometric features of the point and contributes a value towards the total probability of the point being classified as a boundary point

Angle Criterion

For a point p and its neighbors $q_1,\ldots, q_n \in N_p$ the algorithm calculates the tangent plane and projects the points onto the plane. The points are then sorted according to their

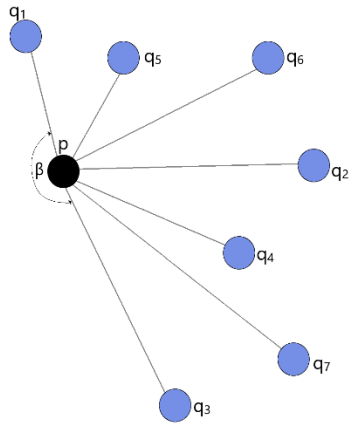


Figure 3-9 Max angle criterion

clockwise positions around p, as shown in Figure 3-9. The clockwise pairwise angle between each pair of neighbors are calculated and the largest angle g is stored along with the pair of vertices that form them. Finally, the boundary probability is calculated according to the equation

$$\Pi_{\angle}(p) = \min\left(\frac{g - \frac{2\pi}{|N_p|}}{\pi - \frac{2\pi}{|N_p|}}, 1\right).$$

Half disc Criterion

The half-disc criterion evaluates the deviation of a point from the average μ_p of its neighbors. The understanding is that points on the border will deviate significantly from the average while interior points will have a significantly lower deviation. A Gaussian kernel is used to reduce the influence of sampling density. μ_p is thus calculated as a weighted average of N_p using the kernel

$$g_\sigma(d) = \exp\left(\frac{-d^2}{\sigma^2}\right)$$

where $\sigma = \frac{1}{3} r_p$ and r_p is the average distance to the neighboring points. μ_p is then calculated by:

$$\mu_p = \frac{\sum_{q \epsilon N_p} g_\sigma(||q - p||) q}{\sum_{q \epsilon N_p} g_\sigma(||q - p||)}.$$

The points are then projected onto the tangent plane and the projection $\bar{\mu}_p$ of μ_p is used to calculate the boundary probability

$$\Pi_{\mu}(p) = \min\left(\frac{\left\|p - (\bar{\mu}_p)\right\|}{\frac{4}{3\pi} r_p}, 1\right).$$

The difference in deviation between the point and the neighborhood average for a boundary point and an internal point is shown in Figure 3-10.

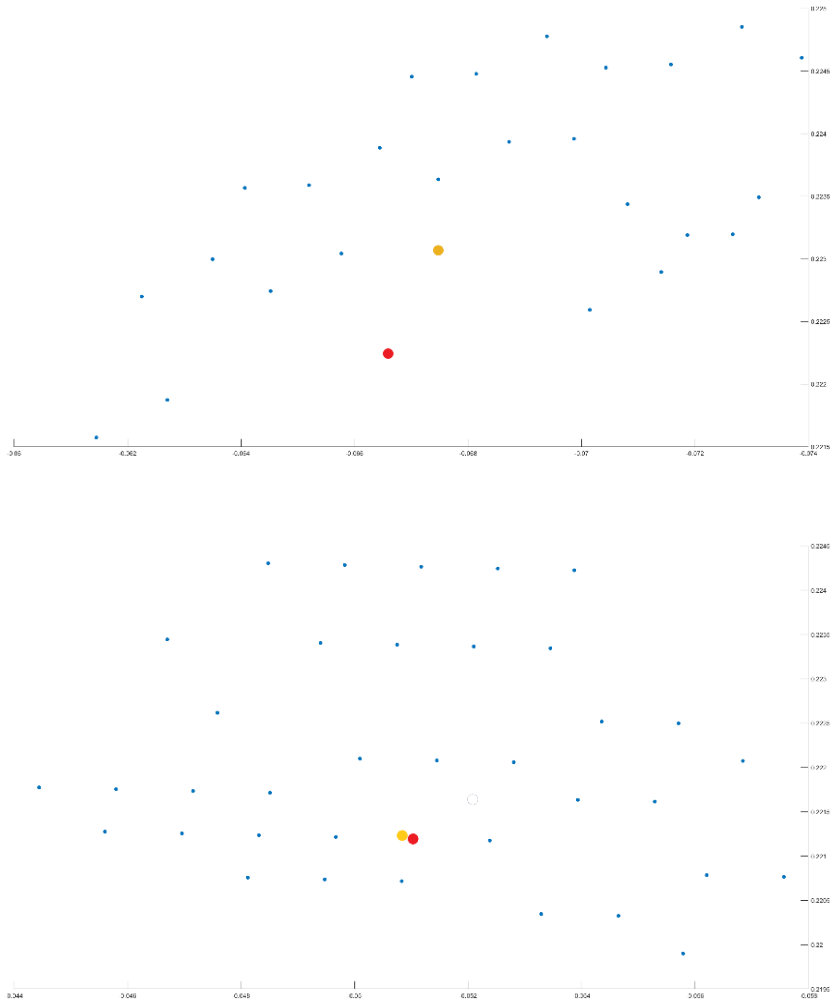


Figure 3-10 Neighborhood average deviation for boundary point (top) and internal Point (bottom)

Shape Criterion

The shape of the correlation ellipsoid formed by the KENN of p, N_p, approximately defines the form of the neighborhood and is itself encoded in the eigenvalues, λ_i, of the weighted covariance matrix C_p

$$Cp = \sum_{q \in N_p} w(q)(\mu_p - q)(\mu_p - q)^t$$

Using the first 3 eigenvalues, a decision vector $\Lambda_p = (\frac{\lambda_0}{\alpha}, \frac{\lambda_1}{\alpha}, \frac{\lambda_2}{\alpha})$ is then generated where $\alpha = \lambda_0 + \lambda_1 + \lambda_2$.

The four characteristic situations representing different neighborhood shape characteristics are shown in Figure 3-11.

- Φ = Boundary and $\Lambda_\varphi = (\frac{2}{3}, \frac{1}{3}, 0)$
- Φ = Interior and $\Lambda_\varphi = (\frac{1}{2}, \frac{1}{2}, 0)$
- Φ = Corner and $\Lambda_\varphi = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$
- Φ = Line and $\Lambda_\varphi = (1, 0, 0)$

We define T_Λ to be the triangle formed by the three vertices of Φ when Φ = {Interior, Corner, Line} and extract potential classification probabilities $\widetilde{\Pi}_\varphi$ of each situation using a Gaussian kernel g_σ where $\sigma = \frac{1}{3}||\Lambda_\varphi - centroid(T_\Lambda)||^2$. $\widetilde{\Pi}_\varphi$ is then calculated as

$$\widetilde{\Pi}_\varphi(p) = g_{\sigma_\varphi}(||\Lambda_p - \Lambda_\varphi||)$$

and normalized as

$$\Pi_\varphi(p) = \frac{\widetilde{\Pi}_\varphi(p)}{\sum_{\varphi \in \Phi} \widetilde{\Pi}_\varphi(p)}$$

The correlation ellipsoid formed by the neighborhood of points in each situation is shown in Figure 3-11.

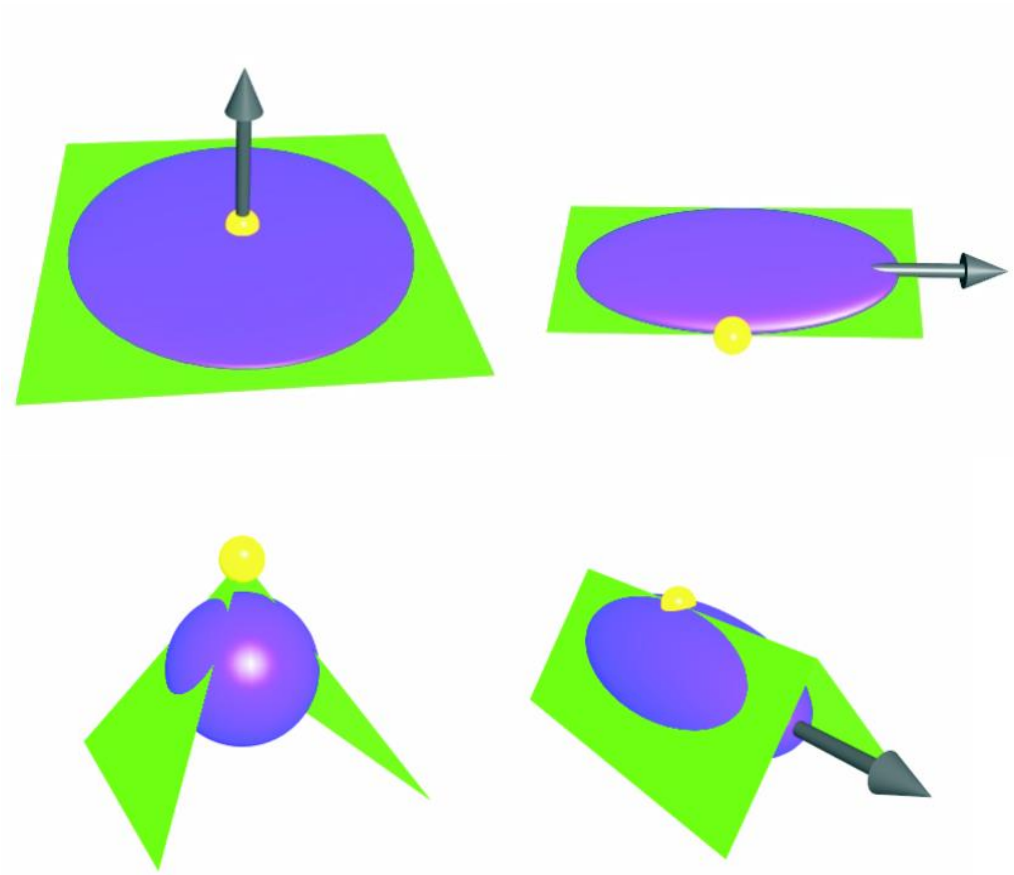


Figure 3-11 Correlation ellipsoids for neighborhood points surface point (top left), crease point (Bottom left), boundary point (top right), and corner point (bottom right) [5]

Criterion Weights

The probabilities from the criteria can be combined in different ways and it is suggested that for noisy data the weight of the shape criterion should be higher since it is less affected by noise. All of the results in this thesis uses a uniform weighting scheme resulting in the final probability equation

$$\Pi(p) = w_{\angle}\Pi_{\angle}(p) + w_{\mu}\Pi_{\mu}(p) + w_{\varphi}\Pi_{\varphi}(p)$$

where $w_{\angle} + w_{\mu} + w_{\varphi} = 1$ and $w_{\angle} = w_{\mu} = w_{\varphi}$.

Point Classification

A classification threshold is provided as an input and a point is only classified as a boundary point in the scenario where both of its max angle vertex neighbors calculated during the angle criterion are also classified as boundary points. This means that the boundary probability of all three points will need to exceed the provided threshold in order for the point being processed to be declared a boundary point. All of the results shown uses a classification threshold between 0.2 and 0.35.

Valid Edge Extraction

A subset of all of the edges identified is extracted based on if they are part of the max angle vertices for any border point. For each valid edge connecting points p_i and p_j, we calculate an associated edge weight using the equation

$$w_{total}(i,j) = w_{prob}(i,j) + w_{density}(i,j)$$

where

$$w_{prob}(i,j) = 2 - \Pi(p_i) - \Pi(p_j)$$

and

$$w_{density}(i,j) = \frac{2||p_i - p_j||}{r_{p_i} + r_{p_j}} \ .$$

A valid edge is only eligible to be added to the tree if both w_{total} and w_{prob} are below pre-defined thresholds. A combination of 3 and 1.1 for the respective thresholds was used for all the results displayed on this thesis.

Minimum Spanning Tree

After identifying all eligible edges, the edges are processed in an ascending order according to w_{total}. The edges are then connected to the graph if they join two distinct components of the graph together.

Loop Extraction

During the addition of eligible edges to the tree, a secondary check is performed to identify which connect vertices of the same component. These edges are still added to the tree, thus creating a graph, in order to complete the cycle based on a predefined loop length e which is calculated using the input radius parameter ϵ using the equation

$$e = \frac{2\pi\epsilon}{d}$$

where d is the average edge length of the graph.

Given a graph the loop extraction algorithm maintains a color for each of the vertices.

- White = Untouched
- Grey = Queued for visitation
- Black = Visited

Given a node the algorithm will add it to a queue and mark it as grey. It will then pop the queue and for each of the neighbors of the vertex mark it as grey. It will then mark the current node as black and then pop it off the queue to get the next node to visit. This process continues until it finds a node with a grey neighbor. The loop will then extract the

path by tracing back the steps of the search. Figure 3-12 shows this process on a simple example.

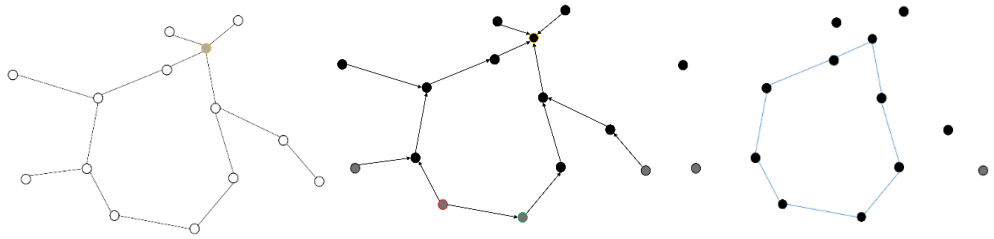


Figure 3-12 Breadth-First Search with the starting node (yellow), the search stops when it reaches the node(green) and detects the grey node(red) as a neighbor. The extracted loop is shown (left).

Combination of Topological and Geometric Hole Boundary Extraction

The feature-based loop extraction method suffers from two major drawbacks. Firstly, the criteria used cannot indicate the presence of points inside the boundary detected and sometimes results in inaccurate (or even erroneous) detection of a hole boundary. Secondly, the loop extraction which is run starting for each point in the point cloud takes a long time to run. Both of these issues can be solved by limiting the number of points to run the loop extraction from by providing an initial set of points to run on extracted from the Minimal 1-cycle boundary, B_{Min}, calculated using the Betti-1 edges found using the topological boundary method. This allows for a much faster runtime of the algorithm as well as increasing the accuracy of the hole boundary detected. The advantage of this combination of methods is that it can take advantage of the efficiency of the topological methodology while also taking into account local the geometric properties of the point cloud neighborhood.

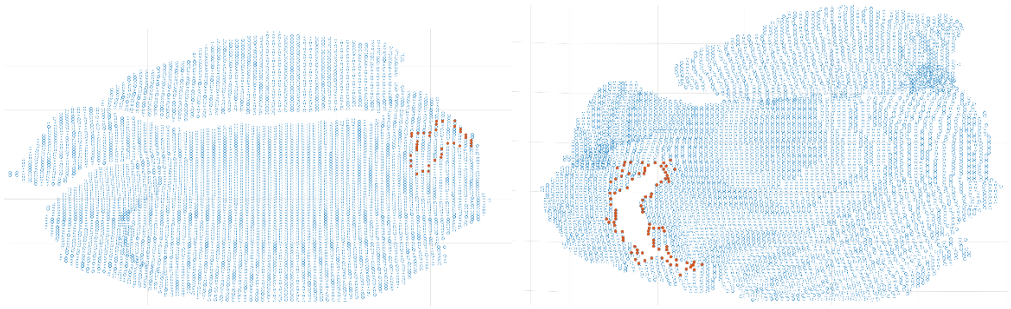


Figure 3-13 Unseeded boundary (left) vs Seeded boundary (right)

For the pitcher point cloud displayed above in Figure 3-13, the pictures show the resulting boundary loop extracted by running the original loop extraction algorithm using BFS and the combined method using the topological pre-seeding. The displayed model point clouds have been rotated in order to display the boundary points (displayed in red) without overlap. The effect of seed points can be seen as the seeded boundary outlines an actual hole and the unseeded method does not. The effects in runtime can be seen in Figure 3-14 as the seeded method (shown as the left bar in the bar graph) only has to run through 94 points and thus takes 81 seconds while producing a boundary which encircles the opening when compared to 2190 seconds necessary to run the unseeded method that has to start from all 5820 points in the cloud and fails to provide a correct boundary.

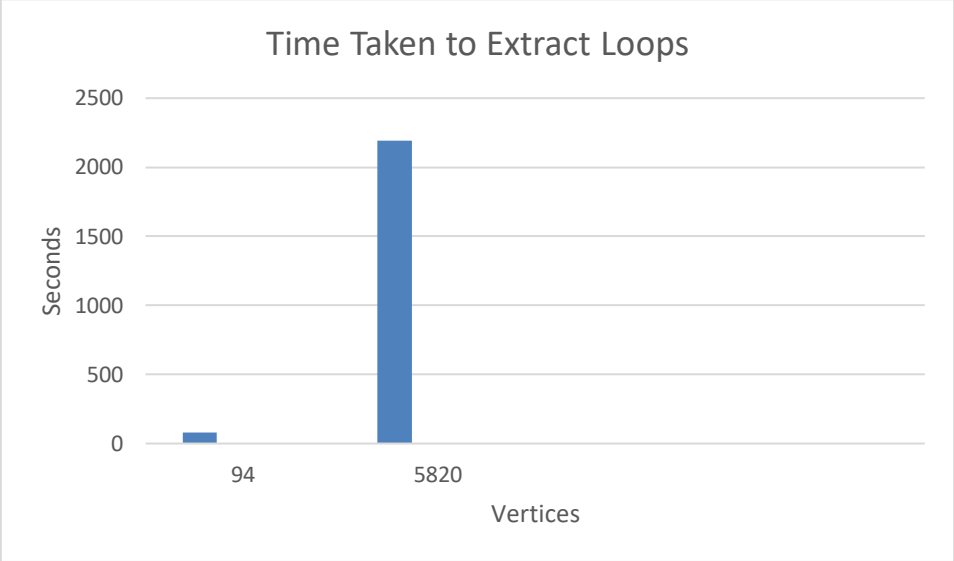


Figure 3-14 Bar chart of the time taken to extract loops using the combined algorithm (left) and the feature-based algorithm alone (right).

Loop Constriction

A hole boundary with a lower number of vertices can be achieved by adding the valid edges determined by their weights to the minimum spanning graph used to calculate the minimum 1-cycle boundary. This is done by adding max angle edges for each point in the boundary such that the neighbor which forms the max angle is in the set of points formed by the initial 1-Cycle boundary, $B_{Initial}$, and the points inside it, $P_{Interior}$. The methodology for adding the edges is shown in Algorithm 3-1. This allows for edges which might allow for a tighter boundary around the hole to be discovered and added to the graph. After adding the extra edges, rerunning the pathfinding algorithm provides us with a boundary with a lower number of vertices as can be seen in Figure 3-15. Constriction usually helps in paths with a large number of vertices and does not always allow for the elimination of vertices.

```
1.    for {∀p | p ϵ B_Min}do
2.        n1 ← p.max_angle_vertex[1]
3.        n2 ← p.max_angle_vertex[2]
4.        if n1 ϵ {B_Initial ∪ P_Interior}
5.            G_topological.Edges = G_topological.Edges ∪ {p, n1}
6.        end if
7.        if n2 ϵ {B_Initial ∪ P_Interior}
8.            G_topological.Edges = G_topological.Edges ∪ {p, n2}
9.        end if
10.   end for
```

Algorithm 3-1 Adding extra edges to Topological Graph

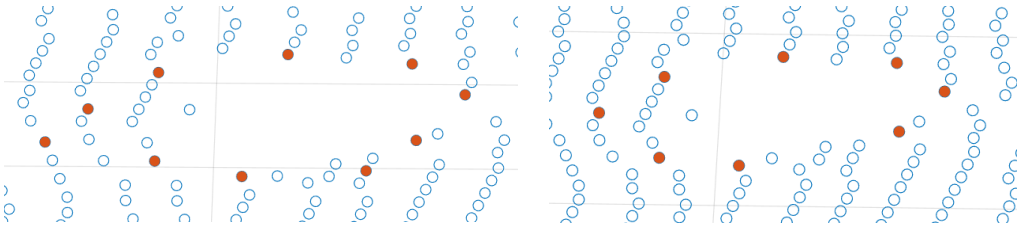


Figure 3-15 Minimal 1-Cycle Boundary (Left) Constricted Boundary (Right)

Accuracy Metrics

With the multiple methods and approaches taken we are presented with a number of boundaries for a given hole after the algorithm has concluded running. Since ground truth for a hole boundary is ambiguous we are left with visual inspection in order to determine which one fits our requirement the best.

The three boundaries presented in Figure 3-16 are:

- Minimal 1-cycle boundary
- BFS Loop extraction from Kruskal's minimum spanning graph
- Constricted 1-cycle

The best amongst the three would depend on the use case of the boundary since each has its benefits and drawbacks. The BFS Loop which usually has the highest granularity does not always provide a clean chain of points. The Minimal 1-cycle, though provides a cleaner chain of boundary points, is not as granular as the BFS loop. The constricted 1-cycle has the lowest number of points, however, has a lower resolution as a result.

A few heuristics can be calculated which indicate the quality of the boundary being extracted. We can calculate the average value of the boundary probability of a chain of vertices to indicate how high the boundary probability of the chain is. A second heuristic might be the number of points which fall inside the polygon of a given chain of points. A high number of points might indicate the possibility of further constriction since there might be points inside the hole boundary surface. Different holes and their boundaries, as well as the corresponding values for these 3 metrics, are shown in Figure 3-16.

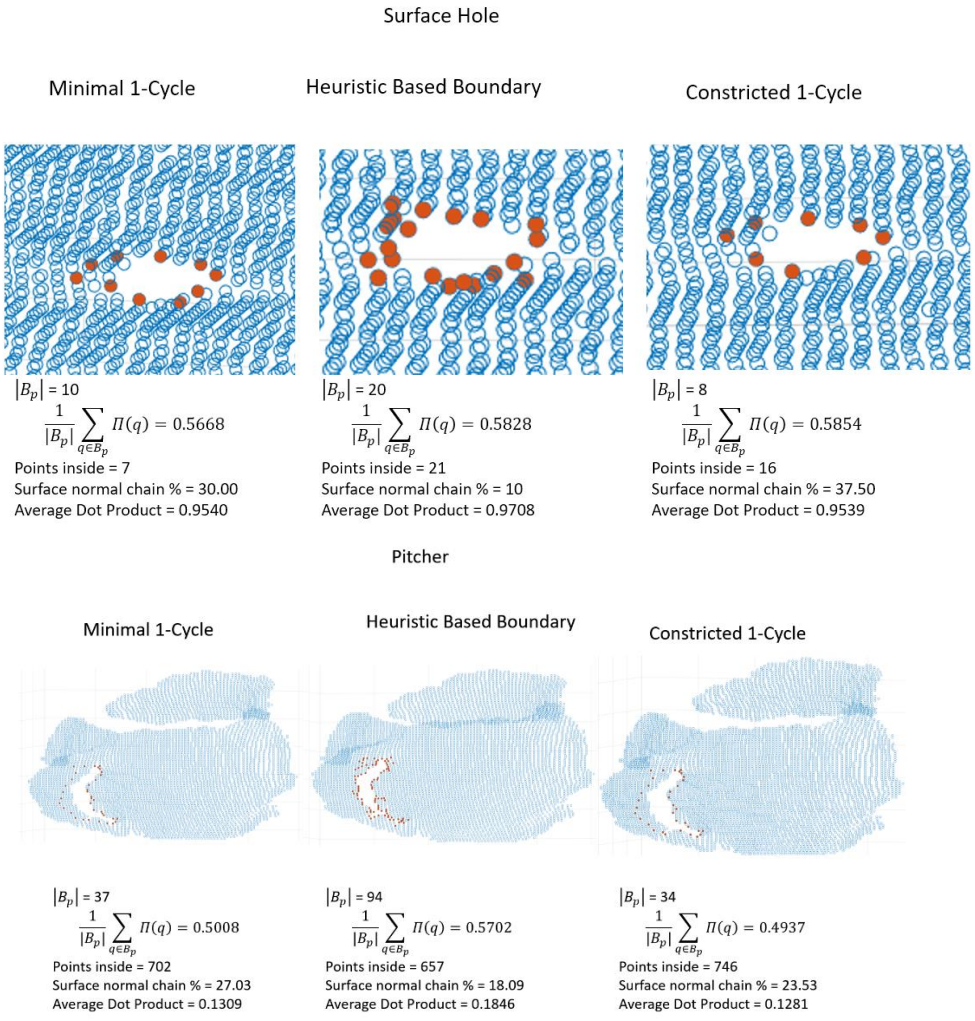


Figure 3-16 Boundaries for physical holes (Bottom) and Surface Holes (Top) with their

Chapter 4

HOLE CLASSIFICATION

The holes present in point clouds can be categorized into two types, physical holes, and surface holes. A surface hole is identified by the hole being an opening on the surface of the point cloud which is approximately a single layer thick. A surface hole could correspond to a hole in a very thin surface but most of the time will represent the absence of information about the enclosed region either due to not having been observed or due to missing sensor signals as a result of reflections or excessive glare at that location. A physical hole is defined as a hole which is surrounded by a surface of points, such as a donut. A physical hole is an important feature of an object as it indicates a region of interest for many tasks such as assembly or grasping.

Approach

The method detailed in this chapter is based on the fact that for a point in a boundary of a hole the surface normal will either align with or be approximately perpendicular to the surface normal of the plane approximated by the set of points that make up the boundary. Thus, we compute the dot product of the surface normal of the hole with the surface normal of each of the boundary points and use a threshold to classify each point as either having a surface normal parallel or orthogonal to the hole. A hole is classified as surface or physical based on the number of points in a chain around the boundary that have the same classification.

Related Work

Very little work has been performed to identify the characteristics of holes in 3D point clouds. In the main work in this area, Nadeer et al. [7] uses multiple images to generate depth and visibility maps. These maps are then used in a graph-cut segmentation

to detect holes. Intensity and depth measures for hole are then used to classify the hole as either real or virtual.

Hole Surface Normal

In the approach proposed in this thesis, the surface of the hole is first estimated by finding the least square fit plane to the chain of border vertices. This gives us a surface normal that can be used to ascertain the orthogonality of the surface normal of the surrounding points.

Boundary Normal Orthogonality

Each of the vertices on the boundary will have a surface normal associated with it based on its neighborhood. The dot product of the surface normal of the boundary points with the surface normal of the hole reveals how orthogonal each of the surface normals of the vertices is to the surface normal of the hole. We take the absolute value of the dot product allowing us to align the surface normals to a specific direction and thus allow us to calculate the chains in the classification step accurately. This information is relevant since the boundary points for a physical hole will always have surface normals for a visible section of its boundary orthogonal to that of the surface normal of the hole which is not the case for surface holes. Figure 4-2 displays the surface normals for a physical and surface hole and their boundaries.

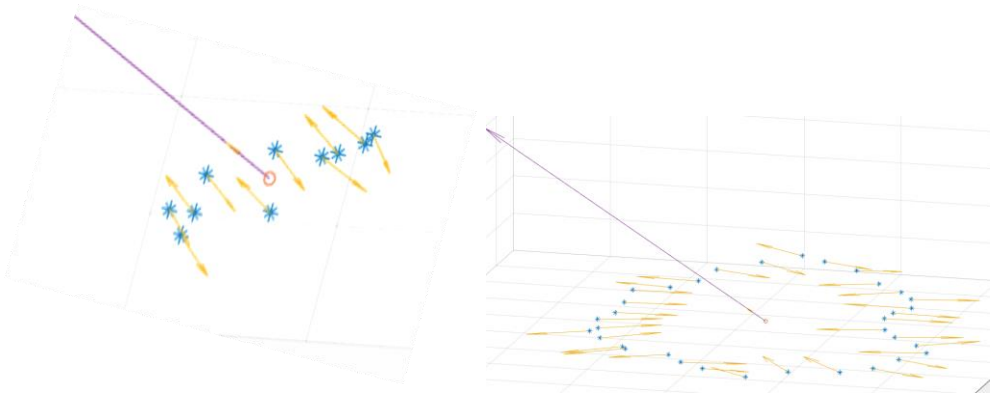


Figure 4-1 The surface normal for a Surface Hole (Left) and Physical hole (Right) marked in purple and the surface normal for the boundary points in yellow

.

Thresholding and Classification

After the dot products are calculated we classify each of the dot products as either parallel or perpendicular based on a threshold of angular difference. For the results shown in this chapter, the threshold was 20 degrees. A secondary threshold was set to be the percentage of the total number of points in a sequence around the border that needed to be classified in order for the hole to be classified as surface or physical. The threshold calculated was 25% using the predefined dot product threshold. Thus a hole would be classified as physical if 25% of the points in a sequence around the border had a dot product less than 0.34. The methodology shown in Algorithm 4-1 outlines the approach where dot_{array} contains the dot product of the hole surface normal with the boundary surface normals. The orthogonality check adds dot-products to the list if the chain has already started and stops if a dot-product exceeding the threshold is observed. The resulting array and the extracted chains are shown in Figure 4-1. If the size of the maximum chain extracted is 0, we re-run the algorithm with the threshold flipped to calculate the

characteristics of the surface hole. Finally, in Figure 4-3 we show the results of running the algorithm on a physical and a surface hole.

```
1.  chains ← {}, j ← 0, chain ← false
2.  for i ϵ [0, |dot_array|] do
3.    if dot_array[i] is orthogonal
4.      if chain is true
5.        append dot_array[i] to chains{j}
6.      else
7.        chains{j} ← dot_array[i]
8.        chain ← true
9.    else
10.     if chain is true
11.       j ← j + 1
12.       chain ← false
13.     else
14.       continue
15. end for
```

Algorithm 4-1 Orthogonal Chain Extraction

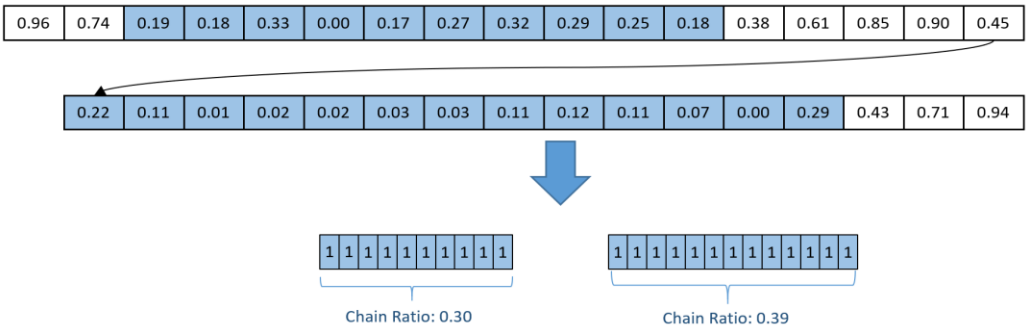
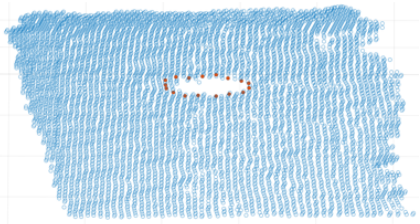


Figure 4-2 Orthogonal Chain Extraction

Surface Hole

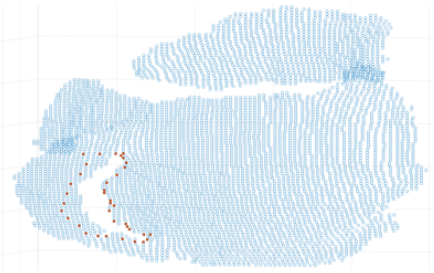


$|B_p| = 17$

$$\frac{1}{|B_p|}\sum_{q\in B_p} \Pi(q) = 0.5272$$

Points inside = 30
Surface normal chain % = 35.29
Average Dot Product = 0.9684

Minimal 1-Cycle



$|B_p| = 37$

$$\frac{1}{|B_p|}\sum_{q\in B_p} \Pi(q) = 0.5008$$

Points inside = 702
Surface normal chain % = 27.03
Average Dot Product = 0.1309

Figure 4-3 Hole Boundaries and their classification information

Chapter 5

CONCLUSION

Conclusion

This thesis focused on the extraction of hole boundaries using a combination of two previously explored approaches as well as demonstrating the improvement in each approach by using elements from the other. It also classified the hole whose boundary has been detected using surface normal information. Accuracy metrics were also provided that try to quantify the quality of the hole boundary detected. The effectiveness of the resulting algorithm was demonstrated using depth scans of real-world objects for physical holes and artificially generated surface holes that represent data holes.

Future Work

The boundary provided for physical holes fails to represent the true surface that surrounds a physical hole and thus a better approach is necessary where surface information may be incorporated more effectively. Like [1] which uses topological persistence as a descriptor for the classification of point cloud objects, the parameters used to calculate hole type can have an extended use case as a descriptor for other classification purposes. The geometric approach [3] could be improved by additional criteria during loop extraction to indicate the presence of points inside the extracted boundary.

References

[1] William Beksi, Topological Methods for 3D Point Cloud Processing. *Ph.D. thesis*, University of Minnesota, 2018.

[2] Herbert Edelsbrunner, David Letscher and Afra Zomorodian, Topological Persistence and Simplification*. Discrete and Computational Geometry*, 28(4):511-533, 2002.

[3] Gerhard H Bendels, Ruwen Schnabel, and Reinhard Klein. Detecting holes in point set surfaces. *Journal of WSCG*, 14, 2006.

[4] Cecil Jose A Delfinado and Herbert Edelsbrunner. An incremental algorithm for Betti numbers of simplicial complexes on the 3-sphere. *Computer Aided Geometric Design*, 12(7):771–784, 1995.

[5] Stefan Gumhold, Xinlong Wang, and Rob MacLeod. *Feature Extraction from Point Clouds.* 10th International Meshing Roundtable, 295-305, 2001.

[6] Van Sinh Nguyen, Trong Hai Trinh, and Manh Ha Tran. Hole Boundary Detection of a Surface of 3D point clouds*. 2015 International Conference on Advanced Computing and Applications (ACOMP)*, Ho Chi Minh City, 2015, pp. 124-129.

[7] Aldeeb N. and Hellwich O. (2017). Detection and Classification of Holes in Point Clouds. In Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 6: VISAPP, (VISIGRAPP 2017) ISBN 978-989-758-227-1, pages 321-330.

[8] J. Wang and M. M. Oliveira, "A hole-filling strategy for reconstruction of smooth surfaces in range images," 16th Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI 2003), Sao Carlos, Brazil, 2003, pp. 11-18.

[9] Mineo, Carmelo & Pierce, Stephen & Summan, Rahul. (2018). Novel algorithms for 3D surface point cloud boundary detection and edge reconstruction. Journal of

Computational Design and Engineering. 10.1016/j.jcde.2018.02.001.

Biographical Information

Aaqif Muhtasim completed his Bachelors of Science in Computer Engineering in Spring of 2016 from the University of Texas at Arlington. He has worked as a Teaching Assistant for Embedded Systems as well as a full-stack developer for multiple web-based applications. His interests lie in developing robotic systems that help increase productivity. He completed his Masters of Science in Computer Engineering in Fall of 2016 with a focus on Systems/Architecture.