2014

Segmentation And Spatial Depth Ridge Detection Of Unorganized Point Cloud Data

James Clark Church
University of Mississippi

SEGMENTATION AND SPATIAL DEPTH RIDGE DETECTION OF UNORGANIZED POINT CLOUD DATA

A Dissertation
presented in partial fulfillment of requirements
for the degree of Doctor of Philosophy
in the School of Engineering
The University of Mississippi

by

James Clark Church

July 2014

ABSTRACT

Visual 3D data are of interest to a number of fields: medical professionals, game designers, graphic designers, and (in the interest of this paper) ichthyologists interested in the taxonomy of fish. Since the release of the Kinect for the Microsoft XBox, game designers have been interested in using the 3D data returned by the device to understand human movement and translate that movement into an interface with which to interact with game systems. In the medical field, researchers must use computer vision tools to navigate through the data found in CT scans and MRI scans. These tools must segment images into the parts that are relevant to researchers and account for noise related to the scanning process all while ignoring other types of noise such as foreign elements in the body that might indicate signs of illness.

3D point cloud data represents some unique challenges. Consider an object scanned with a laser scanner. The scanner returns the surface points of the object, but nothing more. Using the tool Qhull, a researcher can quickly compute the convex hull of an object (which is an interesting challenge in itself), but the convex hull (obviously) leaves out any description of an object's concave features [4]. Several algorithms have been proposed to illustrate an object's complete features based on unorganized 3D point cloud data as accurately as possible, most notably Boissonnat's tetrahedral culling algorithm and The Power Crust algorithm [6] [1]. We introduce a new approach to the area partitioning problem that takes into consideration these algorithms' strengths and weaknesses.

In this paper we propose a methodology for approximating a shape's solid geometry using the unorganized 3D point cloud data of that shape primarily by utilizing localized principal component analysis information. Our model accounts for three common issues that arise in the scanning of 3D objects: noise in surface points, poorly sampled surface area,

and narrow corners. We explore each of these areas of concern and outline our approach to each. Our technique uses a growing algorithm that labels points as it progresses and uses those labels with a simple priority queue. We found that our approach works especially well for approximating surfaces under the condition where a local surface is poorly sampled (i.e a significant hole is present in the point cloud). We then turn to study the medial axis of a shape for the purposes of 'unfolding' that structure. Our approach uses a ridge formulation based on the spatial depth statistic to create the medial axis.

We conclude the paper with visual results of our technique.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Visual 3D data are of interest in a number of fields: medical professionals, game designers, graphic designers, and (in the interest of this paper) ichthyologists interested in the taxonomy of fish. Since the release of the Kinect for the Microsoft XBox, game designers have been interested in using the 3D data returned by the device to understand human movement and translate that movement into an interface with which to interact with game systems. In the medical field, researchers must use computer vision tools to navigate through the data found in CT scans and MRI scans. These tools must segment images into the parts that are relevant to researchers and account for noise related to the scanning process all while ignoring other types of noise such as foreign elements in the body that might indicate signs of illness.

In the interest of this paper, we look at unorganized 3D point cloud data of elongated structures. Specifically, we are interested in studying the 3D surface data of living creatures with the natural ability to twist and contort into various shapes (such as eels) and unfold that surface data in order to study the biological features across multiple specimens in a consistent environment without having to touch the physical specimens beyond the scanning process.

3D point cloud data represents some unique challenges. Consider an object scanned with a laser scanner. The scanner returns the surface points of the object, but nothing more. Using the tool Qhull, a researcher can quickly compute the convex hull of an object (which is an interesting challenge in itself), but the convex hull (obviously) leaves out any description of an object's concave features [4]. Several algorithms have been proposed to illustrate

an object's complete features based on unorganized 3D point cloud data as accurately as possible, most notably Boissonnat's tetrahedral culling algorithm and The Power Crust algorithm [6] [1]. We introduce a new approach to the area partitioning problem that takes into consideration these algorithms' strengths and weaknesses.

This dissertation looks at two problems in the field of computational geometry: the recovery of the features represented by unorganized point cloud surface data and the discovery of the medial axis of a shape for the purposes of 'unfolding' that structure. We then conclude with example images of our algorithm and propose ways to handle some of the current algorithm's points of failure.

Taxonomy is the field by which specimens are grouped based on the characteristics that a collection of specimens have in common and then creating labels to identify these groups. Taxonomists are confronted with several challenges facing their field [10] [11]. First, there are many portions of the world that taxonomists have not yet reached and are rich with undiscovered species. Second, there is a decrease in taxonomists world wide over the past few decades who are qualified to describe new species. Third, we are in a period described as a *biodiversity crisis* where many species are facing extinction and it is feared that many species will die before being discovered. One tool that can be used to assist the taxonomist is automated species identification by which a specimen is scanned using a 3D scanner, analyzed, and a species is identified using a heuristic function. If the heuristic function fails to identify the species of a specimen with a certain degree of confidence, it is possible that the specimen is representative of a new species. This dissertation looks at one particular challenge within the scope of automatic species discovery: unfolding an elongated fish specimen for the purpose of performing species identification. In order to identify a specimen, one must first extract several geometric features from the specimen by first flattening the specimen. For many specimens this is trivial. However, for some specimens that have been stored in jars (such as eels) the specimen takes the shape of the container making it difficult to extract features. Our goal is to estimate the medial axis of

elongated fish specimens for the purpose of unfolding the specimen.

The medial axis is the set of all points within an object with the shortest distance to at least two different points on that object's surface, where "distance" (typically Euclidean) is determined by the application. It is an important tool in computer geometry applications in order to determine the "skeleton" of a shape or to approximate surface reconstruction. We focus on medial axis estimation with the given assumption that our input shape consists of only sample points along the surface and that holes exist. The surface scans are created using 3D scanners and the problems related to scanners are discussed by Bajaj et al. [3].

1.1 Paper Organization

The remainder of this work is organized as follows. In Chapter 2 we review the abundance of papers that are closely related to this work, specifically looking at the fields of medial axis estimation and surface reconstruction. In Chapter 3 we introduce our original algorithm for region growing in three dimensions and demonstrate its effectiveness. We devote all of Chapters 4 and 5 to testing the approaches developed from the previous chapter. In Chapter 6 we introduce an original proof for identifying ridges in three dimensions and we outline an approach to unify chains of ridges and showcase our results. Our proof was published in [13]. Chapter 7 is a unification of Chapters 3 and 6 where we combine these two algorithms (shape segmentation and ridge detection) into an approach for detecting internal ridges in three dimensional space for the purposes of shape manipulation. In Chapter 8, we conclude with several images that demonstrate our algorithm compared to other popular approaches.

CHAPTER 2

LITERATURE REVIEW

2.1 Ridge Detectors

Ridge detectors are an element of computer vision that help to simplify the analysis of images so that ridges may be applied to unique applications. The formulation of a ridge in at least two dimensions was first attempted by Haralick [19] with the facet model. Rather than looking at an image as a series of points, he proposed that an image be identified as a two dimensional signal by which we could compute first and second derivatives at any point and in a given direction. A point could be considered a 'ridge' or 'valley' provided that the first derivative at that location was equal to 0. Harlick's model used 10 five-by-five masks in order to determine the coefficients used in a 10-degree polynomial and a directional vector by which the ridge/valley could be followed.

Ridge and edge detection are closely related. Canny created his edge detector in 1986 [7]. His approach was to first apply a Gaussian filter to an image, then evaluate each point to test for an edge with a high threshold parameter. For the set of points that can be confirmed as edges, the algorithm determines the direction of an edge and attempts to find more edge points along the ridge detection that meet a lower threshold parameter. He sets out the three criteria for an edge in two dimensions. *Accurate* : the edge detector should correctly labels edges where they exist and not label edges where they don't exist. *Robust* : the edge detector correctly labels edges independent of image noise. *Localized* : the edge detector should labels edges as close as possible to the edge's location. We apply the same criteria to a ridge detector in three dimensions.

Most importantly, we focus on the work of Lindeberg [24] describing edge and ridge

detection in two dimensions. While there is some work on extending the work of Lindeberg in three dimensions [33], it appears that previous attempts at this use ad hoc methods for formulating a 3D ridge. What we present is a complete work on extending ridge detection to three dimensional images and we set ourselves up for future work in ridge detection beyond three dimensions.

Traditionally a ridge is defined in two dimensions as a raised separator (e.g. a mountain range) which divides two distinct regions. The ridge itself extends in two directions orthogonal to the two regions. Describing the property of a ridge in three dimensions requires us to think abstractly. We define a ridge as a raised separator which divides an area along two directions which are orthogonal to each other. A third direction orthogonal to both of the two directions represents the direction of the ridge itself. In an n-dimensional setting, we define a ridge as a raised separator which divides an area along n-1 orthogonal directions. The n-th direction (also orthogonal) is the direction of the ridge. Based on this definition, we can set the following criteria that must be met for a ridge: (1) There should be a negative gradient change in two orthogonal directions from a point; (2) In the direction orthogonal to this gradient change, there should be no significant gradient change; (3) This should be invariant to scale; (4) This should be robust with respect to noise.

Pellegrino et al. set out to improve upon the classic ridge detector created by Canny [28]. The work defines ridges that were 1) invariant to scale; 2) handle difficult junctions where 3 ridges meet at a point; 3) robust with respect to noise; and 4) use as few parameters as possible.

2.2 Surface Reconstruction not Involving the Medial Axis

Hoppe et al. made an early attempt at surface reconstruction without the traditional techniques involving medial axis reconstruction or function reconstruction [21]. The paper stresses the difference between their approach and function reconstruction. Function reconstruction implies that the input surface M and output surface S can be represented as

function parameters $x_i \in M$ and $y_i \in S$ so that a function $f\colon M \to S$ can be determined such that $f(x_i) \approx y_i$. The key approach of this paper is broken into two stages: (1) Estimate the distance each surface point x_i is from the true surface of M and (2) recreate the surface using a contouring algorithm. To estimate the tangent plane at each point, principle component analysis (PCA) is used. PCA will create a normal to the tangent plane, but it won't provide a consistent direction to that tangent plane. To handle this, a Riemannian Graph is formed and points that are connected are oriented to the similar normal vectors by flipping the sign of inconsistent vectors. The order of the graph traversal is important in that a vector is selected and a minimum spanning tree is formed and all vectors are oriented to align with the starting vector. Once the planes are oriented, each normal vector is represented with a rectangle. The rectangles are then moved into alignment using a marching cubes algorithm.

Curless and Levoy worked on the problem of creating 3D images using a laser scan of a physical object [15]. The key to Curless' approach is creating a tessellation of the surface points from a range scan of the image. The Curless method depends on a tessellation algorithm to fill in gaps in structures (common in 3D surface scanning). Their approach is speedy and does a good job of estimating the surface of detailed structures using a weighted voxel approach. After computing the weight of a voxel at each point in an image, the zero-crossing isosurface is extracted and the image is formed. Voxels without a weight are ignored in the process.

There are two methodologies of reconstructing a shape using unstructured point cloud data: estimating the medial axis and estimating the surface. We look at a variety of papers that examine both approaches.

2.3 Medial Axis Estimation

Blum first proposed the concept of the medial axis transform by outlining the idea of wave propagation [5]. That idea was first called "grass fire" (now called the "prairie fire" analogy): if a fire is lit along the perimeter of a shape, it should burn inward, leaving the

internal skeleton of the object at the points where the fire is quenched. The goal of this paper was to consider the mathematical properties of shapes that did not appear to have a clear mathematical basis. Consider waves propagating out from two points on a plane. Normally the 'flow-through' will allow waves to propagate through each other, but in Blum's simplified model, once a wave comes in contact with another wave the two waves stop. The point where waves cancel each other will be considered the medial axis. The paper realizes that most of the work in distance transformations has been in the area of the Euclidean distance measure and ideally suited for Medial Axis Functions, but non-Euclidean distance measures should be applicable to this work as well.

The process of finding a medial axis led us to researching papers that had a similar theme to the medial axis, such as the concept of thinning binary shapes. In 1984, Zhang and Suen developed a parallel strategy for thinning 2D binary images [35]. The algorithm takes as input a 2D binary image and each iteration of the algorithm removes contour points that satisfy a few properties. The image is thinned so that shapes within the original image are still connected in the output. The algorithm is quite efficient at reaching a connected thinned image, but sometimes the thinned image lies along the contour of the existing shape, rather than along internal pixels of the shape.

In [32], Rosin, working off of the work of Zhang and Suen in [35], developed a strategy for straightening and partitioning 2D shapes. After performing the thinning step, the branches of the medial axis are iteratively removed until a single line exists with the shape. Each point along the edge of the image is mapped to a point along the remaining line of the shape. The line is straightened, and all of the mapped surface points move accordingly. In addition to straightening, Rosin worked on the idea of shape partitioning, but this is more of a discussion and the paper does not divulge any technical details into how shape partitioning might work.

The λ-medial axis by Chazal and Lieutier [9] is a relatively simple algorithm for approximating the medial axis of a noise-added set of sampled surface points. This paper

highlights a major issue in Medial Axis detection: "small variations in the boundary of an object result in large variations of its Medial Axis." The first phase of the algorithm to determine the Delaunay cells of a shape. Each cell of the Delaunay output that has a circle radius larger than a threshold λ is included in the shape's λ-medial axis. This idea provides a picturesque "reality" of the medial axis of a shape rather than the exact medial axis.

2.4 Surface Estimation

In the same year that Zhang and Suen worked on image thinning, Boissonnat first proposed that the surface of the shape can be estimated using the set of points produced by a Delaunay triangulation [6]. The author acknowledges that as the points in the shape increase, a better approximation of the shape will be produced. At any rate, the Delaunay triangulation can be used to create a minimum representation of the shape's surface. This paper proposes an algorithm for tetrahedron culling. The Delaunay triangulation algorithm by itself produces a convex hull of the original shape, which is only marginally useful. With tetrahedron culling, the crevices of the shape are revealed. Edelsbrunner and Mücke [18] took Boissonnat's ideas and gave them a more formal definition using the α-shape algorithm.

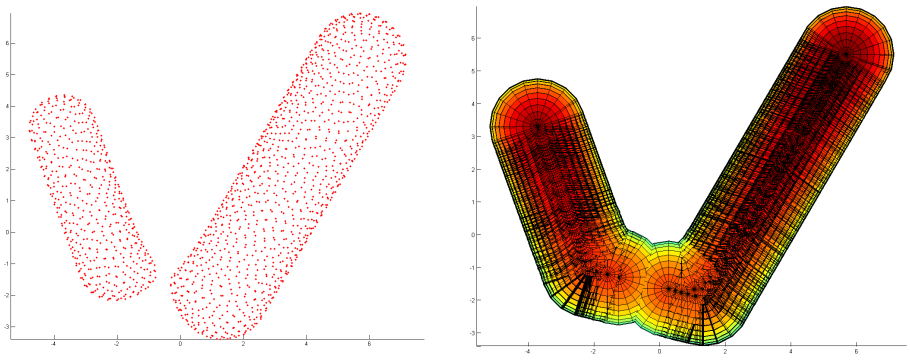
Mangan and Whitaker use a watershed segmentation algorithm for the purposes of segmenting a 3D surface mesh into a collection of polygons [25]. This collection of polygons could be used to represent distinct appendages on a fish specimen that would be treated independently by a straightening algorithm. The watershed approach is as follows: (1) The curvature of each vertex is calculated based on a height function. Typically this height function is represented by $f(x, y) = z$. (2) A unique label is assigned to the minimum adjacent point to the vertex. (3) Label the flat areas as either minima or a plateau. (4) Slowly reduce the height of each plateau and once a plateau reaches a minima, merge the two. Repeat this until all plateaus have merged into minima. (5) Slowly reduce the height of unlabeled vertices and once a vertex reaches a minima, merge the two. (6) Merge regions that have a depth within a threshold. This method is susceptible to noise and without

proper noise removal, this method produces lots of nonsensical segmentation. The algorithm requires the user to adjust the threshold value to each situation.

Often a perfect knowledge of an object's surface does not exist. The scanning process of a object is sampled from different vantage points and noise is introduced, leaving the object incomplete. For this reason, medial axis estimation techniques were developed. The relevant literature focuses on Delaunay or Voronoi decomposition of points in order to reconstruct the medial axis and surface reconstruction of a surface. Amenta et al. [2] worked on surface reconstruction with an emphasis on the maximal union of balls, known as The Power Crust. Amenta argues that the internal Voronoi centers of a complete surface do not sufficiently describe the surface of a shape, but do provide an approximation of the medial axis. This work went on to become The Power Crust which uses the Voronoi decomposition and focuses on narrow structures in shapes that lead to confusion as to which points are "interior" or "exterior" [1].

One interesting feature of The Power Crust is the requirement of a 'watertight reconstruction' of objects. A watertight reconstruction of an object means the algorithm ensures that there will be no holes in the output process. (In this dissertation, we have a plan for managing holes that occur in the scanning process.) While a watertight reconstruction algorithm is nice, it doesn't always produce an intended shape, as seen in Figure 2.1. For example, a flute is an elongated tube containing a cylindrical cavity and holes placed along the outer surface. In a reconstruction of the flute, the reconstruction should contain this cylindrical cavity and holes because those are features of the original object. We argue that a watertight algorithm isn't always the desired output of a reconstruction.

Sampled surfaces often lead to the problem of gaping holes where a surface should be represented after Boissonnat's tetrahedral culling algorithm. Dey and Goswami created the Tight Cocone algorithm [17]. The Tight Cocone algorithm is an evolution of The Power Crust. The Power Crust worked on the basic assumption that all areas of the shape were sufficiently sampled. Dey and Goswami had a different working assumption: "Principle of



(a) Hot Dogs Scan (b) The Power Crust interpretation

Figure 2.1: The image on the left depicts two scanned objects in the same scene known as Hot Dogs. The image on the right is the Power Crust interpretation of the Hot Dog scan. Notice that the resulting objects have been merged into one single object.

locality: The undersampling is local". The Power Crust and the Tight Cocone worked in much of the same way: rebuild the shape where sampling is sufficient. The Tight Cocone attempt to detect undersampled areas and concentrate on those areas separately. As noise is applied to the surface of an object, the algorithm will try to repair the surface, but only up to threshold value. Eventually the algorithm fails to return a surface on areas that are deemed too noisy for an accurate repair.

Mitra and Nguyen looked at estimating the surface normals of unstructured point cloud data [26]. Estimating surface normals is important for building the surface polygons used to recreate the shape. In [14], Coeurjolly looks at using a reverse of the Euclidean Distance Transformation to identify control points of a medial axis of an n-dimensional structure. He begins the paper by outlining the three key ways of identifying the medial axis of a shape.

1. The Prairie Fire Model: waves propagate out from the points and intersections indicate the medial axis. First proposed in [5].

2. Ridge Detection Model: distances from points are aggregated and points that form 'peaks' of large distance form the medial axis.

3. Maximal Disk Model: The medial axis is formed by the set of all of the possible maximal disks (sometimes called maximal balls) that are within the interior of the shape.

Based on the following papers, we can properly define the following terms.

1. The skeleton is the centers of the union of maximal balls that represent a shape.

2. The Voronoi decomposition of a shape represents the medial axis of a shape only if the surface of the shape is complete and known.

2.5 3D Region Growing Algorithms

The field of 3D region growing algorithms is preoccupied with medical images such as MRI scans and CT scans [27] [31] [23] [29] [16] [30]. The majority of the images handled by these algorithm represent 3D volume data created by X-Rays while the algorithms focus on segmenting the volume images into relevant areas while navigating around noise. In the last few years, region growing algorithms have taken off again, thanks in part to research being done with the Kinect add-on for the Microsoft XBox [34] [20]. This class of algorithms takes the unorganized point cloud data obtained from a single vantage point (the Kinect's camera) and attempts to cluster that data into individual (and hopefully meaningful) parts.

Chauve et al. created an algorithm that recreates the surfaces of unorganized point cloud data [8]. Much like the algorithm proposed in this paper, the algorithm proposed by Chauve et al. utilizes a localized Principal Component Analysis in order to create a 3D scene. This approach uses a single vantage point in order to reconstruct a 3D surface of a scene with the assumption that the scene represents an outdoor scene with architecture. Using this assumption allows the algorithm to assume the location and shape of missing parts of building parts in order to fill in holes with 'ghost' structures.

CHAPTER 3

IMAGE SEGMENTATION

3.1 Changes to this document since the Prospectus

Shape Partitioning is an approach to partitioning a space into 'inside' and 'outside' of a solid object (solid, meaning that the object exists in an original form without holes) based on sampled surface points in much the same manner of medial axis approximation algorithms and surface reconstruction algorithms. There are three challenges that must be addressed when evaluating a shape partitioning algorithm.

3.2 Challenges

There are three challenges with respect to the class of problems dealing with medial axis estimation, surface estimation and shape partitioning: holes (gaps in the sampled surface), narrow corners (areas where the surface cannot be considered smooth), and noise.

Holes come in two varieties: irregularities in surface scanning and areas of a shape not scanned at all. Irregularities in surface scanning can happen due to a number of reasons: a portion of the surface was obstructed by another portion of the surface, a portion of the surface was perpendicular to the scanner, or the material composition of the object at a portion presented unique challenges to the scanning process so that scanned points are sampled with some irregularities. Objects that are reflective are a challenge for laser scanners to interpret. If the scanning process is purely two dimensional (for example, if the object spins on a fixed platter that only moves clockwise and a laser scans top to bottom), then at least a small portion of the object will be perpendicular to the scanning process. Holes caused by areas of the shape not being scanned can happen for a number of reasons as well.

Perhaps the missing portion of the scan was an intentional omission: the object was part of a larger object and the sliced cross-section was not part of the original scanning process. There is a circumstance of the missing surface problem that appears frequently and is easily remedied: we make sure that points labeled as interior points also exist on the interior of the convex hull of the set of surface points. Any appearance of a point that is labeled as an interior point and exists outside of the convex hull is always mislabeled. Note that this does not fix all situations: one can easily imagine a situation where the missing surface exists inside the convex hull. This is an open problem.

Narrow corners represent the portions of an object's surface that cannot be considered smooth. This could mean that two distinct portions of a surface are joined by a ridge/valley or it could mean that a portion of the shape forms a cone in three dimensional data. Algorithms that attempt to determine a surface are confused at narrow corners as to outward facing direction of a surface point since multiple outward facing directions could be argued as equally correct. Algorithms that attempt to partition the object into interior and an exterior regions have a similar problem with narrow corners: two surface points in close proximity will either share a similar, smooth surface or they fail to share a surface and are divided by one of the types of narrow corners mentioned earlier.

Noise is introduced in the scanning process and to some degree is unavoidable. Noise can be introduced due to the lighting conditions of the scanning process or to vibrations in the structure supporting the scanned object. Even if variables in the scanning process are minimized, a scanned object should never be considered noise-free. Approaches to surface reconstruction and medial axis estimation will either incorporate the noise into the desired model or minimize it with a smoothing process. Either way, a good object segmentation algorithm should account for a degree of noise in the object.

3.3 Localized Principal Component Analysis

Our first task in object segmentation is understanding the surface of our object. An object can exhibit different features on different areas and those features have to be understood individually (or locally). One way to achieve this is by using a localized Principal Component Analysis. A localized PCA will be sufficient for estimating a vector that is tangent to the surface of our shape. We begin by finding the k nearest surface points to a particular surface point p, which will be called the neighborhood of p, or $Nbhd(p)$. We take the sum of the outer product vector of the difference between the surface point p and each neighborhood surface point x.

Data: S: surface, k: nearest neighbors

Result: P: principal unit vectors normal to surface S

for i *in* $1..\|S\|$ **do**

$Nbhd(S_i) \leftarrow k$-nearest neighbors to S_i;

$CV = \sum_{x \in Nbhd(S_i)} (S_i - S_x) \otimes (S_i - S_x)$;

$(\lambda, V) \leftarrow eig(CV)$, where λ is the eigenvalues of CV and V is the matrix of eigenvectors.;

$P_i \leftarrow$ column vector of V associated with the smallest element of λ;

end

Algorithm 1: Localized Principal Component Analysis

The localized PCA results in a vector that is normal to the estimated surface at every surface point in the shape, but does not indicate whether the aligned direction is 'inside' or 'outside' of the shape. Hoppe et al. proposed an algorithm for outer vector alignment [21]. Outer Vector Alignment is an attempt to align all vectors that are normal to the surface so that they are pointed 'outward' and away from the shape. We experimented with the Hoppe approach, which uses a Minimum Spanning Tree [1]. On most shapes, the conventional algorithms are sufficient for complete vector alignment, and thus we can easily determine

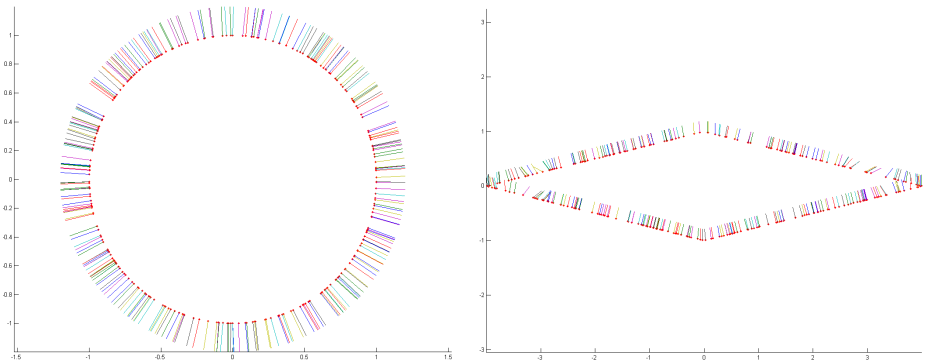
[1] Krager et al. created a randomized algorithm for the minimum spanning tree which has been shown to run in $O(E)$, where E is the number of edges in the graph [22].

whether a point in space is 'inside' or 'outside' based on the proximity of nearby vectors. Unfortunately, these approaches tend to fail at edge cases where a portion of a shape extends to a narrow point. In this context, aligned vectors tend to point all in the same direction, regardless of which side of the narrow point the shape point rests. Fig 3.1 shows the results of vector alignment of two shapes using Hoppe's Outer Vector Alignment algorithm.

Algorithm 2: Hoppe's Outer Vector Alignment

Data: S: surface, P: localized PCA unit vectors
Result: P is changed so that PCA unit vectors point outward from the shape
Compute the MST (minimal spanning tree) of the shape. The neighbors of a selected point p are $MST(p)$;
Select a point o in space that is 'outside' of the shape ($o = \sum_i \|S_i\|$, for example).;
Find point S_{min} in S so that S_{min} has the shortest distance to o.;
$cosinesimilarity \leftarrow \frac{(o-S_{min})\cdot P_{min}}{||o-S_{min}||\times||P_{min}||}$;
if $cosinesimilarity < 0$ **then**
    $P_{min} \leftarrow -P_{min}$;
end
initialize $seen(i) \leftarrow False$ for i in $1..\|S\|$;
$seen(min) \leftarrow True$;
initialize $queue \leftarrow \{min\}$;
while *queue is not empty* **do**
    $q \leftarrow$ *dequeue queue*;
    for *each neighbor in* $MST(q)$ **do**
        $cosinesimilarity \leftarrow \frac{P_{neighbor}\cdot P_q}{||P_{neighbor}||\times||P_q||}$;
        if $cosinesimilarity < 0$ **then**
            $P_{neighbor} \leftarrow -P_{neighbor}$;
        end
        if $seen(neighbor)$ *is False* **then**
            $seen(neighbor) \leftarrow True$;
            $queue \leftarrow queue \bigcup \{neighbor\}$;
        end
    end
end

We see that Outer Vector Alignment is useful, but can we avoid using it since we know of obvious points of failure? We believe that we can avoid it. It is possible to discern and prove that a point in space exists 'inside' our 'outside' as long as certain lemmas hold true and without performing the expensive step of Minimum Spanning Tree construction on



(a) Circle with Outer Vector Alignment (b) Diamond with Outer Vector Alignment

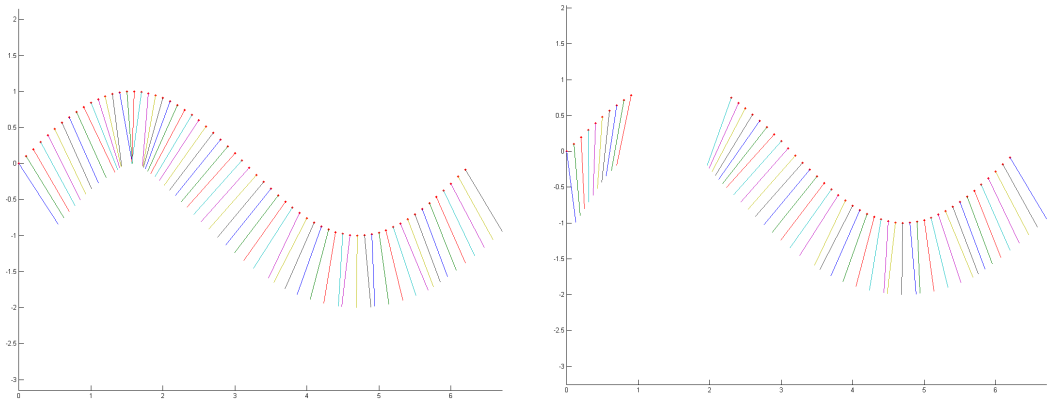
Figure 3.1: Two shapes depicted with Hoppe's Outer Vector Alignment algorithm ($k = 20$). Notice that when the surface is smooth (such as the circle shape on the left), all vectors are aligned and pointed outward from the center of the shape. When the surface contains narrow points (such as the diamond on the right), the algorithm is unable to maintain proper alignment throughout the shape.

the shape.

3.4 Surface Noise

We want to address the issue of localized PCA normals at narrow corners and on the edge of holes. On flat or smooth data, the normals are accurate and usable. On noisy data, these normals are less reliable but still usable. On narrow corners (even with minimal noise in the data), these normals are almost always unreliable and unusable. A demonstration of how normals shift in order to react to a hole is seen in Figure 3.2.

Our approach works best if the surface of our shapes is smooth. In areas where the surface is jagged, noisy, and randomly sampled, it is difficult to achieve the same jagged edge of the original image. To maintain accuracy, we elect to smooth the surface of the image where we can. We do this by computing a 'smoothness score' for each point along the surface which is computed using, which is defined by Equation 3.1. In this equation, $Nbhd(S_i)$ represents the k nearest surface points to a particular surface point S_i, just as it did in the algorithm for determining the localized PCA normal.



(a) Sine Wave (b) Sine Wave with a portion missing

Figure 3.2: The image on the left demonstrates the localized PCA normals at every point along one period of a sine wave. The image on the right represents the same sine wave, but points at $sin(x) > 0.8$ have been removed. Notice how the localized PCA normals shift now that a gap has formed.

$$smoothness(S, P, i) = \frac{1}{k} \sum_{x \in Nbhd(S_i)} \left| \frac{P_i \cdot P_x}{\|P_i\| \|P_x\|} \right| \tag{3.1}$$

This equation represents an average cosine similarity of a surface normal to the k nearest surface normals and will range from 0 (meaning this normal is orthogonal to every other normal) to 1 (meaning this normal and all k nearest neighbors to it are all in perfect alignment). To smooth the surface of an image, we drop from the surface any point that does not demonstrate a smoothness score of at least 0.5, representing a PCA normal that is more orthogonal to neighboring normals than it is in alignment.

We've found that this approach highlights the most difficult portions of the shape's surface quickly. Typically the approach highlights the possibility of holes in the shape. If the surface is mostly smooth and there are no significant holes, then there is no noise removal. Case in point, examine Figure 3.3 of an outline of the Linux Tux image that has been sampled 455 times (or one-fourth of the length of the perimeter). This is a mostly smooth shape (most of the smooth scores are greater than 0.9), so no points are removed based on our formulation. Contrast this to a sampled surface of the outline of the country of Norway in Figure 3.4 (sampled 1086 times, also one-fourth of the length of the perimeter). We see a

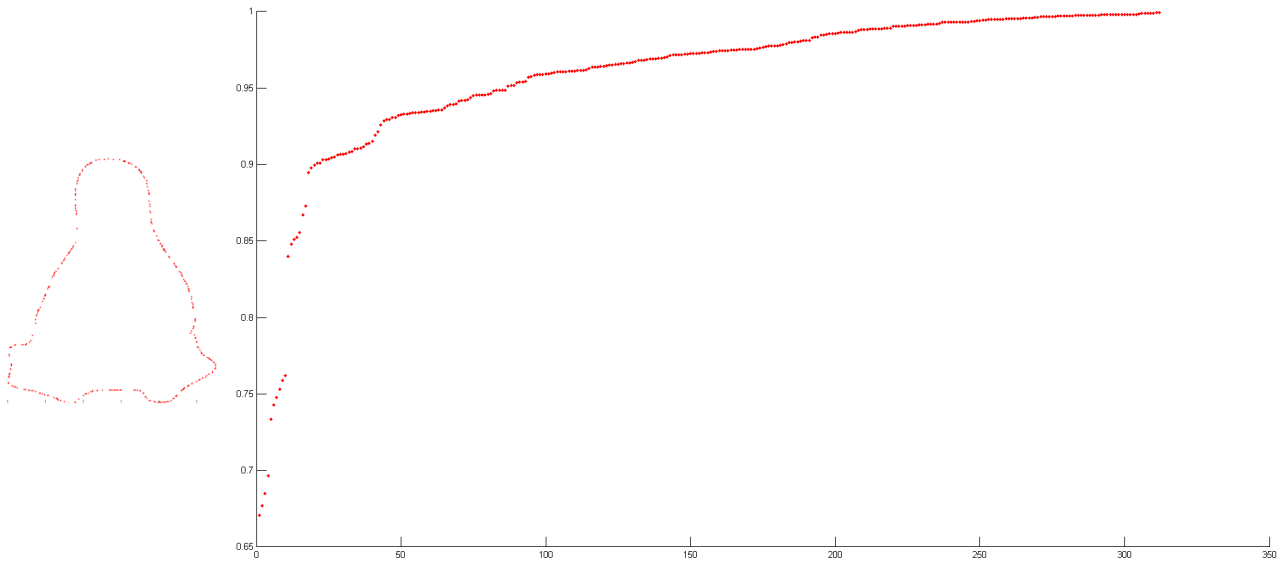


Figure 3.3: On the left is a sampled outline of the Tux image. On the right is the sorted smooth score of each surface point in the sampled image. Most of the points have a smooth score of 0.9 or higher. None have a smooth score less than 0.5.

surface far less smooth than the Tux image (less than half of the smooth scores are greater than 0.9), but that most of the normals are still usable. In this particular example shown, only 19 sampled surface points are removed.

By dropping points that do not meet our criteria, we are admitting a trade off in our algorithm: we would rather have larger holes in our surface than noisy surface normals. An analysis of this trade off is explored in the next chapter on testing our approach.

3.5 Turn Lemons into Lemonade

Let us discuss how we might further extend our algorithm if the Outer Vector Alignment algorithm was sufficient under all circumstances. We would select a point in space and the neighborhood of surface points that is closest to that selected point. We would then form a vector from each surface point to the selected point and compute the cosine similarity score between that vector and the surface point's estimates surface normal vector. If the majority of cosine similarity scores were positive values, we could reasonably assert that the unknown point was 'outside' (because the estimated surface normal points outward). With

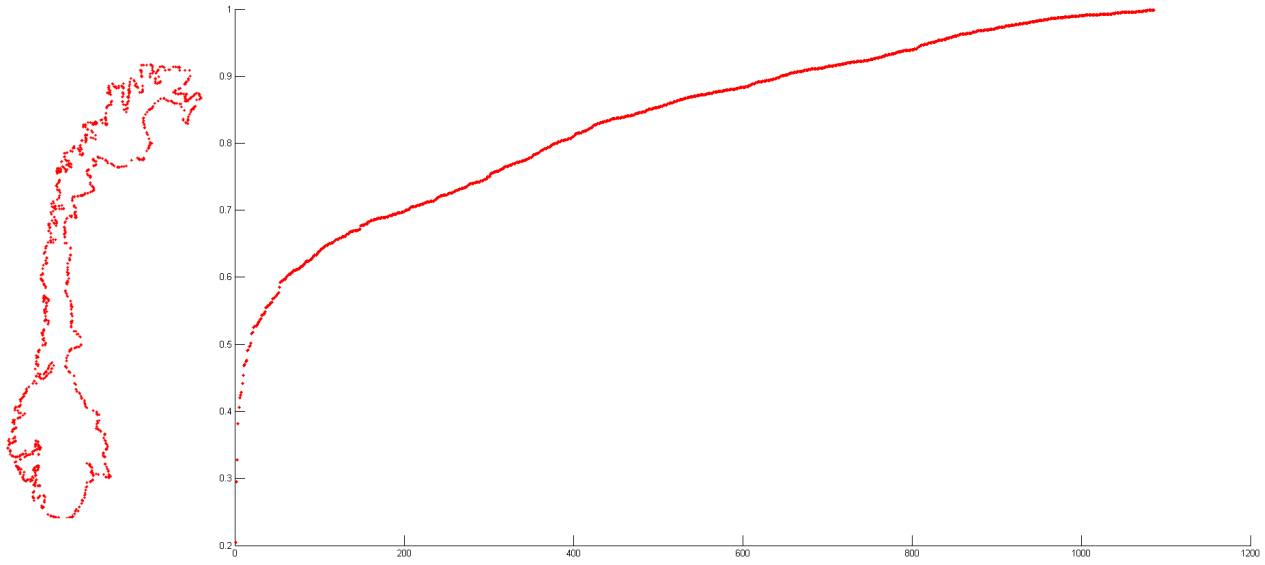


Figure 3.4: On the left is a sampled outline of the Norway image. On the right is the sorted smooth score of each surface point in the sampled image. Far fewer points have a smooth score of 0.9 or higher compared to the Tux image. Few have a smooth score less than 0.5 and will be removed by this approach.

any other result, we would assert that the points lies inside or on the surface.

This preceding paragraph explains how the theory would work if the Outer Vector Alignment algorithm worked, but we have already seen that it can't. We can still use a similar approach to determine whether a point in space is 'inside' or 'outside' without paying attention to the sign of the vector aligned, only trusting that each vector is aligned normal to the surface (which for now we assume to be true given that most of the surfaces in our testing objects are represented by smooth contours). While the particular orientation of the normal at a surface point cannot be trusted, the fact that it is normal to the surface can be trusted (as we discussed earlier, if the surface is smooth). Each normal vector extends outward from a point and every orthogonal vector to this normal vector forms a hyperplane that is aligned with the surface. We test for the presence of a surface by forming a line segment between points p and q in space. If we already know that point p is outside (or at least tentatively outside), we can assess the location of an unlabeled point q. Our points will be labeled in three classes: *outside* (those points that are far enough from the surface

that their status is not in doubt), *cusp* (or tentatively outside), and *inside*. Points that are labeled as *cusp* points are those thought to be outside. Points that are unlabeled and cannot be determined are left as such until the completion of the algorithm and are then converted to *inside*.

At the start of our algorithm, all points are considered unlabeled (as *unknown*) except for a set of candidate starting points which will be manually labeled as *outside* and added to our priority queue. At the start of each loop in our queue, the front of the queue is known as p and each neighboring unknown point is q. We define a neighboring point as the set of $(3^d) - 1$ nearest neighbor points to p, where d is the dimensionality of the data. We pick $(3^d) - 1$ neighbors since it coincides with the same 8-neighbor pattern (in 2 dimensions) or 26-neighbor pattern (in 3 dimensions) of a raster image. Each point in this set will be evaluated as q and we are testing if $\overline{pq}$ fails to cross any of the hyperplanes in the vicinity. Points that are labeled *cusp* and *outside* will be propagated. In the design decision to avoid propagation inside the shape, a bias emerges: a layer of points outside the surface are left as *unknown* due to the spacing of the grid formed around the shape. Before *unknown* points are relabeled as *inside*, we perform Minkowski Subtraction on the set of *unknown* points.

Detecting a crossing is as simple as evaluating the following equation and testing for 0, where S_i and P_i represent a location and orientation of a hyperplane, p and q represent the location of our points, and *sign* represents the sign function. A result of 0 indicates that the line segment $\overline{pq}$ crosses the hyperplane of surface point S_i.

$$sign\left(\frac{P_i \cdot (p - S_i)}{\|P_i\|\|(p - S_i)\|}\right) + sign\left(\frac{P_i \cdot (q - S_i)}{\|P_i\|\|(q - S_i)\|}\right) = 0 \tag{3.2}$$

We collect all the surface points of S within a threshold distance d to $\overline{pq}$. This set of points is known as the *bubble*. For each surface point in *bubble* (if any), we form a vector extending to each of the line segment endpoints and compute the sign of the cosine similarity of each vector to the normal vector associated with this surface point. If the sum of the two cosine similarity signs is zero, then this hyperplane intersects the line segment,

thus confirming that a surface edge crossing exists between p and q (See Eq. 3.2), but this is not enough information to conclude that the point q is inside the shape, so the point remains *unknown*. Depending on the scale of the grid field, evidence of multiple hyperplane crossings indicates that there could be multiple surface crossings (and an even number of surface crossings from an *outside* point is another *outside* point). Our approach does not attempt to answer the question of how many surface crossings exists on segment $\overline{pq}$, but merely that at least one exists.

This differing criteria for points outside of the shape allows us to propagate labels using different rules for similar circumstances. If there is a lack of evidence for a surface crossing between points p and q, we follow a set of rules to determine if and how the unknown q should be propagated. For example, consider a circumstance where a point p in space is outside the shape (not just labeled *outside*), and we wish to know if there are any hyperplane crossings between p and an unlabeled point q. First, we form our *bubble* of nearby surface points. In this example, we learn that there are no nearby surface points to consider. This could be the result of two circumstances: (1) point q is also outside the shape. -OR- (2) point q is inside the shape, but the scanning process failed to detect a sufficient coverage of surface points at this particular region of the shape (i.e. a hole has been encountered). To address this problem, we split our notion of outside into two categories: *outside* (for points that are sufficiently distant from the surface) and *cusp* (for points that are outside and in the vicinity of the surface). If our point p is *outside*, we can confidently label our unknown point q as *outside* as well. If our point p is labeled *cusp*, we cannot confidently label q as anything, so it is left unchanged. After a point's label is changed, the point is added to the priority queue: *cusp* points are added to the front of the queue and *outside* points are added to the back. This ensures that a 'cusp'-like point neighboring both a *cusp* point and an *outside* point will inherit the *cusp* label.

Propagation of the labels follow these rules, which are visually explained in Fig 3.5:

1. If point p is *outside* and there are no points in *bubble* between segment $\overline{pq}$, then q will

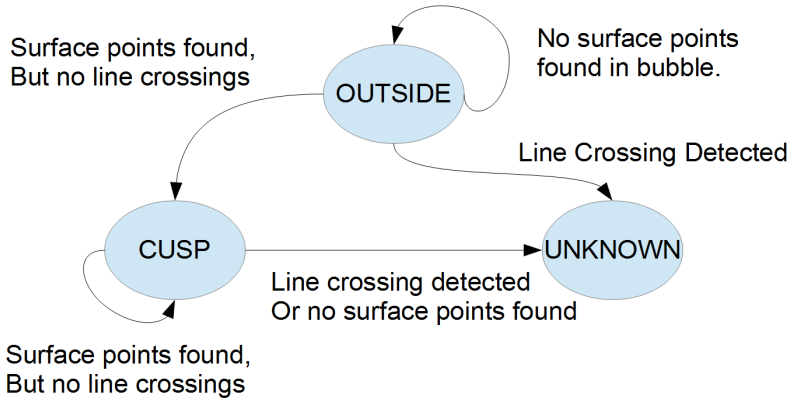


Figure 3.5: Diagram outlining the proposed rules for our propagation algorithm. Given two points, a labeled p and an unlabeled q, find the current label of p on this figure and apply the rules to determine the proper label for q. At the start of our propagation algorithm, the priority queue is initialized with the location of an point and assigned the label of *outside*.

be labeled *outside*.

2. If point p is *outside* and there is at least one point in *bubble* between segment $\overline{pq}$ and there is no evidence of a hyperplane crossing $\overline{pq}$, then q will be labeled *cusp*.

3. If point p is *outside* and there is at least one point in *bubble* between segment $\overline{pq}$ and there is some evidence of a hyperplane crossing $\overline{pq}$, then q's label will not be changed.

4. If point p is *cusp* and there are points in *bubble* between segment $\overline{pq}$ and no evidence of a hyperplane crossing, then q will be labeled *cusp*.

5. If point p is *cusp* and there are no points in *bubble* between segment $\overline{pq}$ (or there are points in *bubble* and there is some evidence of a hyperplane crossing $\overline{pq}$), then q's label will not be changed.

3.6 The Growing Algorithm

For the complete growing algorithm, see Algorithm 3.

Data: S: surface, δ: desired grid spacing, k: nearest neighbors for localized principal component analysis step, j: nearest grid points to a particular point, d: threshold distance from a line segment
Result: *labels*, which will be labeled as 'outside' or 'cusp' or 'unknown'
Compute the Localized Principal Component Analysis P on each point of the surface S using the k nearest neighbors.;
Create a N-dimensional grid of points U in a box surrounding the shape at interval δ. Make this grid sufficiently large to encompass the shape on all sides.;
initialize $label(u) \leftarrow unknown$ for each u in U;
Select a point o in one corner of grid U.;
$label(o) \leftarrow outside$;
initialize $queue \leftarrow \{o\}$;
while *queue is not empty* **do**
  $p \leftarrow dequeue\ queue$;
  initialize $neighbors \leftarrow The\ j\ elements\ of\ U\ nearest\ to\ p$;
  for q *in neighbors* **do**
    if $label(q) == unknown$ **then**
      initialize $count \leftarrow 0$;
      initialize $bubble \leftarrow elements\ of\ S\ less\ than\ d\ units\ from\ \overline{pq}$;
      for b *in bubble* **do**
        if $Orthogonal\ Hyperplane\ P(b)\ at\ S(b)\ crosses\ \overline{pq}$ **then**
          $count \leftarrow count + 1$;
        end
      end
      if $label(p) == outside$ **then**
        if $\|bubble\| == zero$ **then**
          $label(q) \leftarrow outside$;
          $queue \leftarrow queue \bigcup \{q\}$;
        else if $count == zero$ **then**
          $label(q) \leftarrow cusp$;
          $queue \leftarrow \{q\} \bigcup queue$;
        end
      end
      if $label(p) == cusp\ and\ count == zero\ and\ \|bubble\| > zero$ **then**
        $label(q) \leftarrow cusp$;
        $queue \leftarrow \{q\} \bigcup queue$;
      end
    end
  end
end

Algorithm 3: Growing Algorithm

3.7 On Selecting the Threshold Distance d

There is no provable silver-bullet selection of a threshold distance d that will generate a perfect shape for every sampled shape. One reason for this goes back to our discussion of holes. For example, consider a shape S in two dimensions sampled n times and its perimeter $perim$. The size of the average gap between two adjacent surface points within the sampled shape will be $perim/n$. If there are any concave paths within the shape that are smaller than $perim/n$ units wide, then there is no way to distinguish between a surface gap from a concave path and the algorithm will slip through the cracks. The result will be a grid that has few (if any) *inside* points at all. **Our algorithm assumes that there are gaps in the surface of a shape, but they are all smaller than the smallest concave path width within the shape.**

Care should be taken in selecting a value of d so that it is not too small. Care should also be taken in selecting a value of d that is not too large, but here is where the algorithm is more forgiving, but only slightly. When the value of d is large, the algorithm is accepting a large number of points in order to make a decision and the algorithm becomes unstable with respect to noise, resulting in concave portions of the shape to be ignored if the d threshold value is too large. For example, if d is infinity, all surface points would be considered when evaluating every line segment, regardless of the proximity to the line segment. This case reveals that the approach is highly sensitive to noise if d is large.

We took several tries at utilizing the value of d to create better results. We noticed that when d was large, there were more surface crossings detected when only one is needed. We found that our best results came when each true surface was detected when at least 5% of the surface points in the *bubble* confirmed a surface crossing. Based on this knowledge, we changed our criteria to confirm a line segment crossing through a surface that a minimum of $\lceil 0.05 \times \|bubble\| \rceil$ surface crossings were needed. Experiments with this threshold of minimum hyperplane crossings combined with the threshold distance d seemed to work well at first, except that there was still a noticeable break in results once the threshold distance d grew

to be too large. The approach ultimately fails to solve the problem of infinity mentioned earlier.

We tried again with a new approach. We sort the number of surface points within the *bubble* by distance and accept the closest n surface points. Only those surface points are used to inform our decision making process. We began testing images using both a threshold distance d and a point cut-off value of n closest points within our bubble (where 20 was used for n). The reason for using a cut-off value of the closest 20 surface points was so that we could maintain our rule of requiring $\lceil 0.05 \times \|bubble\| \rceil$ surface crossings, since this evaluates to 1 (and a minimum of 1 surface crossing was part of our original proposal). The results of this testing on a variety of 2D images was promising and we will publish some of those results in our testing section. When we tested the image in 3D, we discovered that the n parameter needed to be changed based on the image. We quickly realized that we traded one parameter that required constant tweaking for two parameters that need constant tweaking and the idea of the second parameter was dropped.

In summary, when d is too small, the algorithm will slip through holes in the surface, but when d is too large, narrow corners and other concave spaces of the shape are lost. Through much testing, we have yet to identify a method of estimating the threshold distance d. We tested this approach extensively in the next chapter, yet we did come up with a better approach.

3.8 Making the Threshold Distance d Adaptive

Let us return to our working assumption: There are gaps in the surface of a shape, but they are all smaller than the smallest concave path width within the shape. Every image containing a narrow, concave corner breaks this assumption. A narrow, concave corner represents a portion of the image with a concave path leading to a finite point and there is no way to enforce our assumption on every image. We must conclude that a portion of every narrow, concave corner will be partially filled if the threshold distance d is larger than

necessary for this particular concave corner.

Estimating a value for our threshold distance d would be trivial if the surface of the shape was organized. We simply measure the distance between two connected surface points, find the largest gap, divide by 2 and set that as our threshold distance d. While our approach makes no assumptions about surface point organization, we could estimate this value based on a cursory examination of convex hull information, but even this approach would require a lengthy discussion.

The optimal threshold distance d should be small, but no smaller than necessary. We can make the threshold distance d be adaptive to the contours of the shape by extending our algorithm. First, we make our threshold distance d be unnecessarily large. As described earlier, this will create a $CUSP$ layer around the shape that will result in many of the concave portions of the shape being ignored. Taking the output of the first pass of the algorithm, we use those $CUSP$ points as the initial seed points of another pass of the algorithm with a smaller threshold value d. None of the rules of our algorithm change on successive passes of our approach. Each time we make a pass with our algorithm, we monitor the number of *unknown* points, store the results, and reduce the value of d. We take advantage of the fact that if the d parameter is too small, there will be few (if any) remaining points in the image. If the number of *unknown* points drops too low (say 5% of the number of total grid points), then we realize that our d is now too small for this shape and we simply return the results of the next-to-last pass.

The preceding paragraph describes the principle of our approach. In practice, we do something slightly different. We maintain yet another grid D of integer values where each value in the grid is initialized to zero. This grid will store the threshold distance attempted at every point as a multiplier m of the distance between two neighboring grid points (a constant throughout the grid). We begin by initializing a set of border grid points surrounding the image with a large multiplier m (we use 20, which means we begin by looking for all surface points 20 grid points away from a line segment being tested) and then running

our algorithm. Those border points initialize the queue and the core algorithm begins. Each time the algorithm detects a new cusp point, it writes a value of $m-1$ to the D grid, where m is the value in D of the known half of the testing line segment. At the conclusion of the core algorithm, we monitor the number of grid points in D still assigned as 0. Should the algorithm need to continue, the set of points assigned to the two smallest non-zero grid points are supplied to the queue for the next run. Once the number of gird points in D still labeled 0 drops too low, we return the set of points assigned to the smallest two labels in D (this time, including 0) as our *unknown* set.

One interesting product of our D grid is that we are left with a crude grid revealing density information near the surface, where lower values indicate areas with a higher density of surface points. It also represents a progression of how the algorithm grows closer to the surface and eventually stops just short of entering the shape.

3.9 Discussion on Aliasing

There is a problem that we aren't able to solve with this approach alone: aliasing. Aliasing artifacts occur at our boundary conditions, especially when there are two narrow convex areas along a surface in close proximity. The growing algorithm works on the assumption that if a region of an image is inaccessible due to being blocked by all hyperplanes in the region, then it must be because of a surface. This is merely a working assumption and is easily fooled when there is a concave region between two or more convex narrow points. The hyperplanes on the outer walls of the two or more convex regions will intersect at a point distant and well outside the shape. Our adaptive threshold distance approach does make some progress towards solving this issue, but the success or failure ultimately depends on the granularity of the initial grid used.

3.10 Cleaning

Once the propagation phase algorithm is concluded, we perform Minkowski Subtraction on the set of *unknown* points. We create a new binary field of the same size and shape as our label field where each element is set to 1 if the corresponding point in the labeled field is *unknown* and set to 0 otherwise. Minkowski Subtraction is a binary erosion process that considers a vector mask A which is a 3^d-mask (where d is the dimensionality of the data) of all 1 values and a vector B, which is any given point in binary field. We then perform

$$\{A - B = \{\mathbf{a} - \mathbf{b} \,|\, \mathbf{a} \in A,\ \mathbf{b} \in B\}\}$$

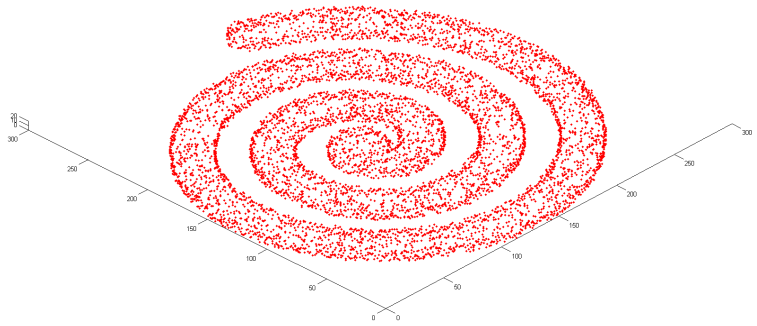
on the binary field at every point. This process removes noise from the surface of our recovered object. In our experiments, we discovered that the Minkowski Subtraction step failed to improve the accuracy of any shape that was tested and it will remove significant detail from portions of an image with narrow slivers of content on occasion. Still, we feel the step is necessary for cleaning the output. To counter this erosion of detail that comes from Minkowski Subtraction, we perform a Minkowski Addition immediately following. Our experiments show that this second step to the cleanup did improve the overall accuracy of the algorithm (since many of the good points removed in the subtraction step were added back into the shape during the addition step). Addition works in much the same way as subtraction, but we add the A mask rather than subtracting it from the image.

$$\{A + B = \{\mathbf{a} + \mathbf{b} \,|\, \mathbf{a} \in A,\ \mathbf{b} \in B\}\}$$

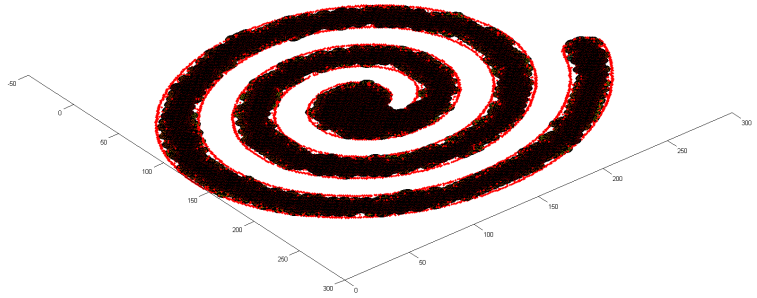
We did experiment with several approaches for improving the cleanup phase of the algorithm, but none were as reliable and defensible as this approach. The combination of Minkowski Subtraction and Minkowski Addition is equivalent to a one-neighbor requirement to justify being called 'inside' the shape. This combination of subtraction then addition, as we will see in Chapter 5 on testing 3D shapes, enhanced the results of our approach in each of our tests by cleaning aliasing lines.

3.11 Conclusions

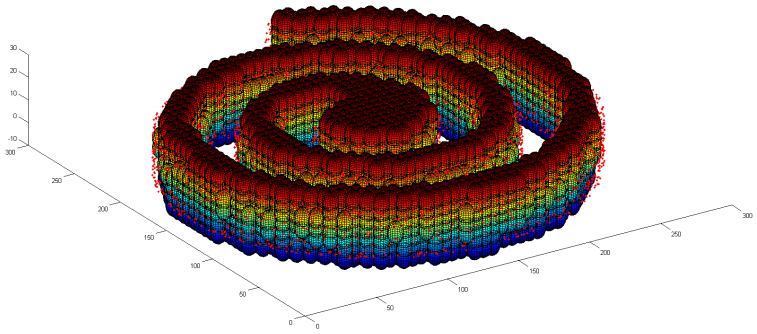
The above stated algorithm is sufficient for segmenting multiple objects in a scene represented by surface point scans of those objects (see Figure 3.7). The algorithm can ignore holes regardless of their location (whether they are facing the convex hull or not) up to the size specified by the d bubble size parameter all while hugging the surface of the shape to a distance of less than that of the spacing of the points in the original unknown depth field (See Figure 3.8). The algorithm performs well when provided with noisy surfaces (see Figure 3.9). The algorithm still has trouble on narrow crevices (when the d bubble size parameter is large, more smoothing happens at the location of a crevice and detail is lost) (see Figure 3.6). The algorithm performs well when navigating complex structures (see Figure 3.10). The speed of the algorithm is a factor that should be considered: since this is a growing algorithm, a dense unknown points field yields better detail at the cost of time. On a Quad-core Intel i7 3.07 GHz processor with 8 GB of RAM, an initial point field of 76,849 points took 11 minutes 38 seconds to segment. For our testing purposes, we've found that a point field of 30,000 points takes about 5 minutes to complete and yields sufficient results that can be used to evaluate the algorithm.



(a) Cinnamon Roll Point Cloud

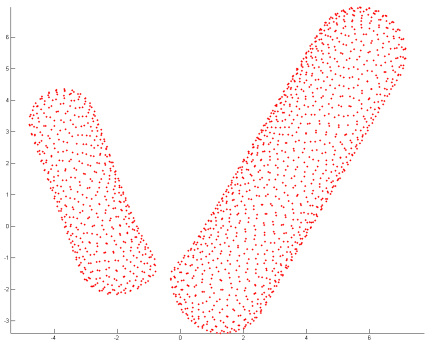


(b) Cinnamon Roll Overhead

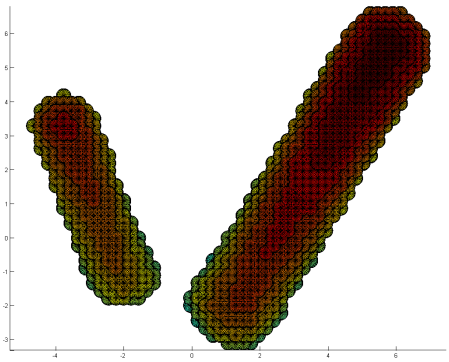


(c) Cinnamon Roll Side View

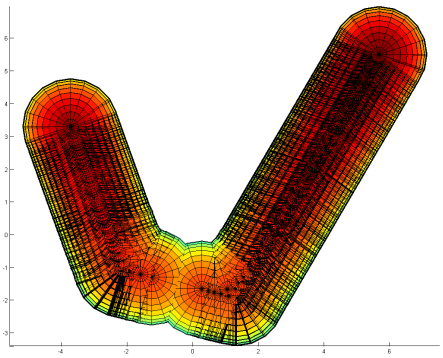
Figure 3.6: The Cinnamon Roll shape illustrates the strength of the growing algorithm in navigating narrow passages of a shape.



(a) Hot Dog Point Cloud

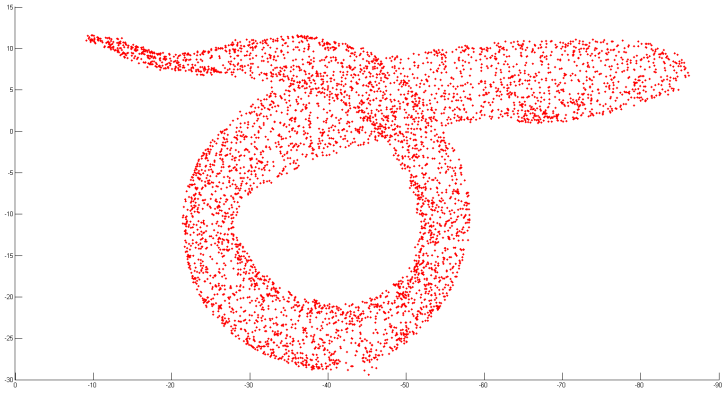


(b) Hot Dog (Our Approach)

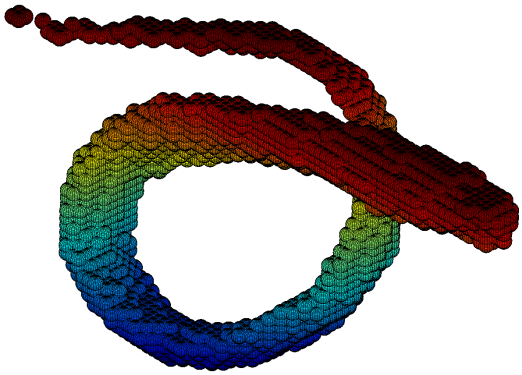


(c) Hot Dog (Power Crust)

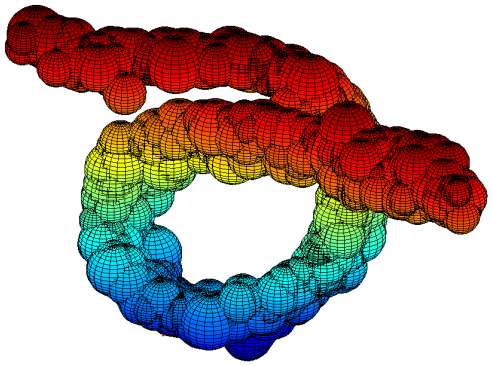
Figure 3.7: The Hot Dogs represent two objects that are close together but not touching. The challenge here is differentiating between the multiple shapes in the same scene.



(a) Eel Point Cloud

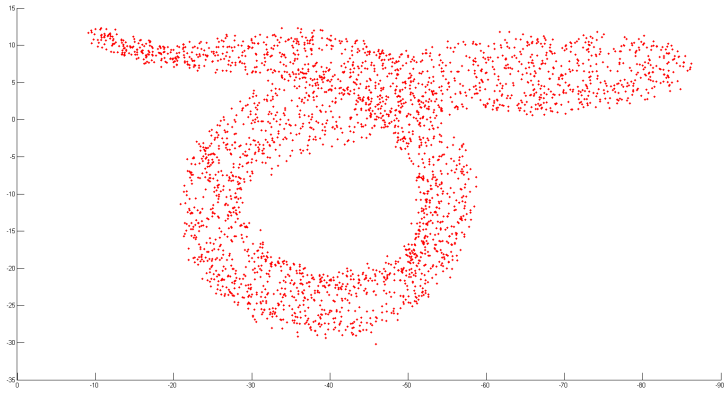


(b) Eel (Our Approach)

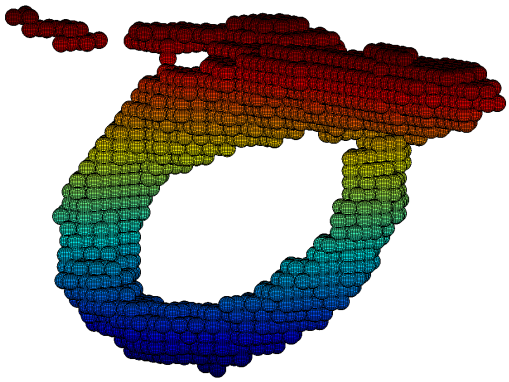


(c) Eel (Power Crust)

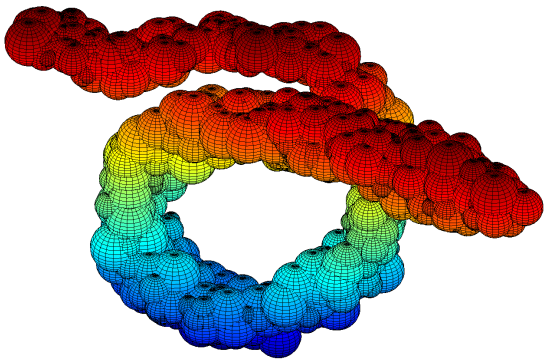
Figure 3.8: The inner surface of the Eel's loop is less sparse with surface points than outer surface. The tail of the eel is narrow, cloudy, and it is difficult to segment properly. Our approach almost creates a single, complete shape. The Power Crust requires a single complete shape, and that requirement leads to a new set of issues.



(a) Eel With Noise Point Cloud

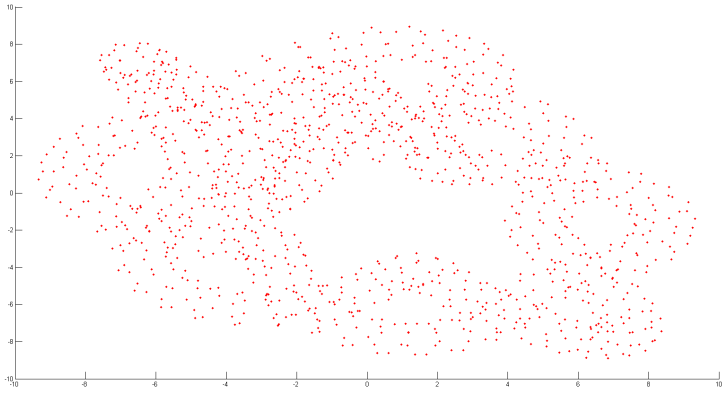


(b) Eel With Noise (Our Approach)

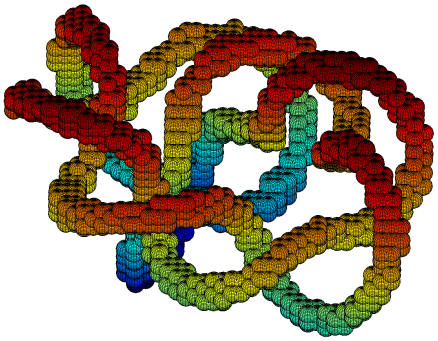


(c) Eel With Noise (Power Crust)

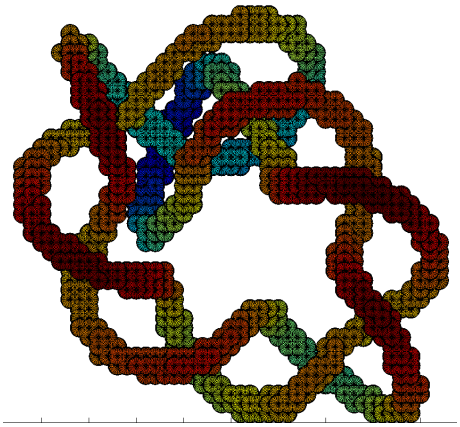
Figure 3.9: The Eel with noise makes this an even greater challenge. Every surface point in the eel had $\{-1..1\}$ noise applied. For both algorithms, the result is about the same as without noise. This demonstrates that our approach is robust to noise.



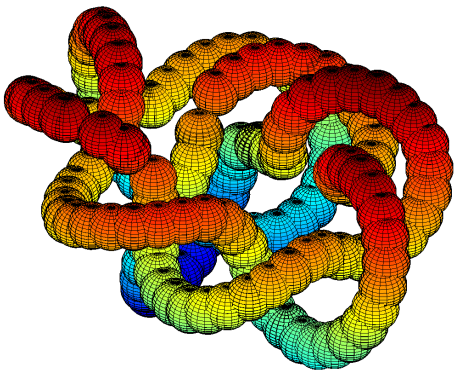
(a) Knot Point Cloud



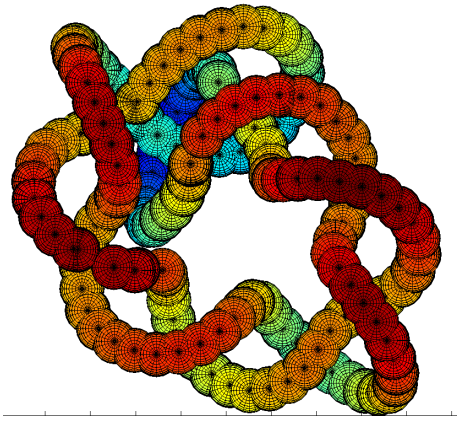
(b) Knot (Our Approach), Angle 1



(c) Knot (Our Approach), Angle 2



(d) Knot (Power Crust), Angle 1



(e) Knot (Power Crust), Angle 2

Figure 3.10: The knot is a challenging shape because it is a single structure that weaves around itself. Holes on the object can be related to the object's surface or it could be a natural hole formed by the weave pattern. Both algorithms perform well on the knot.

CHAPTER 4

TESTING IMAGE SEGMENTATION IN TWO DIMENSIONS

For our algorithm in two dimensions we use 10 test cases, each with a ground truth for which we can compare our accuracy.

- Evenly Sampled Unit Circle
- Randomly Sampled Unit Circle
- Randomly Sampled Star based on Formula - Figure 4.7
- Spiral Image - Figure 4.1(a)
- Tux Image - Figure 4.1(b)
- Puppy Image - Figure 4.1(c)
- Rooster Image - Figure 4.1(d)
- Norway Image - Figure 4.1(f)
- Splat Image - Figure 4.1(g)

To create the ground truth of each function or image, we overlay a grid in the vicinity of the shape and set a value of 1 to each point that should be correctly labeled as *inside* and a value of 0 otherwise. To measure the accuracy of the algorithm, we use Tanimoto's similarity score:

$$T_s(R,T) = \frac{\sum_i (R_i \wedge T_i)}{\sum_i (R_i \vee T_i)},$$

where R represents the vector results of the algorithm (using 1 as *inside* and 0 otherwise) and T being the ground truth vector. Tanimoto's similarity score was selected because it provides an equal penalty when our approach incorrectly labels too many incorrect *outside* points as *inside* as well as *inside* points as *outside*. Should our results perfectly match the ground truth, the Tanimoto similarity score will be equal to 1. We consider a good result to be a value greater than 0.7 and a great result to be a similarity of 0.9 and higher.

We set out to demonstrate that our approach will successfully recreate the intended shape based on a random sampling of surface points of that shape with more than 70% accuracy provided that at least 25% of the surface is available for analysis. Of course, more surface points will produce better results. Across all of our testing, accuracy based on reconstructing a surface with 25% of the surface ranged from 56% to 97% accurate. There were two approaches to our algorithm that we wish to test. The first was an early verion of our algorithm described in the previous chapter that we call the two-parameter-single-pass algorithm. The second version of the algorithm (and the version which we ultimately wish to defend) is the zero-parameter-multipass algorithm.

Our testing procedure for the two-parameter-single-pass algorithm is as follows.

- An image or function has its ground truth assessed and then it is randomly sampled.
- We create a grid of points that is 1.1 times larger than the dimensions of the image or function being analyzed.
- We select a distance threshold d based on a multiplier m of the distance between two grid points. The second parameter for each algorithm (representing the k nearest surface points used in each bubble) will be set to 20.
- We test each image for each value of m desired using the two-paramter-single-pass algorithm.

- We record the accuracy based on ground truth using the Tanimoto's similarity score.

Our testing procedure for the zero-parameter-multipass algorithm is as follows.

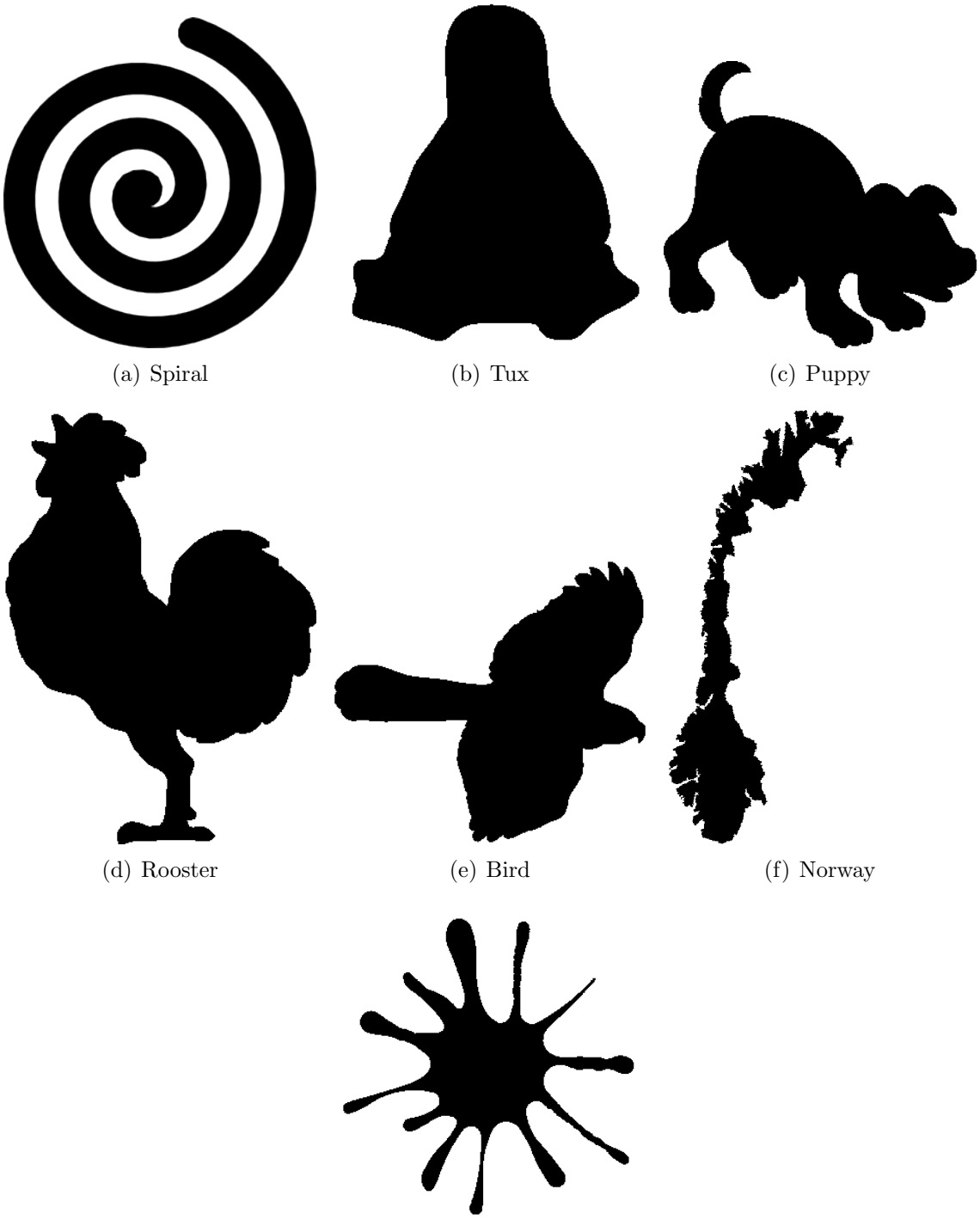
- An image or function has its ground truth assessed and then it is randomly sampled.
- We create a grid of points that is 1.1 times larger than the dimensions of the image or function being analyzed.
- We test each image using the zero-parameter-multipass approach.
- We record the accuracy based on ground truth using the Tanimoto's similarity score.

4.1 Noise Removal Testing

We wish to test the accuracy of our procedure for removing 'noisy' surface points compared to no removal at all. To perform this test, we ran 10 random samplings of each of our 7 images using our zero-parameter-multipass approach with no noise removal and the same approach with noise removal. We then assess the overall outcome. Table 4.1 shows our results where each of the 10 runs for each image and approach were averaged.

We find that in most cases there is little difference between our approach to noise removal and accepting every point as-is. We do see differences in at least two areas. The first is that sometimes the noise removal approach reduces the accuracy of an image result. Our noise removal approach removes surface points that we determine are of too poor quality for the algorithm, but does nothing to put something back in the place were a 'bad' surface point once existed. Since the noise removal approach has less to work with, it must compensate by using a larger threshold distance in the algorithm. This is to be expected and we aren't terribly concerned since the drop in accuracy is small. We can see this demonstrated in our Puppy test runs.

Second, we do see the noise removal algorithm work as intended on occasion. In one of our experimental tests using the Spiral image, we see that removing surface points allowed



(a) Spiral (b) Tux (c) Puppy

(d) Rooster (e) Bird (f) Norway

(g) Splat

Figure 4.1: We selected 7 shapes for the next phase of the testing process.

the algorithm to reach a higher accuracy by navigating down the narrow concave path of the image. Out of 70 different image samples that were tested, we only found one for which we can claim a definitive success of our noise removal approach. A visual comparison of the same spiral with same parameters recreated without and with our noise removal approach can be seen in Figure 4.2. We conclude that noise removal is rarely necessary and does have drawbacks, but it is nice to have in the rare circumstance when the approach demonstrates the intended benefit.

4.2 A Perfect Circle

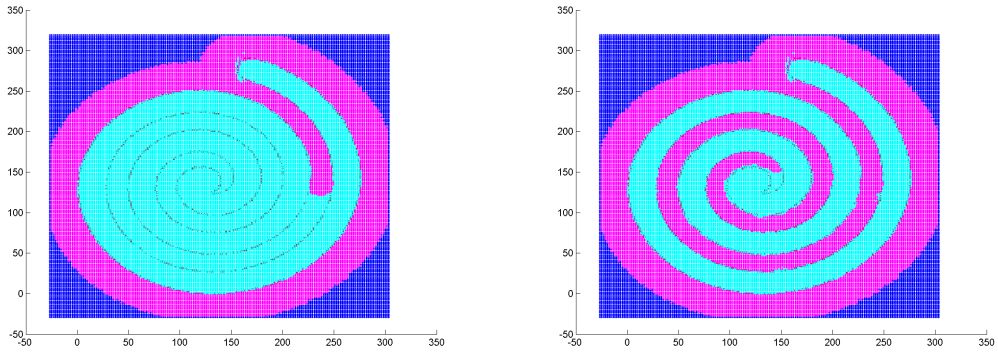
Our initial test is a shape in two dimensions: a unit circle sampled 100 evenly-spaced times. This contrived test case represents the easiest possible shape to evaluate and assess the accuracy. Because this example is contrived, we know that each point is exactly $\pi/50$ arc length units apart (in a real world example, we won't know this information). This example is free from any concave portions, free from noise, and all of the principal component vectors will be perfectly orthogonal to the surface at every surface point. When testing using the two-parameter-single-pass approach, the algorithm will be considering any surface points that are among the 20 closest surface points within d units away from a testing line segment. Since we only need 1 hyperplane crossing to justify saying that a surface crossing exists between two points, we will always satisfy this condition once the growing algorithm approaches a midpoint between two surface points (in this case is $sin(\pi/100) \approx \pi/100 \approx 0.03141$ units from a surface point). Should our threshold distance d be smaller than this value, our approach will weave between surface points because the algorithm believes it is following a concave path and will always fail to recreate a shape. None of this is an issue for the zero-parameter-multipass approach, which recreates the shape near perfectly (Figure 4.4).

4.2.1 Initializing the unlabeled grid

Throughout each of these tests, we define a grid of unlabeled points surrounding the shape by using a bounding box that is a factor of 1.1 times each dimensional length. The

Table 4.1: We tested our noise removal algorithm using 10 randomly sampled versions of our 7 testing images by comparing our original approach (designated as 'No Noise Removal') and our approach (designated as 'With Noise Removal') and averaged the accuracy of each group of 10 runs. There are two take-aways from this data. (1) Our noise removal approach doesn't have a strong impact either for or against the overall accuracy of the algorithm in the majority of cases. (2) The algorithm will work as intended, where we see that our noise removal approach helped to recreate the Spiral shape with a greater accuracy than no noise removal at all.

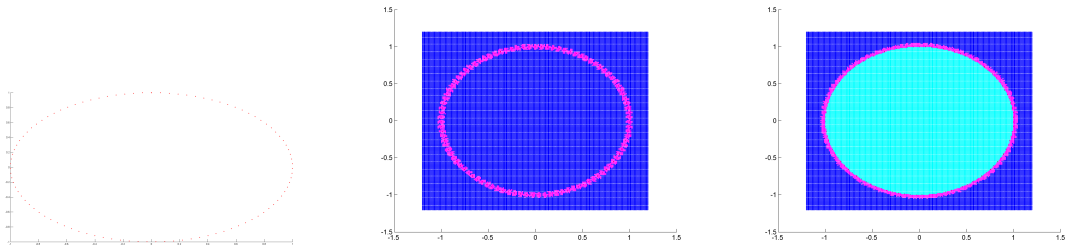
Image	No Noise Removal	With Noise Removal
Tux	0.95	0.95
Rooster	0.91	0.91
Spiral	0.83	**0.90**
Puppy	**0.90**	0.88
Bird	0.87	0.87
Norway	0.81	0.81
Splat	**0.77**	0.76



(a) No Noise Removal

(b) With Noise Removal

Figure 4.2: In our testing of noise removal, we examined 7 images each sampled 10 times randomly. In this particular run, we see the results of the algorithm with no noise removal achieve an accuracy of 0.70 on the left and with noise removal achieve an accuracy of 0.94 on the right. Both images are generated using the zero-parameter-multipass approach. This comparison represents an extreme case. Most of the time there is little difference in accuracy between the noise removal and approach and the no noise removal methods.



(a) Input Shape (b) $m = 2, d = 0.0256$ (c) $m = 3, d = 0.0384$

Figure 4.3: The first image represents an input shape: 100 perfectly spaced points along the edge of a unit circle. We use a 168-by-168 grid overlaid on the input shape, each 0.0128 units apart. We test our algorithm twice with a threshold distance d equal to $0.0128 * m$, where $m = 2$ in the first test (the algorithm fails) and $m = 3$ in the second test (the algorithm passes). The cyan color represents points labeled as *inside*, purple represents *cusp* and blue represents *outside*.

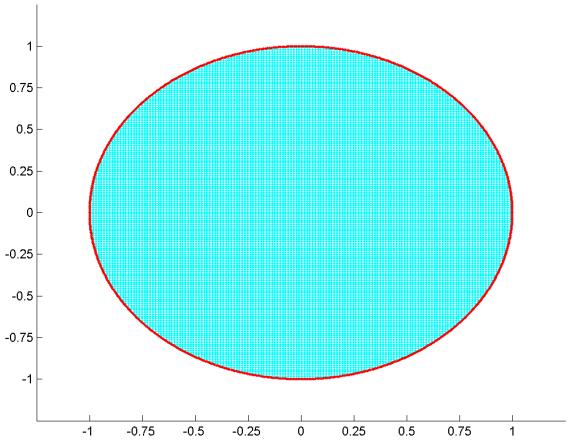


Figure 4.4: When using the zero-parameter-multipass approach on the perfect circle, we recreate the original shape perfectly.

algorithm for creating the initial unlabeled grid takes a value estimating the total number of desired points in the grid and then uses that information to create a grid of perfectly spaced points that is as close to the desired number as possible within the image field. The grid algorithm will then return the set of grid points as wells as G_Δ, which is the distance between any two neighboring grid points along one dimension. For each of these tests, we refer to the threshold distance d as m, which is a number of neighboring grid points from a line segment. $m = \frac{d}{G_\Delta}$. For each of these image tests, except where noted, we provide the grid creation algorithm a desired value of 25,000 grid points. The true number of grid points used for each image will vary.

At the start of each run of an algorithm, we select a grid of evenly spaced points to be placed in the vicinity of the shape. In this particular set of test runs, the grid was comprised of a 168 by 168 grid of 28,224 points, each 0.0128 units apart, creating a dense grid by which to test our algorithm. We performed 2 tests of the algorithm on this shape, where the threshold distance d was selected to be $0.0128 * m$, where m is an integer score of 2 and 3. Under these constraints, we should expect the algorithm to fail when $m = 2$ (thus $d = 0.0256$) and successfully recreate the shape when $m = 3$ (thus $d = 0.0384$). See Figure 4.3 for a comparison of these results.

We know that when d is less than optimal, we will fail to recreate our shape. We now examine what happens when d is equal to or greater than an optimal value. Our goal is for the algorithm to correctly label inside points as *inside* regardless of the value of the threshold distance d (provided that it is sufficiently large). It should be noted that for purposes of two dimensional testing we ignored convex hull information because we want to see exactly how the algorithm reacts. A grid point is labeled as *inside* if there exists a hyperplane crossing among the 20 nearest surface points that also passes between this point and each of its *outside* or *cusp* neighboring grid points. A grid point can be labeled as *inside* when the grid point exists outside of the convex hull up to d units away from the surface of the original shape. Within our contrived example, we can predict that a layer of points will be

labeled as *inside* if they are within $(\sqrt{1+d^2}) - 1$ units[1] from the surface of the shape.

4.3 A (Not So) Perfect Circle

We continue testing on a more difficult shape: a unit circle sampled 100 times randomly. As expected, the mean arc length between any two points will be $\pi/50$ units. Our algorithm depends on the distance threshold d being larger than $\frac{1}{2}$ of the distance between the largest gap between two coplaner surface points. Before performing this experiment, we simulated 100,000 unit circles randomly sampled 100 times and found that the largest undivided arc length between any two surface points across all sampled unit circles was 1.075 radian arc lengths (approximately 17.11 times greater than the average arc length). However, we found through similar simulations that the 95th percentile of the largest undivided arc lengths of 100,000 unit circles sampled 100 times was 0.458 radians. Based on this information, we can estimate that a threshold distance d of at least 0.23 units will successfully recreate the unit circle for 95% of possible 100 point sampled unit circles.

4.4 A Star

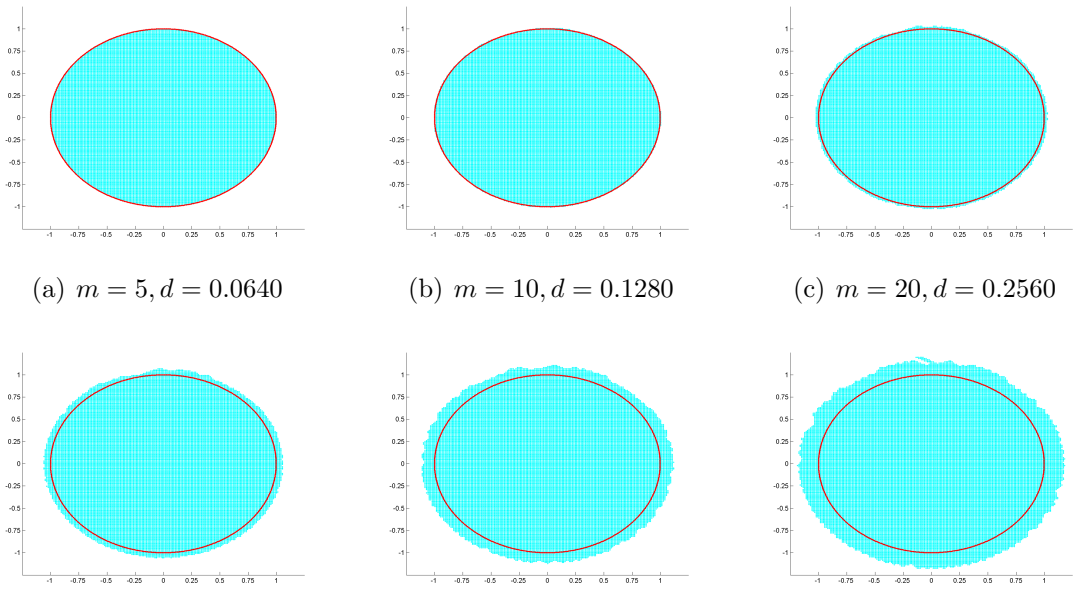
We created several test example beyond that of a simple unit circle. The following function is that of a triangle wave pattern mapped to a polar coordinate frame.

$$r(\varphi) = 1 + |(\varphi \mod (2\pi)/tips) - (\pi/tips)|,$$

where *tips* represents the number of tips on the stars (in our example sets, we use a 10 point star). We sampled this function 500 times where $\varphi \leftarrow [0, 2\pi)$ and then converted these values to their Cartesian equivalents in order to form a star pattern.

As with the previous examples, this is a specialized case of the unit circle examples, but this time with some fairly narrow corners by which we can test the corner following

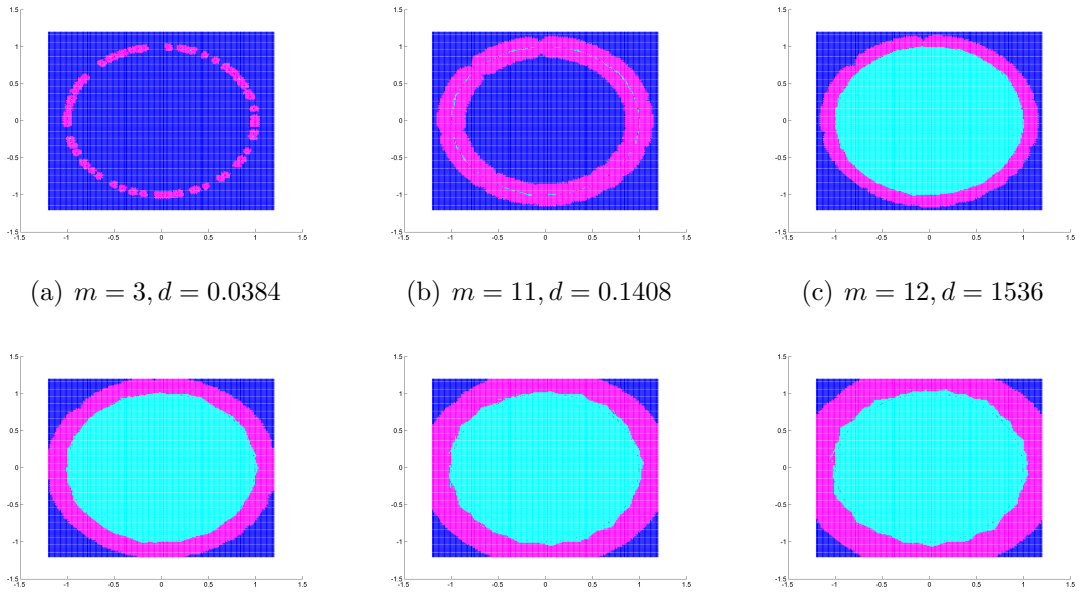
[1]We arrived at this formula by applying the following: A point in space will be labeled as inside if it resides within d units of a surface tangent. Since this shape is a unit circle with a radius of 1, the layer can be derived by subtracting the radius from the hypotenuse line of the formed triangle, or $(\sqrt{1+d^2}) - 1$.



(a) $m = 5, d = 0.0640$ (b) $m = 10, d = 0.1280$ (c) $m = 20, d = 0.2560$

(d) $m = 30, d = 0.3840$ (e) $m = 40, d = 0.5120$ (f) $m = 50, d = 0.6400$

Figure 4.5: We continue to test our contrived 100 point unit circle with differing values of d using our two-parameter-single-pass approach, where $d = 0.0128 * m$. The ring in each image represents a complete unit circle (used for visualization purposes only). The cyan color represents the points labeled as *inside*. We can see that as d increases, the number of points labeled as *inside* that exist outside of the ring grows. The width of this layer beyond the unit circle is always equal to $(\sqrt{1 + d^2}) - 1$ units. When $d = 0.2560$, this extra layer is barely noticeable. The final image represents the outcome when $d = 0.64$, which is a threshold distance of almost $1/3^{rd}$ of the diameter of the unit circle, yet the layer of questionable points labeled as *inside* is contained within 0.19 units from the surface.



(a) $m = 3, d = 0.0384$ (b) $m = 11, d = 0.1408$ (c) $m = 12, d = 1536$

(d) $m = 17, d = 0.2176$ (e) $m = 25, d = 0.320$ (f) $m = 30, d = 0.384$

Figure 4.6: We test our dataset on a 100 point randomly sampled unit circle. Similar to our previous tests, we select a square grid to surround the shape that is 168 by 168 grid points each 0.0128 units apart. We test our approach on this dataset with different values of d, where $d = 0.0128 * m$. Through our simulations, we predict that a threshold distance d of 0.154 should be sufficient for roughly half of the possible random configurations of a 100 point randomly sampled unit circle. We demonstrate the effects of this algorithm when m equals 3, 11, 12, 17, 25, and 30. We see that for this example, the shape was recreated when $m = 12$, making $d = 0.1536$. As with the previous example, we see that the layer of points labeled as *inside* that are outside the bounds of the unit circle are limited to those points within $(\sqrt{1 + d^2}) - 1$ units from the circle.

aspects of our algorithm.

After evaluating several different values of the threshold distance d, we see that the algorithm recreates the original shape (compared to our ground truth) with a similarity score of 0.97 when the d is 10 times the distance between grid points. When the threshold distance d is 30 times the distance between grid points (an extreme case), the algorithm still successfully recreates the shape with a similarity score of 0.95, resulting in a 2.1% degradation of accuracy after the threshold distance d was tripled from the optimal value. The results can be seen in Figure 4.8. A closeup of the algorithm's effectiveness at tight corners is shown in Figure 4.9 where it can be seen that the algorithm does an effective job of holding close to the shape's surface at both narrow concave and convex portions.

When testing the star pattern using the zero-parameter-multipass algorithm, we varied the input shape by restricting the sampling of the shape to be from 50 to 500 surface points in increments of 50.

4.5 Testing the Multipass Approach

Our multipass approach will attempt to re-evaluate an image continuously until it believes that a good result is created. Internally, the algorithm starts with an m value of 20 and reduces this value by 1 until the number of internal points in the image drops below 1%. This algorithm will fail if the initial grid is so dense that a circumstance arises when there exists a path of more than 20 grid points from any given surface point. Across all of our testing on static images, we encountered this type of failure once on an image that was subject to a case of oversampling in one part of the image (which lead to undersampling in other parts). We demonstrate our best results in Figure 4.11. In the spirit of showing our approach at its weakest, we also demonstrate 6 of our worst results discovered in Figure 4.12.

Most images can be recreated with 90% similarity to ground truth with no issue. It's images like our Norway image that seem to give the algorithm problems. There are lots of concave patches along the Norwegian coastline and any time a portion of the image is

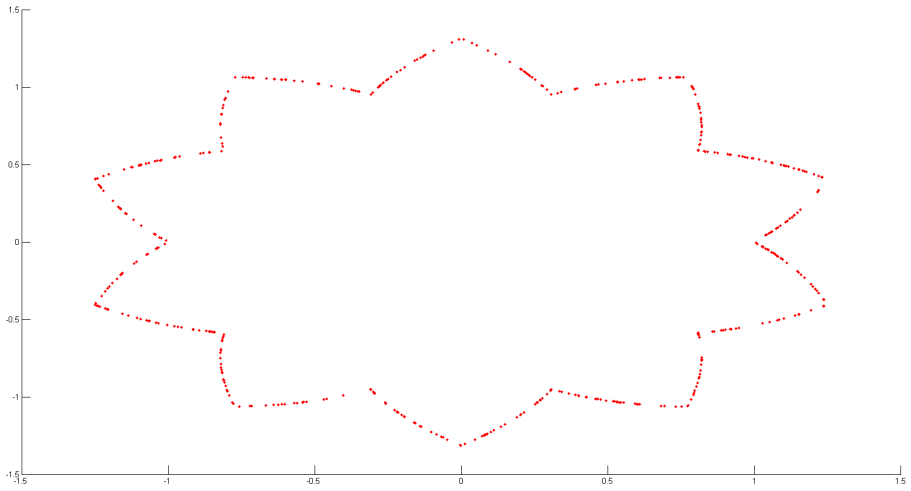
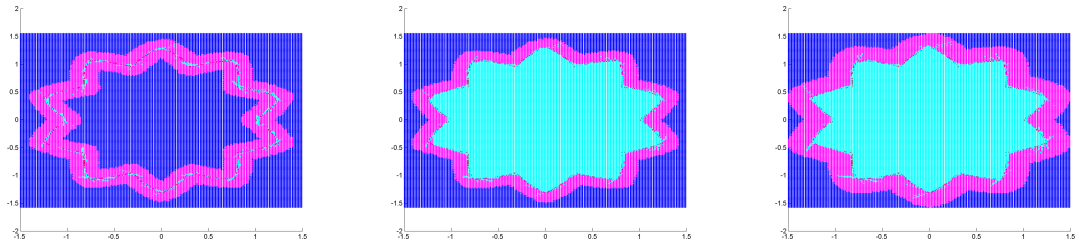
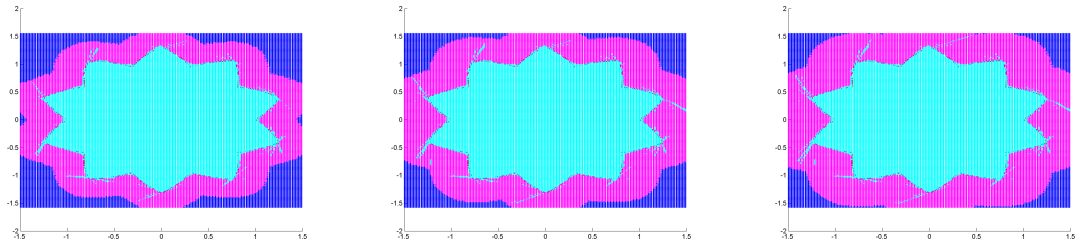


Figure 4.7: This 10 point star was created by sampling an equation 500 times. We will use it in the evaluation of our algorithm.



(a) $m = 9, T_s = 0.02$ (b) $m = 10, T_s = 0.97$ (c) $m = 15, T_s = 0.96$



(d) $m = 20, T_s = 0.96$ (e) $m = 25, T_s = 0.96$ (f) $m = 30, T_s = 0.95$

Figure 4.8: We test our dataset on a 500 point randomly sampled 10 point star using a 184 by 192 grid of 35,328 points, each 0.0163 units apart. The threshold distance d was selected based on a multiplier of the grid density, $m = d * 0.0163$. We tested the algorithm for values of m from 1 to 30 but only the values 9, 10, 15, 20, 25, and 30 are shown, along with the Tanimoto's similarity score compared with the ground truth. We see that for this example, the shape was recreated when $m = 10$ and holds a 97% similarity to ground truth. As the threshold distance d increases, we see the similarity degrade to 95% when $m = 30$.

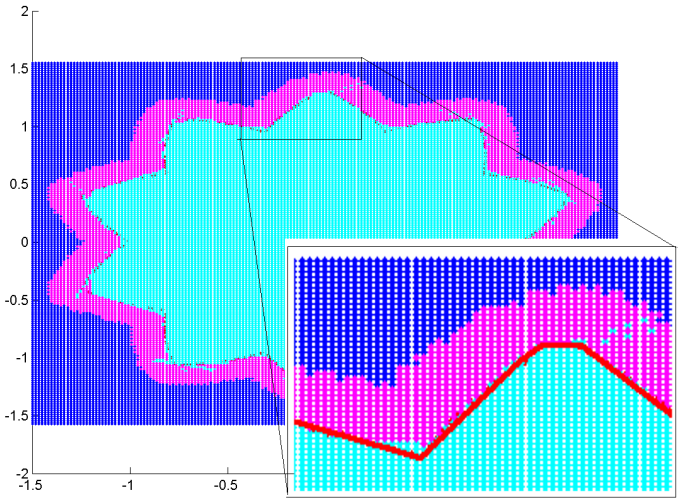
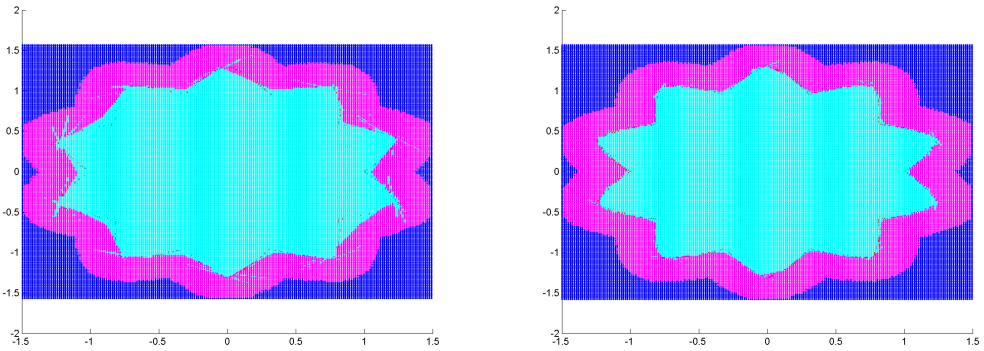


Figure 4.9: We highlight the ability of the algorithm to hug narrow corners in this image. This image is the same as the image in the previous figure with $m = 10$ with a similarity score with ground truth of 0.97. The detailed image shows the algorithm's ability to hug the surface of the shape while navigating narrow corners, both concave and convex. (A red line has been added along the surface of the shape.)



(a) $\|S\| = 300, T_s = 0.94$ (b) $\|S\| = 500, T_s = 0.98$

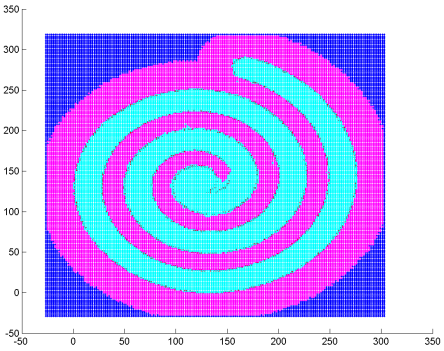
Figure 4.10: These are the products of our zero-parameter-multipass approach on two star formations, the first with 300 surface points ($T_s = 0.94$) and the second with 500 surface points ($T_s = 0.98$). When the number of surfaces points drops below 300, we failed to recreate this star shape.

undersampled we lose information about a concave patch. When we compare these 'best' images to their 'worst' counterparts, we gain insight as to how the algorithm works when presented with poor data. Simple images that consist of mostly smooth surfaces (such as our Bird image and our Tux image) are able to be recreated with 90% accuracy on the hardest of datasets. We do see some images cause the algorithm to stumble (such as our Puppy image and our Rooster image). These images contain narrow points which will cause problems if those same areas of the surface are undersampled. Our Spiral image fails to navigate down the difficult concave path, again due to undersampling (across all of our testing, this one sampling of the Spiral image failed to be recreated with 90% accuracy). Most surprising to us is that Norway only has a 2% difference between our best recreated image and our worst recreated image. We attribute this stability to the fact that there are so many concave patches that undersampling is going to happen somewhere on the image.

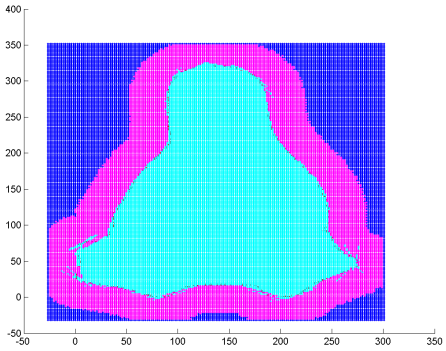
4.6 Spiral

The Spiral image was selected because it contains one long concave portion which (in our view) represents a difficult condition: in order to reconstruct the shape to any high accuracy, the algorithm must track the perimeter of the concave portion of the shape without a misstep into the shape itself. Unlike other images, this image's perimeter was randomly sampled 1,500 times (seen in Figure 4.13(a)), which is less than half of the image's full perimeter. We will examine the visual results and the similarity scores of a few of our test runs. We highlight three particular trial runs, one at $m = 6$ (the best run) in Figure 4.13(b), at $m = 10$ (where we start to see problems forming) in Figure 4.13(c), and at $m = 30$ (the worst run) in Figure 4.13(d).

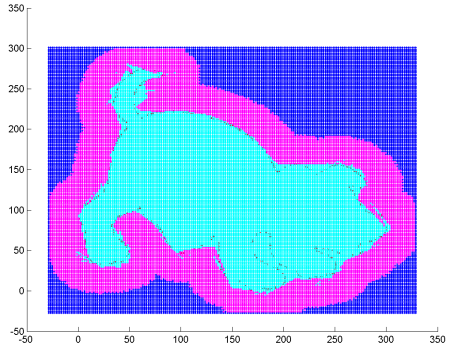
At $m = 6$, the similarity of this image compared to ground truth is 0.94. The algorithm has a difficult time tracking into the acute angle at the center of the spiral, but otherwise does a sufficient job of tracking the surface. At $m = 10$, our spiral image that was reconstructed with a similarity score of 0.92. With the distance threshold increased, we



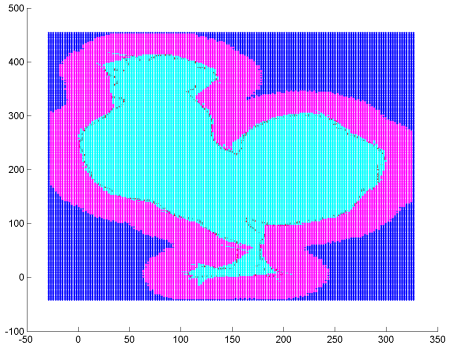
(a) Spiral, $T_s = 0.95$



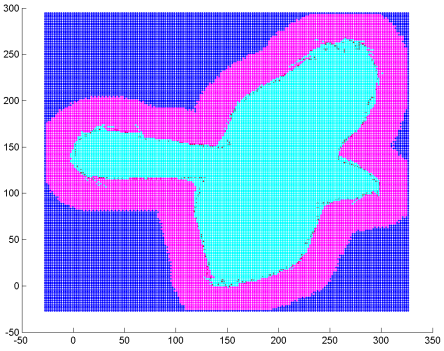
(b) Tux, $T_s = 0.97$



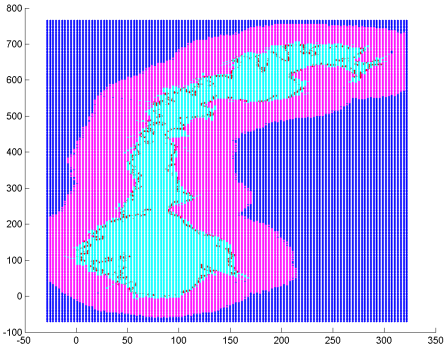
(c) Puppy, $T_s = 0.91$



(d) Rooster, $T_s = 0.93$

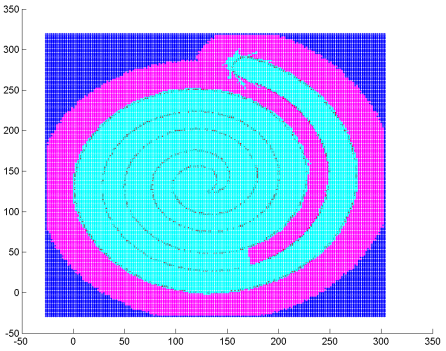


(e) Bird, $T_s = 0.95$

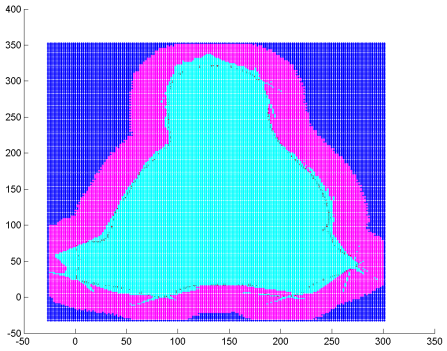


(f) Norway, $T_s = 0.82$

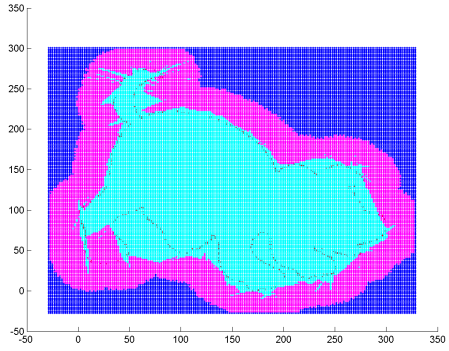
Figure 4.11: This represents some of our best accuracies of our approach.



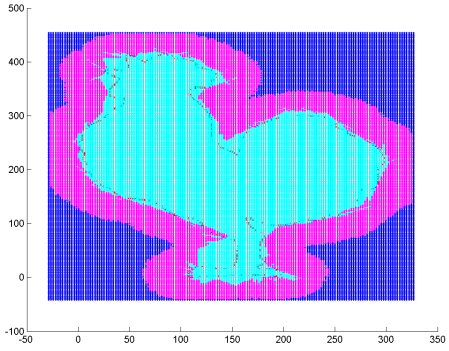
(a) Spiral, $T_s = 0.72$



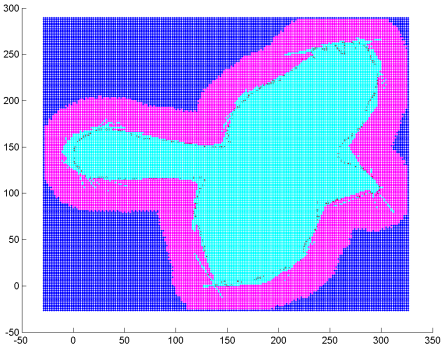
(b) Tux, $T_s = 0.92$



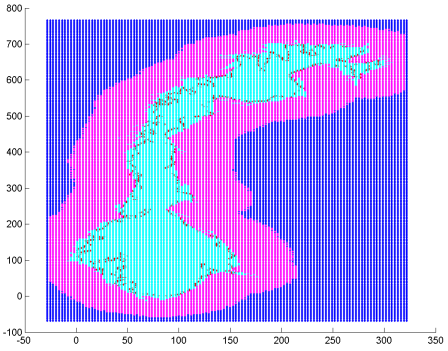
(c) Puppy, $T_s = 0.80$



(d) Rooster, $T_s = 0.86$



(e) Bird, $T_s = 0.91$



(f) Norway, $T_s = 0.80$

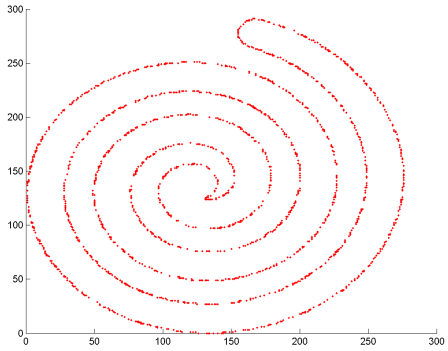
Figure 4.12: This represents our worst results across all of our testing.

see small artifacts form along the surface of the shape. The narrow cavity at the end of the spiral has not improved, but at least it does not appear worse than when $m = 6$. At $m = 30$, our spiral image that was reconstructed with similarity of 0.91. The distance threshold is purposely set to be larger than the width of the concave path, yet the algorithm still tracks the wall of the shape with little trouble. The accuracy of the algorithm has decreased only slightly when the parameter was set to a third of the current value.

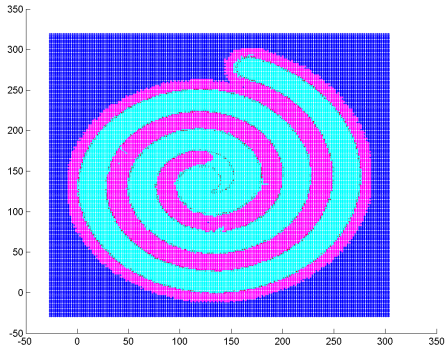
4.7 Puppy

The Puppy image represents an image that we believed would be simple for the algorithm since it consists of nearly all smooth surfaces, yet the algorithm produced poor overall results. We like it when we see similarity scores compared to ground truth be greater than 0.9, but that was not the case in this set of tests. We can see the input data set that the algorithm had to work with in Figure 4.14(a). The dataset consisted of 455 sampled surface points from the original image. The highest similarity score seen among tests for this image was 0.85 when $m = 14$ (Figure 4.14(b)). We notice that problems related to the structure of the image get worse when $m = 17$ and the similarity score is 0.83 (Figure 4.14(c)). Problems continue to get worse for the accuracy of our approach at the final test when $m = 30$ and the accuracy is 0.78. (Figure 4.14(d)).

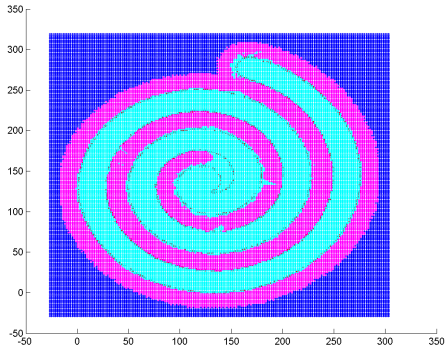
Our Puppy image that was sampled 455 times and reconstructed with our approach with various threshold distance parameters. We have selected 3 for inspection. At $m = 14$, the similarity of this image compared to ground truth is 0.85. At $m = 10$, our puppy image that was reconstructed with a similarity score of 0.92. With the distance threshold has increased, we see small artifacts form along the surface of the shape. The narrow cavity at the end of the puppy has not improved, but at least it does not appear worse than when $m = 6$. At $m = 30$, our puppy image that was reconstructed with similarity of 0.91. The distance threshold is purposely set to be larger than the width of the concave path, yet the algorithm still tracks the wall of the shape with little trouble. The accuracy of the algorithm



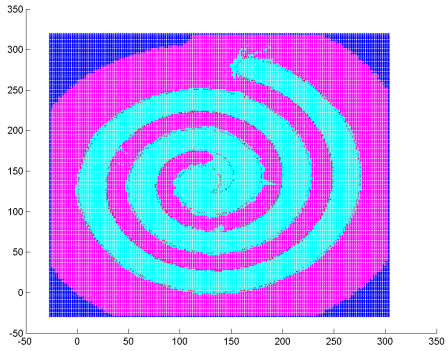
(a) Spiral Sampled



(b) $m = 6$,$T_s = 0.94$



(c) $m = 10$,$T_s = 0.92$



(d) $m = 30$,$T_s = 0.91$

Figure 4.13: Our Spiral image was sampled 1,500 times.

has decreased only slightly when the parameter was set to a third of the current value.

4.8 Tux

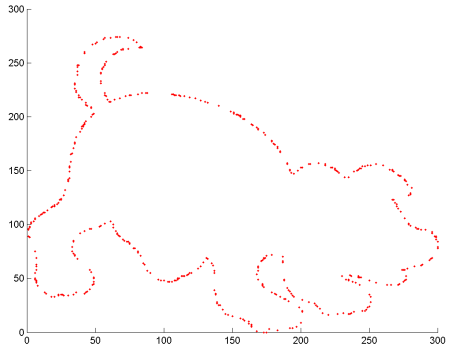
The Tux image was selected because it gives us a chance to show off the accuracy of our approach when conditions are optimal. The original sampled image is in Figure 4.15. Every surface in the Tux image has a smooth, gradual curve. This example gives us a chance to demonstrate that our approach can provide a high accuracy on some images. Even when the distance threshold parameter is large (the largest we tested was $m = 30$), we still see a high accuracy of 0.93 (Figure 4.17). Our best accuracy at 0.95 similarity appears when $m = 15$ (Figure 4.16).

4.9 Rooster

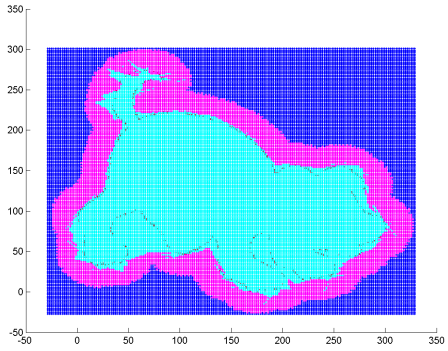
The Rooster represents one of the more challenging images that were tested, containing a variety of features that are thick and thin. The original sampled image is in Figure 4.18. In each of these images, notice the reconstruction surrounding the area of the toes. As the threshold distance parameter increases, significant artifacts appear. These artifacts represent points in space where two or more normals cross in space in a manner that prevents our growing algorithm to cross through. Our algorithm currently doesn't have a stable manner in which these areas can be addressed in the two-parameter-single-pass approach aside from the Minkowski subtraction (which only trims the shape's final appearance). As stated earlier, we do not perform the final step of Minkowski subtraction in our testing of two dimensional images so that we can illustrate the source of artifacts such as those seen in Figure 4.21 where the threshold distance factor of m is set to 30. Our best image with this approach can be seen in Figure 4.19.

4.10 Bird

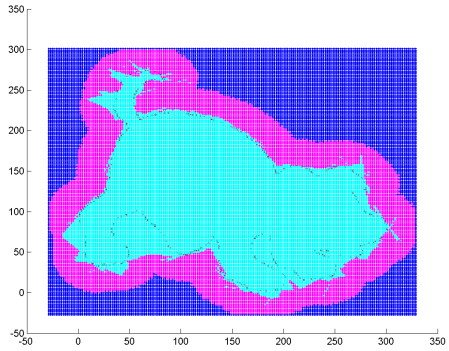
The Bird image represents another image where we demonstrate that the algorithm can produce great results regardless of the starting parameters if the initial dataset consists



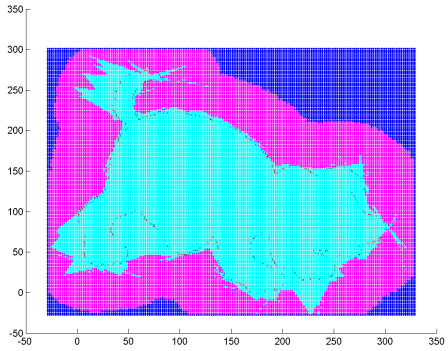
(a) Puppy Sampled



(b) $m = 14$,$T_s = 0.85$



(c) $m = 17$,$T_s = 0.83$



(d) $m = 30$,$T_s = 0.78$

Figure 4.14: Our Puppy image was sampled 455 times.

Figure 4.15: The original Tux image was randomly sampled to a set of 312 surface points.

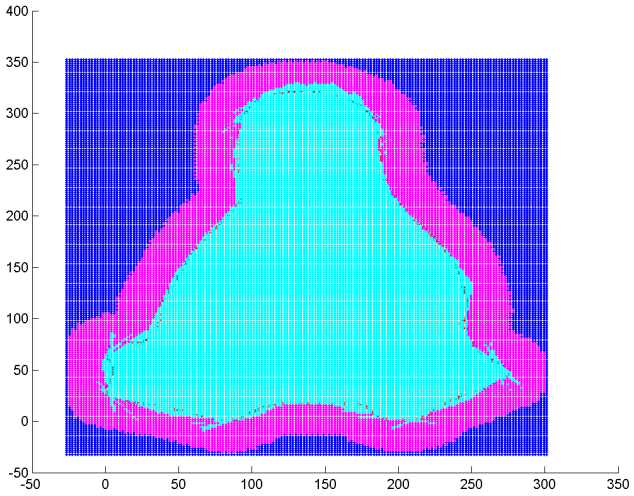


Figure 4.16: Our approach recreated the Tux image with a 0.95 accuracy. The m in this construction is set to 15.

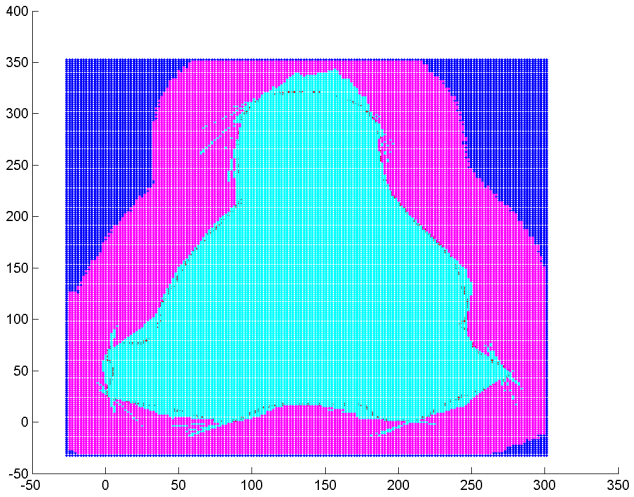


Figure 4.17: Even when we increase the m parameter in our algorithm to a high value, we still maintain high accuracy. With m set to 30, we still recreated the Tux image with a 0.93 accuracy.

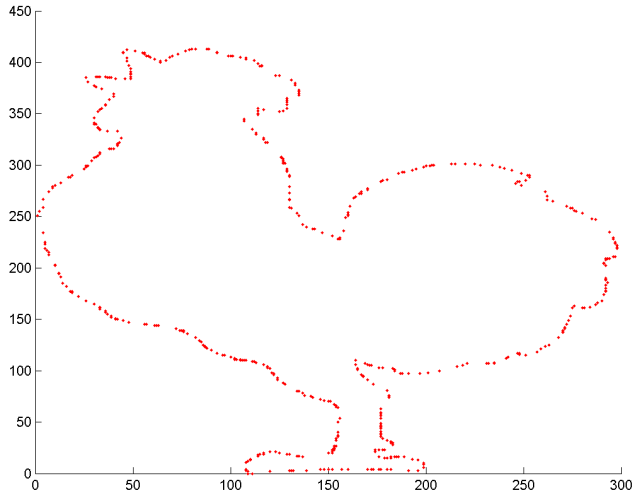


Figure 4.18: The original Rooster image was randomly sampled to a set of 499 surface points.

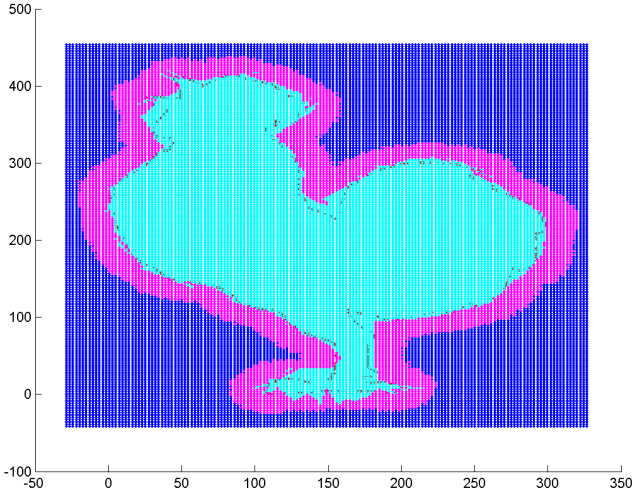


Figure 4.19: The Rooster image is reconstructed with 0.92 similarity to the original when $m = 10$.

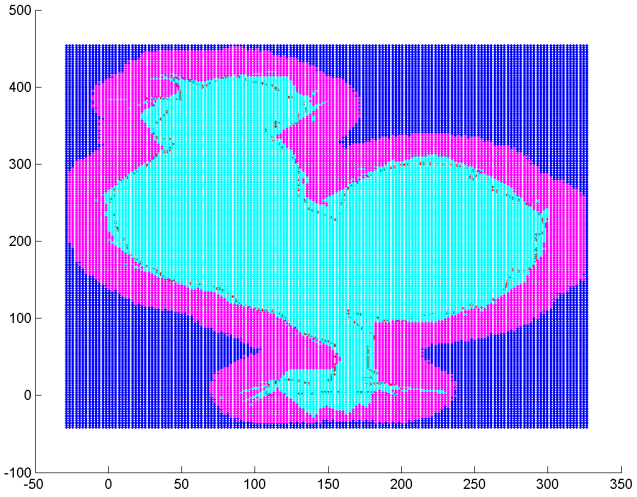


Figure 4.20: The Rooster image is reconstructed with 0.88 similarity to the original when $m = 16$.

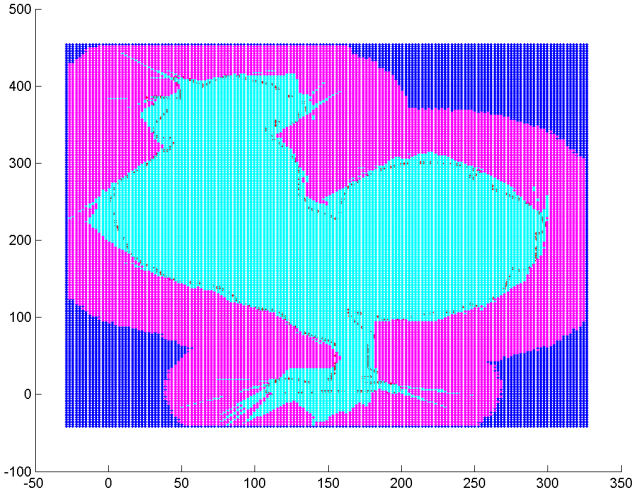


Figure 4.21: The Rooster image is reconstructed with 0.86 similarity to the original when $m = 30$.

of mostly curved surfaces. Our original sampled image can be seen in Figure 4.22. At our best when $m = 12$, we were able to recreate this image with 0.93 similarity (Figure 4.23). At the worst when $m = 30$, we still achieve a 0.90 similarity (Figure 4.24).

4.11 Norway

The Norway image represents our attempt at demonstrating how the algorithm handles recreating images of things found in nature. The coastline of Norway has few simple curves to the surface and each of those crevices along the surface represent a potential for error. Our sampled surface used for testing is seen in Figure 4.25. We found that this circumstance results in lower accuracy than our simple images (as to be expected), but we also found that the accuracy doesn't degrade as much as we would expect. Our best result came when $m = 4$ and results in a similarity of 0.88 (a good score in our opinion) as seen in Figure 4.26. As the threshold distance increases, we do see a noticeable drop in image similarity. At $m = 15$, the image similarity slips to 0.79 (see Figure 4.27. At $m = 30$, the reconstructed image has a 0.72 similarity score (see Figure 4.28). While we don't consider 0.72 to be great, we still think it gives an accurate depiction of the original image.

4.12 Splat

The Splat image represents our most difficult data set tested. The image consists of several long, narrow portions that are bound to compound errors if the algorithm fails to hug the surface properly. Our sampled input can be seen in Figure 4.29. Not only was it the hardest image in our library of testing images to recreate, it represented the image with the largest degradation of performance from 0.81 similarity to 0.56 similarity as the threshold distance increased from a factor of $m = 11$ (Figure 4.30) to $m = 15$ (Figure 4.31) to $m = 20$ (Figure 4.32). Most of the images that we used in our test used a shape that had an area of at least half of the image space. This is not true of the Norway and Spiral images: most of the original image for these two tests is whitespace. Our Tanimoto similarity score is more

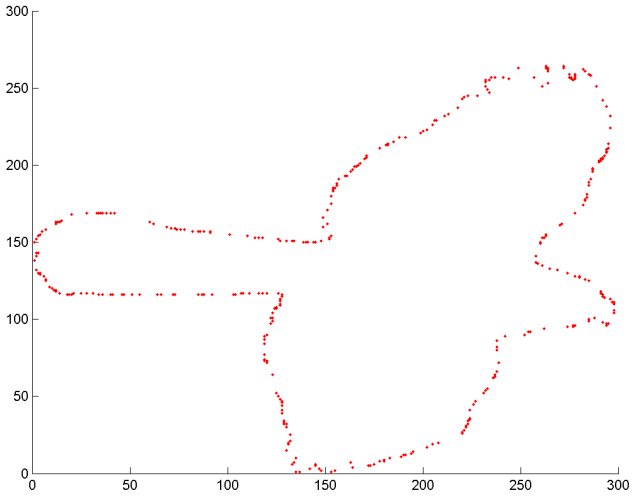


Figure 4.22: The original Bird image was randomly sampled to a set of 350 surface points.

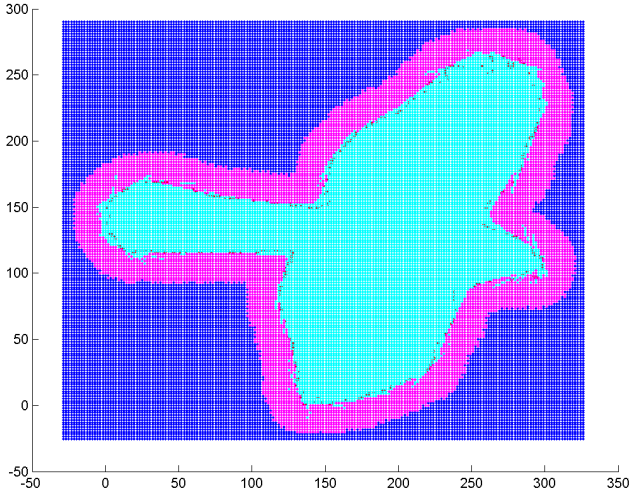


Figure 4.23: The Bird image is reconstructed with 0.93 similarity to the original when $m = 12$.

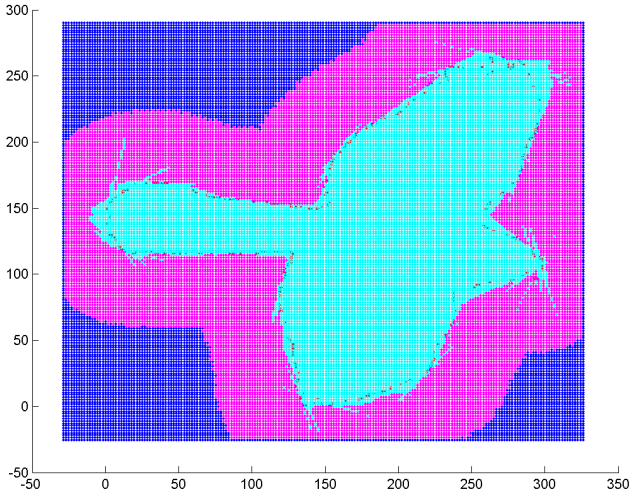


Figure 4.24: The Bird image is reconstructed with 0.90 similarity to the original when $m = 30$.

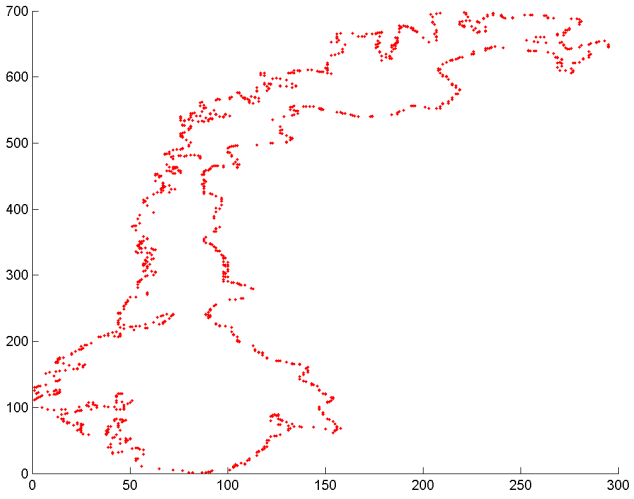


Figure 4.25: The original Norway image was randomly sampled to a set of 1086 surface points.

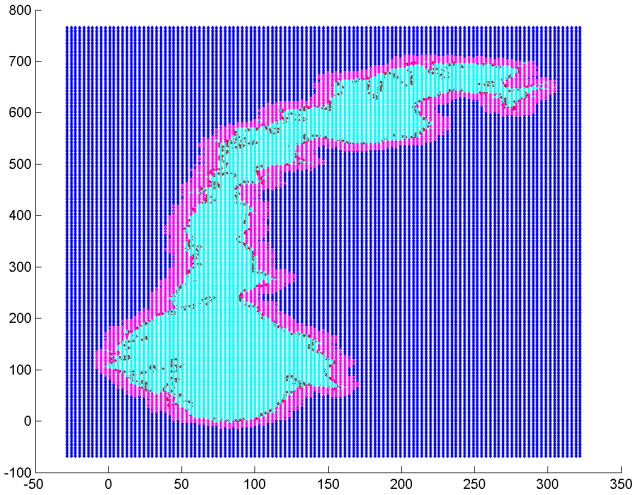


Figure 4.26: The Norway image is reconstructed with 0.88 similarity to the original when $m = 4$.

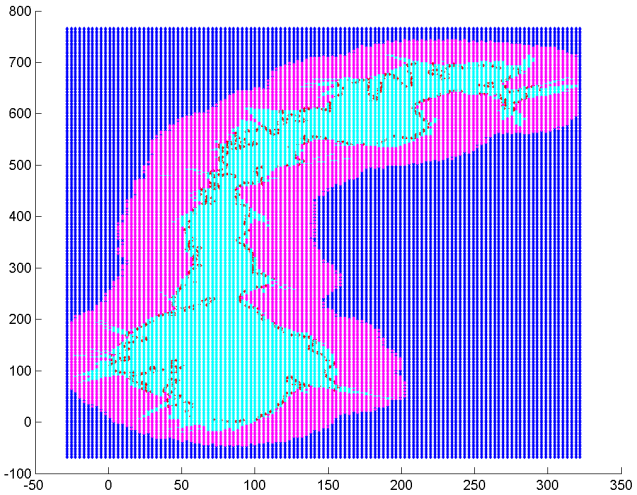


Figure 4.27: The Norway image is reconstructed with 0.79 similarity to the original when $m = 15$.

forgiving when the original shape represents something bulky rather than an image with lots of narrow portions. Still, at our best (0.81 similarity) we produce a good image that comes close to the original.

Still, we acknowledge that 0.56 similarity is poor. This example tested the limits of our approach: what do we get if we take a difficult dataset and combine it with a poor parameter?

4.13 Conclusions

In this chapter we demonstrate the accuracy of our approach when provided a limited set of randomly sampled, unorganized surface points using a multipass approach and a two-parameter-single-pass approach. We feel that it provides a high enough accuracy in two dimensions to proceed with a series of tests on images in three dimensions.

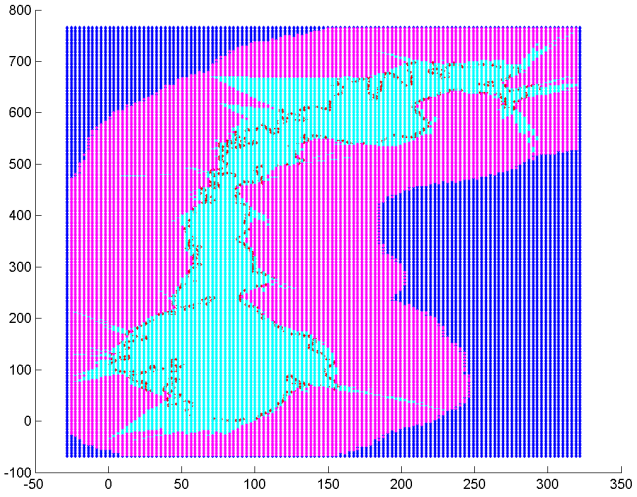


Figure 4.28: The Norway image is reconstructed with 0.72 similarity to the original when $m = 30$.

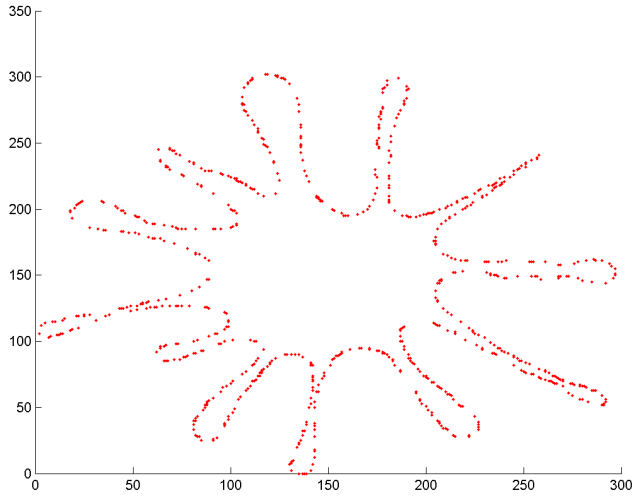


Figure 4.29: The original Splat image was randomly sampled to a set of 742 surface points (representing 25% of the length of the perimeter of the shape.

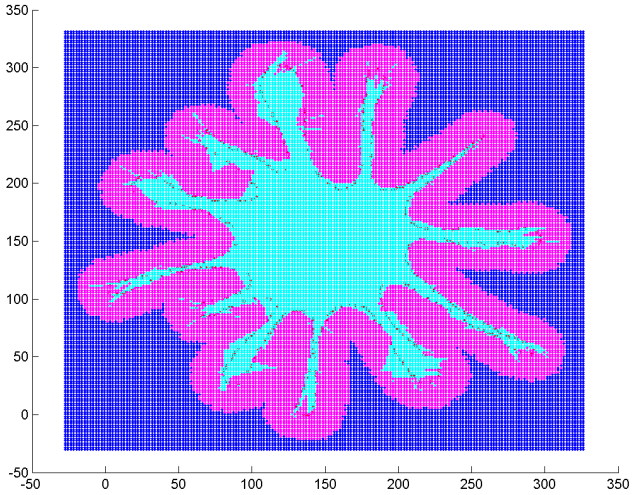


Figure 4.30: The Splat image is reconstructed with 0.81 similarity to the original when $m = 11$.

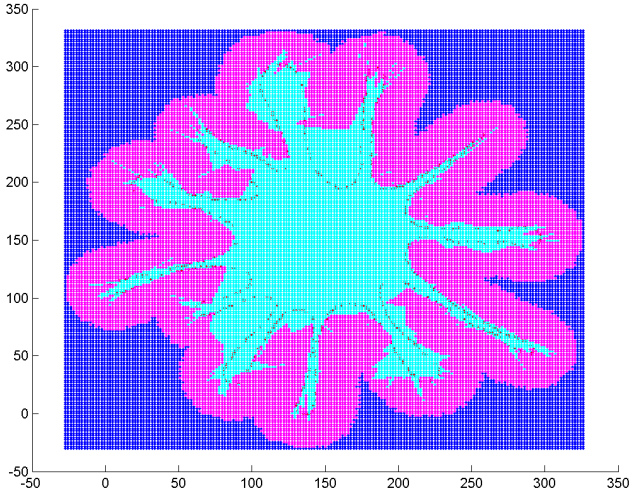


Figure 4.31: The Splat image is reconstructed with 0.69 similarity to the original when $m = 15$.

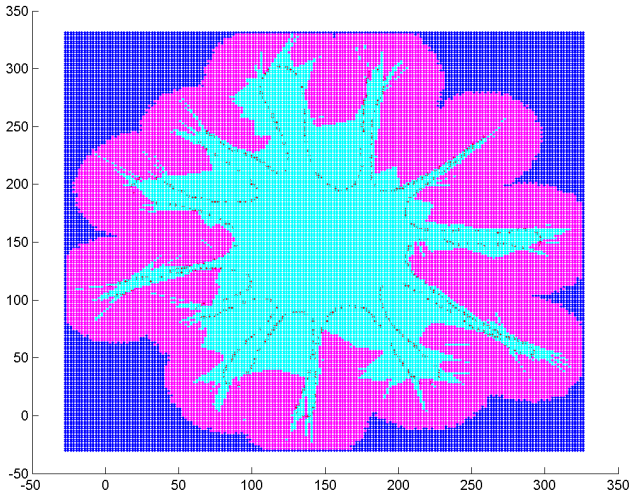


Figure 4.32: The Splat image is reconstructed with 0.56 similarity to the original when $m = 20$.

CHAPTER 5

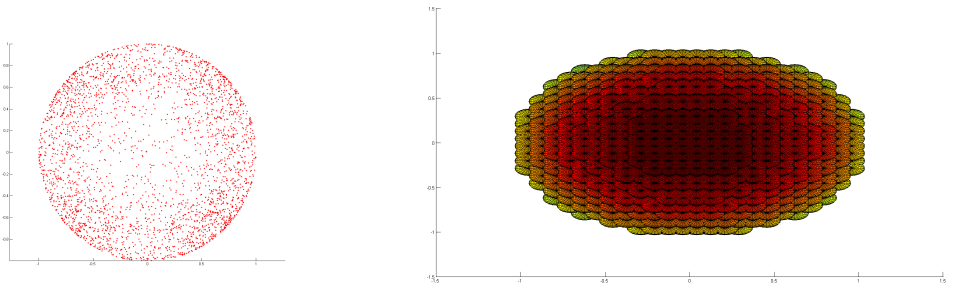
TESTING IMAGE SEGMENTATION IN THREE DIMENSIONS

For testing our algorithm in three dimensions we use a small set of 3D shapes generated by equations by which we can test against a ground truth. We want to demonstrate that our approach works as well in two dimensions as it does in three dimensions and assess the accuracy of a few three dimensional images. Each of the structures tested in this chapter were generated using sampled formulas, which has the benefit of being able to used an unorganized point cloud data as well the ease of which we can create an objective standard by which we can compare results.

Unlike our previous chapter, we chose to perform the cleanup phase approach to each of the tested shapes. We will be reporting the pre-cleanup accuracy and the post-cleanup accuracy.

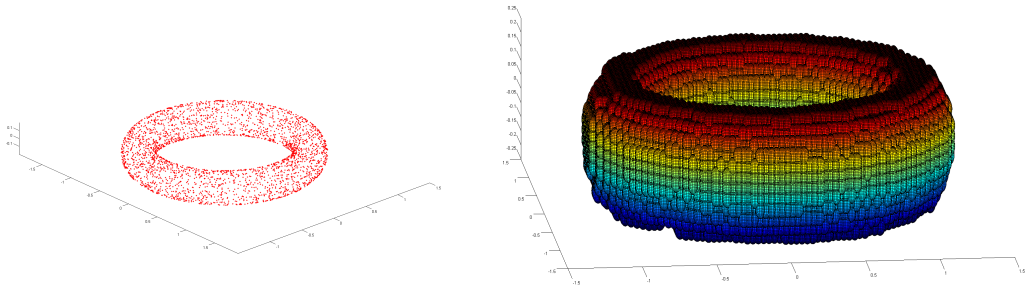
5.1 A Sphere

The sphere represents the simplest shape that we can test: it's a ball of random unit vectors from an origin point. When placing a grid in the space of the sphere, we can assess which points in the grid are inside by testing which points are less than or equal to 1 unit distance from the origin. As was the case with our unit circles mentioned in the previous chapter, it's this circumstance that reveals the error created by having a perfectly curved, smooth surface at all points along the surface of the shape. Prior to cleaning the image, the accuracy of our approach came to be 0.9556. After the cleaning phase, the accuracy of the approach increased slightly to 0.9583. Figure 5.1 demonstrates our input and segmented output of the shape.



(a) Sphere Point Cloud (b) Sphere, 3D Rendered

Figure 5.1: The Sphere Point Cloud was passed through our adaptive segmentation algorithm. The image on the right represents the 3D rendered look at the sphere point cloud once the image has been segmented. The algorithm achieved almost a 96% accuracy. This is the most interesting view.

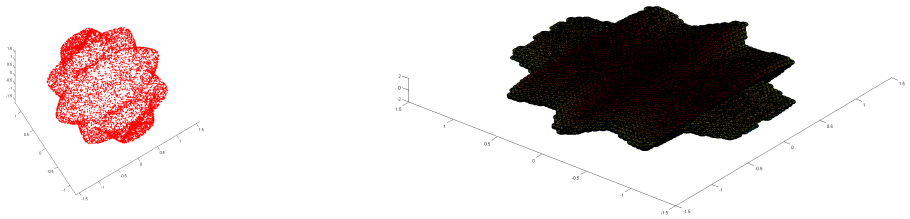


(a) Torus Point Cloud (b) Torus, 3D Rendered

Figure 5.2: The Torus Point Cloud (left) was segmented and rendered (right).

5.2 A Torus

The torus shape is another relatively simple shape to discern. To generate the torus, we generated two random values for each point: a value from $[0, 2\pi)$ representing the radian angle from the center of the torus along a major radius of 1 unit and a value from $[0, 2\pi)$ representing the radian angle along the surface of the shape along a minor radius of 0.2. Generating the ground truth for the torus required that we take every grid point and compute the distance from that point to the nearest point along the circle created by the major radius. If this distance was less than or equal to 0.2, that point was considered 'inside' the shape. Figure 5.2 demonstrates our input and segmented output of the shape. Prior to cleaning the image, the accuracy of our approach was 0.8271. After cleaning, the accuracy increased to 0.8345.



(a) Star Point Cloud (b) Star, 3D Rendered

Figure 5.3: The Star Point Cloud (left) was segmented and rendered (right).

5.3 A Star

Our star shape is the same basic shape that was used in the previous chapter, but after generating the shape we add third dimension to our shape by generating a random theta value from $[0, \pi)$ for each point and rotate that point about the y-axis by this randomly generated theta. Figure 5.3 demonstrates our input and segmented output of the shape. Prior to cleaning the image, our approach's accuracy was 0.9006. After the cleaning phase, it increased to 0.9214.

5.4 Conclusion

In this chapter we looked at 3 randomly sampled surfaces and how our complete image segmentation algorithm reacts. We also examine the cleanup phase of the algorithm (which we purposely avoided in the previous chapter). In all three cases, we note that there was at least a slight improvement in the accuracy of the segmentation process. There is no evidence of aliasing lines in any of our images. We conclude that our algorithm works well in two dimensions as well as three dimensions and our cleanup algorithm provides an equivalent or better enhancement of the results.

CHAPTER 6

RIDGE DETECTION

6.1 The Spatial Depth Formulation

Unlike other contemporary work, this approach to medial axis estimation does not utilize the Delaunay or Voronoi decomposition. Rather, we use a localized spatial median [12] approach to estimating the medial axis. The advantage of the Voronoi decomposition is that it always provides the discrete medial axis, provided that the shape is perfectly known. The various techniques involving the Voronoi decomposition and sampled input data focus on different aspects of the problem of incompleteness. Our approach using the localized spatial depth formalization allows us to create an implicit interpolation of the medial axis. Areas without sufficient coverage that fail to create a ridge are thus passed over by the algorithm. Our reason for opting for the spatial median formulation is that it turns the discrete set of surface points into a suitable image which then allows us to take advantage of an array of computer vision techniques.

The initial input is the sampled surface points of an object. We begin by crafting a localized spatial depth image of the initial shape at a specified grid interval. The spatial depth of a surface scan is created by calculating a weight for each point p within an image based on an observation (or surface scan point) o at d dimensionality and a particular window size σ:

$$w(p, o) = e^{-\frac{\sum_{i=1}^{d}(p-o)^2}{\sigma^2}}.$$

Once the weights have been computed for the entire set of observations O (numbering n observations), we combine those weights into a spatial depth:

$$Spatial\ Depth(p, O) = 1 - \left\| \sum_j^n \left[\frac{w(p,O_j)}{\sum_{i=1}^n w(p,O_i)} \times \frac{p - O_j}{\sqrt{\sum_{i=1}^n (p-O_i)^2}} \right] \right\|.$$

The spatial depth is a reflection of the symmetry of a point in space to the surface points of our scan. The Spatial Depth formula has a bound of 0 to 1 and higher values indicate a better symmetry of surrounding surface points. It is by this property that we formulate our ridge detector.

6.1.1 Selecting the Right σ

Care should be taken in selecting the proper σ value in order to reconstruct the medial axis when using the spatial depth formulation in the context of estimating the medial axis of a shape. Values of σ that are too small run the risk of being too local for a proper analysis of the ridges. Values of σ that are too large run the risk of creating a border area where the calculated values within σ units of the image edge are ineffective due to edge noise.

We created a cylinder shape to use as a test case for investigating the most effective value of σ. The shape consists of a series of rings stacked to resemble that of a three dimensional cylinder. Each ring in the cylinder consist of 128 evenly spaced points along the perimeter of a circle with a radius of 1. The shape consists of 24 rings that are evenly spaced $\frac{1}{6}$ units apart. The overall shape consists of 3072 surface points. The overall dimension of the shape is 2 units by 2 units by 3.833 units.

A sufficiently small delta of 0.005 was selected and a spatial depth image was created using 100 σ values ranging from 0.01 to 2. The small delta produced an image dimension of 401 by 401 by 767 voxels. Only the center line of the cylinder image was relevant to our analysis. We plucked the single dimension line existing long the z-axis coordinate for all 100 σ values. Since the center line of the shape represents the highest conceptual spatial depth, we took the sum of each line. Our finding for this circumstance was that the σ representing the highest summation of spatial depths was 0.5, which is $\frac{1}{4}$ the diameter of our cylinder.

In practice, we feel that a σ should be selected that is roughly $\frac{1}{4}$ the diameter of the

tubular shape being studied. It may require observing the various appendages of the shape prior to analysis and determining a proper σ value.

6.2 Estimating a Medial Axis

In order to determine the ridges in three dimensional space, we must rotate each point within the (x, y, z) coordinate frame to a (p, q, r) coordinate frame individually so that one of the three directions is aligned with the ridge direction. To reduce the amount of noise in the image, a scale-space filter is first applied to the image. Scale space smoothing is a smoothing function using parameter t to determine the smoothness. t is a non-negative integer value which represents the scale of the smoothing. A larger t indicates a greater smoothing scale. If $t = 0$, the image is not smoothed. For an input sample with no noise, no smoothing is required. For noisy data, larger values of t will be required. The smoothing function, as defined by Lindeberg in [24] is

$$g(x, y, z; t) = \frac{e^{\frac{-(x^2+y^2+z^2)}{2t}}}{2\pi t^{\frac{3}{2}}}.$$

The smoothed image is defined by the convolution of the smoothing function and the image function (a.k.a the brightness function):

$$L(x, y, z; t) = g(x, y, z; t) * f(x, y, z).$$

Lindeberg's 2D formulation of the scale space ridge detector required that each voxel within the image be rotated from a (x, y) coordinate frame to a (p, q) coordinate frame. The rotation matrix used to rotate the points was based on the eigenvectors discovered by the eigendecomposition of the Hessian matrix at point (x, y). In order to rotate each point within the image from the (x, y, z) coordinate frame to the (p, q, r) coordinate frame, we compute the 1^{st} and 2^{nd} derivatives of the image at point (x, y, z). The 1^{st} derivatives are known as L_x, L_y, and L_z. From the 1^{st}, we find the 2^{nd} derivatives by taking the derivatives

of each of the 1^{st} derivatives. The 2^{nd} derivatives are known as L_{xx}, L_{xy}, L_{xz}, L_{yy}, L_{yz}, and L_{zz}.

Using the 2^{nd} derivatives of the image, we formulate the Hessian Matrix:

$$H_{(x,y,z)} = \begin{bmatrix} L_{xx} & L_{xy} & L_{xz} \\ L_{xy} & L_{yy} & L_{yz} \\ L_{xz} & L_{yz} & L_{zz} \end{bmatrix}.$$

Using the Hessian matrix, we perform the eigendecomposition to discover the eigenvalues of $H_{(x,y,z)}$: $\lambda_1, \lambda_2, \lambda_3$. The eigenvalues correspond to the 2^{nd} derivatives of the system when rotated to the p, q, and r axis:

$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} L_{pp} \\ L_{qq} \\ L_{rr} \end{bmatrix}.$$

We also determine the eigenvectors for each of the eigenvalues,

$$V = \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{bmatrix},$$

where each column of V represents an eigenvector of the Hessian matrix.

We wish to formulate a rotation matrix that will transform the 1^{st} derivatives of the image along the x, y, and z axis to the p, q, and r axis. We call this 3×3 rotation matrix R. If ∇g is the 1^{st} derivatives at a point in (x, y, z) space, then $R \times \nabla g$ equals our 1^{st} derivatives in (p, q, r) space, i.e.

$$\nabla g = \begin{bmatrix} \partial x \\ \partial y \\ \partial z \end{bmatrix},$$

$$R \times \nabla g = \begin{bmatrix} \partial p \\ \partial q \\ \partial r \end{bmatrix}.$$

We can show that

$$\begin{aligned}
&(R \times \nabla g) \times (R \times \nabla g)^T \times L \\
&= \begin{bmatrix} L_{pp} & L_{pq} & L_{pr} \\ L_{pq} & L_{qq} & L_{qr} \\ L_{pr} & L_{qr} & L_{rr} \end{bmatrix} \\
&= \begin{bmatrix} \partial p \partial p L & \partial p \partial q L & \partial p \partial r L \\ \partial p \partial q L & \partial q \partial q L & \partial q \partial r L \\ \partial p \partial r L & \partial q \partial r L & \partial r \partial r L \end{bmatrix} \\
&= \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}.
\end{aligned}$$

Now we evaluate $(R \times \nabla g) \times (R \times \nabla g)^T \times L$:

$$\begin{aligned}
&(R \times \nabla g) \times (R \times \nabla g)^T \times L \\
&= (R \times \nabla g \times \nabla g^T \times R^T) \times L \\
&= R \times (\nabla g \times \nabla g^T \times L) \times R^T \\
&= R \times H_{(x,y,z)} \times R^T
\end{aligned}$$

By definition,

$$R \times H_{(x,y,z)} \times R^T = V^T \times H_{(x,y,z)} \times V.$$

We can further prove that R is equal to V^T by verifying each of the following:

$$L_{pp} = \partial p \partial p L = a^2 L_{xx} + 2abL_{xy} + 2acL_{xz} + b^2 L_{yy} + 2bcL_{xz} + c^2 L_{zz} = \lambda_1 \tag{6.1}$$

$$L_{qq} = \partial q \partial q L = d^2 L_{xx} + 2deL_{xy} + 2dfL_{xz} + e^2 L_{yy} + 2efL_{xz} + f^2 L_{zz} = \lambda_2 \tag{6.2}$$

$$L_{rr} = \partial r \partial r L = g^2 L_{xx} + 2ghL_{xy} + 2giL_{xz} + h^2 L_{yy} + 2hiL_{xz} + i^2 L_{zz} = \lambda_3 \tag{6.3}$$

$$L_{pq} = \partial p \partial q L = a(dL_{xx} + eL_{xy} + fL_{xz}) + b(dL_{xy} + eL_{yy} + fL_{yz}) + c(dL_{xz} + eL_{yz} + fL_{zz})) = 0 \tag{6.4}$$

$$L_{pr} = \partial p \partial r L = a(gL_{xx} + hL_{xy} + iL_{xz}) + b(gL_{xy} + hL_{yy} + iL_{yz}) + c(gL_{xz} + hL_{yz} + iL_{zz})) = 0 \tag{6.5}$$

$$L_{qr} = \partial q \partial r L = d(gL_{xx} + hL_{xy} + iL_{xz}) + e(gL_{xy} + hL_{yy} + iL_{yz}) + f(gL_{xz} + hL_{yz} + iL_{zz})) = 0 \tag{6.6}$$

Therefore, the matrix V^T is our rotation matrix R. With this we can determine L_p, L_q, L_r:

$$R \times \begin{bmatrix} L_x \\ L_y \\ L_z \end{bmatrix} = \begin{bmatrix} L_p \\ L_q \\ L_r \end{bmatrix}.$$

Using our values of L_p, L_q, L_r, L_{pp}, L_{qq}, and L_{rr}, we can define a ridge. A ridge in three dimensions is a point in space which has a negative gradient change along two 2^{nd} derivatives and no negative gradient change along the third 1^{st} derivative and 2^{nd} derivative.

It's this third direction where a ridge line exists that can be followed for our purposes. We can define a ridge using the following:

$$\begin{cases} L_p = 0 \\ L_{qq} < 0 \\ L_{rr} < 0 \\ |L_{pp}| < |L_{qq}| \\ |L_{pp}| < |L_{rr}| \end{cases} \text{or} \begin{cases} L_q = 0 \\ L_{pp} < 0 \\ L_{rr} < 0 \\ |L_{qq}| < |L_{pp}| \\ |L_{qq}| < |L_{rr}| \end{cases} \text{or} \begin{cases} L_r = 0 \\ L_{pp} < 0 \\ L_{qq} < 0 \\ |L_{rr}| < |L_{pp}| \\ |L_{rr}| < |L_{qq}| \end{cases}$$

6.3 Implementing Canny's Edge Detector in 3D

To assist in determining which points can be identified as ridges, we implement a "dual threshold" technique for identifying ridges similar to the approach developed by Canny for identifying edges [7]. Canny's edge detector required two things to be effective: strength and direction.

We formulate the strength of the ridge by summing the absolute values of the 2^{nd} derivatives orthogonal to that of the ridge direction. Due to inevitable nuisances (i.e. rounding error), we must apply a narrow threshold on the ridge conditionals. We have found a threshold of 0.001 to be effective. We have developed a "ridge strength" (RS) to help us determine the strength of an edge based on the eigengap of a ridge in pqr-space:

$$RS = \begin{cases} |L_{rr}| + |L_{qq}| \\ \qquad \text{if } |L_p| < ridgeThresh \wedge \\ \qquad\quad L_{qq} < 0 \wedge L_{rr} < 0 \wedge \\ \qquad\quad |L_{pp}| < |L_{qq}| \wedge |L_{pp}| < |L_{rr}| \\ |L_{pp}| + |L_{rr}| \\ \qquad \text{if } |L_q| < ridgeThresh \wedge \\ \qquad\quad L_{pp} < 0 \wedge L_{rr} < 0 \wedge \\ \qquad\quad |L_{qq}| < |L_{pp}| \wedge |L_{qq}| < |L_{rr}| \\ |L_{pp}| + |L_{qq}| \\ \qquad \text{if } |L_r| < ridgeThresh \wedge \\ \qquad\quad L_{pp} < 0 \wedge L_{qq} < 0 \wedge \\ \qquad\quad |L_{rr}| < |L_{qq}| \wedge |L_{rr}| < |L_{pp}| \\ 0 \\ \qquad \text{otherwise} \end{cases}.$$

Provided that the RS score at a point is greater than 0, we determine the ridge direction RD by retrieving the eigenvector of the primary ridge determined by the maximum of L_{pp}, L_{qq}, or L_{rr}.

$$RD = \begin{cases} \begin{bmatrix} x_1 y_1 z_1 \end{bmatrix}^T, & \text{if } L_{pp} = max(L_{pp}, L_{qq}, L_{rr}), \\ \begin{bmatrix} x_2 y_2 z_2 \end{bmatrix}^T, & \text{if } L_{qq} = max(L_{pp}, L_{qq}, L_{rr}), \\ \begin{bmatrix} x_3 y_3 z_3 \end{bmatrix}^T, & \text{if } L_{rr} = max(L_{pp}, L_{qq}, L_{rr}). \end{cases}$$

It should be noted that while the input shape can and will contain holes, our algorithm

makes no guarantees as to whether or not a discovered ridge exists on the interior or exterior of the shape. Still we can reduce the number of exterior ridges by filtering out all of the candidate ridges if they do not also have a local spatial depth greater than a high threshold (0.9 in our experiments).

Once we have determined the set of candidate ridges, their scores and their directions, we must now select two threshold values by which to select our ridges. We call these thresholds *high* and *low*. First, the image is scanned for all voxels with RS scores greater than *high*. These scores and locations are added to a priority queue where priority is determined by greatest RS score. While the priority queue is not empty, take the front element off the queue and label the point as a confirmed point. We then look at the two voxels in the directions of RD and $-RD$ and determine if those voxels have a RS score greater than *low*. If so, we add the voxel location and RS to the priority queue.

6.4 The Complete Algorithm

1 Create a localized spatial depth image I of the shape using a predetermined window size σ.

2 Perform the scale space smoothing filter on the image I with preferred scaling factor t.

3 For each point within the image I, determine the 1^{st} derivative which represents L_x, L_y, and L_z.

4 For each of the 1^{st} derivatives, determine the 2^{nd} derivatives and formulate the Hessian matrix H.

5 Compute $[\lambda, V] = eig(H)$, where λ and V are the eigenvalues and eigenvectors of matrix H. The λ represents the 2^{nd} derivative directions in the (p, q, r)-coordinate frame: L_{pp}, L_{qq}, L_{rr}.

6 Compute the rotation matrix R for this point, which is V^T.

7 Determine the 1^{st} derivative directions in the (p, q, r)-coordinate frame, which is $R \times [L_x, L_y, L_z]^T$, which are represented by L_p, L_q, L_r.

8 For each point that meets the criteria for a ridge, compute the ridge score for this ridge point.

9 For each point that meets the criteria for a ridge, note the eigenvector associated with the largest eigenvalue of the Hessian matrix at this point. This becomes the dominate eigenvector.

10 Select a desired *low* and *high* threshold values for ridge points.

11 Create a three dimensional matrix equal to the size of the image I composed of all Boolean false flags. This matrix will represent the confirmed ridges.

12 For each ridge point that has a RS greater than the *high* threshold, add that ridge point's ridge score and location to the end of a priority queue.

13 While the priority queue is not empty:

   a Sort the priority queue by the ridge score in descending order.

   b Dequeue the first element from the priority queue. Flag this point in the confirmed ridge point matrix as true. If it has been previously flagged as a confirmed ridge, skip the remaining steps in this loop iteration.

   c Test the neighboring point in the direction of the dominate vector for this point. If the neighboring point meets the criteria for a ridge and has a ridge score greater than the *low* threshold, add the neighboring ridge score and the ridge point location to the end of the priority queue.

   d Test the neighboring point in the direction opposite of the dominate vector for this point. If the neighboring point meets the criteria for a ridge and has a ridge score greater than

the *low* threshold, add the neighboring ridge score and the ridge point location to the end of the priority queue.

6.5 Experimental Result

In order to test the validity of our algorithm, we created three dimensional images representing both artificial structures and real objects scanned with a 3D scanner. For each shape, we select an appropriate σ, smoothing parameter t, and ridge thresholds $high$ and low and executed the algorithm, which can be seen in Figure 6.1. For the artificial structures, we evaluate each image based on the identified ridges by how closely they conform to a known center line of the shape. We have compared our results with the Power Crust method using a simple correspondence algorithm using a Root-Mean-Squared Error formulation comparing each point in the discovered medial axis to the nearest point in the ground truth (RMSE MA) and comparing each point in the ground truth to the nearest point in the discovered medial axis (RMSE Truth). We can show that the two approaches are comparable in Table 6.1.

U-Pipe	RMSE MA	RMSE Truth	Avg.
L. Spatial Depth	0.0786	0.2088	0.1437
Power Crust	0.0042	0.1117	0.0579
Cork Screw	RMSE MA	RMSE Truth	Avg.
L. Spatial Depth	1183.4	6748.3	3965.9
Power Crust	16182.0	2564.0	9373.1

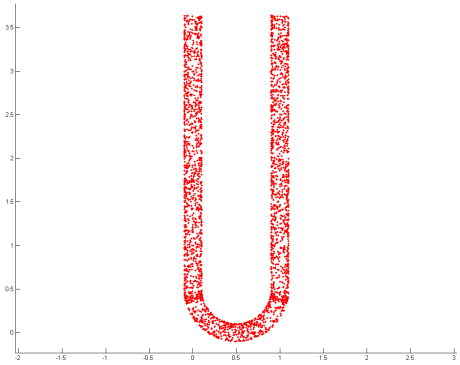
Table 6.1: We show that the Localized Spatial Depth and the Power Crust method have comparable results.

6.6 Conclusions

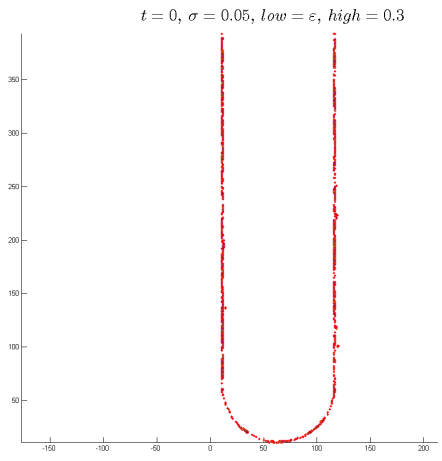
In each experimental result, we have discovered enough ridge lines in order to connect the ridges using Dijkstra's shortest path algorithm. Our data sets mostly represent the elongated structures of fish with an anterior, a narrow body, and a posterior. By flagging the two points representing the anterior and posterior, we can use this technique to create a full length vertebra of the internal structure of the shape. After obtaining the body-length

vertebra, to unfold the structure is trivial: associate each surface point with the nearest vertebra point and align it on a single axis, taking care to rotate the point with respect to the vertebrae point immediately preceding the connecting vertebra point. There is one problem with this approach which needs addressing in our future work. The body contortions of the fish specimen will cause the fish's body to twist around its own internal structure. The end result of our unfolding process will also be a twisted contortion around the discovered vertebra. We feel that one way to alleviate this issue is to add landmarks along the dorsal side of the fish specimen. With this information, we can perform a rotation along the roll axis of the specimen.

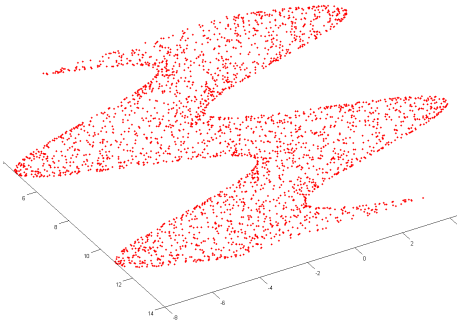
Our algorithm depends heavily on the selection of a σ window. A good σ window can fall into a range of values, but it is still dependent on the diameter of the appendages of the object being analyzed. We feel that there is more research that can be done into evaluating the shape prior to the start of the algorithm that could be used in σ discovery.



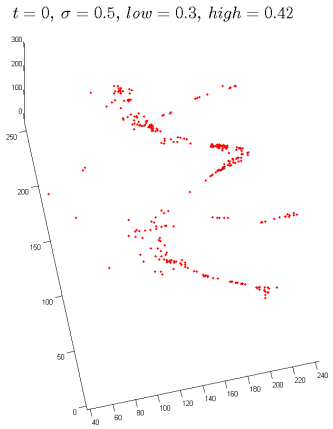
(a) Surface of Randomly Sampled U-Pipe



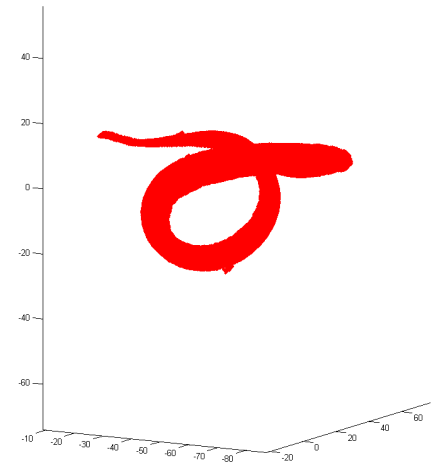
(b) Medial Axis of U-Pipe



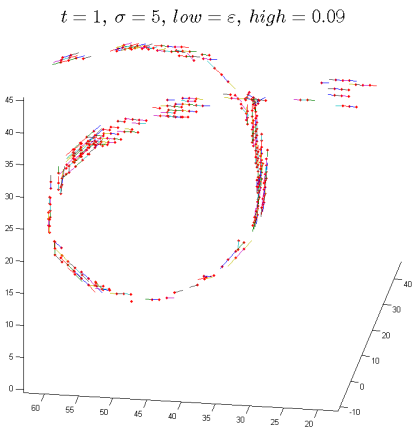
(c) Surface of Randomly Sampled Corkscrew



(d) Medial Axis of Corkscrew



(e) Surface of Randomly Sampled Eel Surface



(f) Medial Axis of Eel

Figure 6.1: Shape and Estimated Medial Axis of Select Images

CHAPTER 7

MERGING IMAGE SEGMENTATION AND RIDGE DETECTION

This chapter will describe a method to combine the algorithms described in Chapters 3 and 6 to create one algorithm that is proficient in detecting ridge lines in three dimensional space that are also internal to a shape (which is know by sampling of unorganized surface points). The algorithms work independently at their respective tasks, but this chapter merges the two into one and presents results.

We merge the two algorithms in this manner. This is also a summary of our complete unfolding algorithm.

1. Create a 'label grid' of unlabeled points in the vicinity of a shape.
2. Perform shape segmentation on the shape to label the cells of the grid as 'inside' and 'outside'.
3. Perform Minkowski Addition on the set of 'inside' points. Call this set of points 'M1'.
4. Perform Minkowski Addition on the set of 'M1' points. Call this set of points 'M2'.
5. Create a spatial depth grid of all zero values that is the same size as grid used in step 1.
6. For points in the label grid that are considered 'M2', compute the spatial depth score.
7. Perform ridge detection analysis of the points in the spatial depth grid for the points labeled 'inside'.

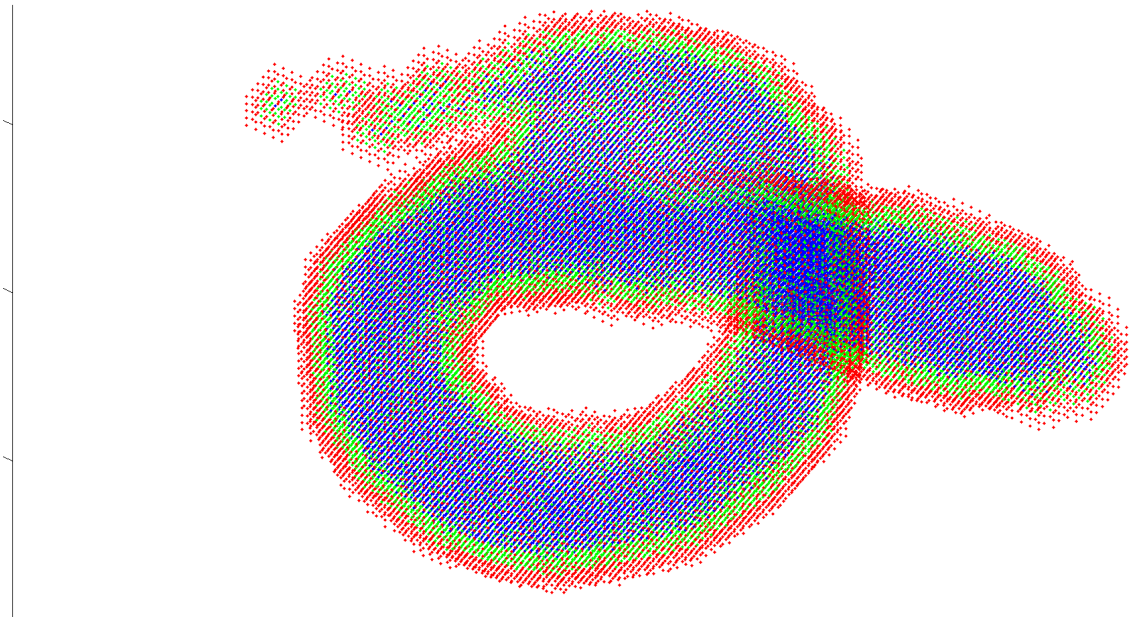


Figure 7.1: After we perform our image segmentation step, we are left with a set of points labeled as 'inside', seen in blue. In green, we see the layer of points created by 1 iteration of Minkowski Addition known as 'M1'. We perform Minkowski Addition on the points of 'M1' creating a new player of points known as 'M2', seen in red.

8. Create a connected grid using the grid. Set edges that align with the ridge detection analysis to zero.

9. Compute a shortest path between two selected points using the Dijkstra algorithm.

10. Unwind this path and align the original surface points of the shape to this new alignment.

Most of these steps are better explained in their respective chapters and we wish to use this chapter to explain how we intend to use these two algorithms in conjunction to solve our initial problem of creating a single internal structure by which we can manipulate the surface points of an image.

7.1 Gluing our two algorithms together

Our two algorithms both use grids for storing information. The first algorithm (Image Segmentation) uses a grid for maintaining the point labels within the grid space. Our second

algorithm (ridge detection) uses three grids (spatial depth grid, a ridge score grid, and a connected path grid). The complete spatial depth algorithm is a time consuming algorithm with a runtime of $O(n * m)$, were n is the number of grid points and m is the number of surface points. This runtime can be reduced without a loss of accuracy to $O(i * m)$, where i is the set of points in 'M2' (described in the algorithm at the beginning of this chapter). This creation of additional layers to our internal shape is seen in Figure 7.1.

Minkowski Addition is a binary expansion process that considers a vector mask A which is a 3^d-mask (where d is the dimensionality of the data) of all 1 values and a vector B, which is any given point in binary field. We then perform

$\{A + B = \{\mathbf{a} + \mathbf{b} \,|\, \mathbf{a} \in A,\ \mathbf{b} \in B\}\}$.

on the binary field at every point. We perform Minkowski Addition twice to our set of internal points discovered in the shape segmentation step (thus we create a set of points with two layers of points outside the interior) in order to prepare for the ridge detection step. We call this set of points 'M2' and its this set of points for which we will perform the spatial depth analysis. All points not in 'M2' will have a spatial depth of 0, which is incorrect, but the remainder of the algorithm will be ignoring this set of points. We then perform our ridge detection analysis on the set of points labeled as 'inside'. Ridge detection analysis involves the creation of six 2^{nd} derivative images which are used in the formation of a Hessian matrix. By first performing Minkowski Addition on the internal set of points twice, we can ensure that the matching ridge scores of the internal grid will be accurate because the 2^{nd} derivative images at those points will not take into account the incorrect portions of the spatial depth grid that have already been acknowledged.

Using this addendum to the previous proposed algorithms in Chapters 3 and 6, we have created a single algorithm for detecting a set of ridges that will reside on the set of points labeled as 'inside' to a shape.

7.2 Connecting the grid points

At the conclusion of our ridge detection analysis, we are left which a series of ridges within our grid space. Depending on how the ridge detection algorithm is implemented, ridges may be discovered throughout the shape, but we should concern ourselves with those detected ridges that coincide with 'inside' points. Our next step is to utilize this information provided by the ridge detector to create a connected graph. In addition to the detected ridges, we also use the spatial depth scores to fill in the blanks where ridges have not been detected in our graph. We begin by creating another node weights grid NW that is initialized to the spatial depth scores obtained by the spatial depth analysis of 'M2'. Recall that not all points within 'M2' are analyzed and those that are not analyzed will remain as zero. We will use this to our advantage. After creating our new node weights grid that is a copy of the spatial depth grid, we overwrite the values of the node weights grid where internal ridges have been detected with a new value of infinity.

We must create our new graph consisting of nodes and edges. Each point within the grid will be a graph node. Each node will have $(3^d) - 1$ neighbors (this corresponds to a 8-connected neighborhood window in two dimensions and a 27-connected neighborhood window in three dimensions). Each edge weight within the graph will be determined based on the following definition,

$$E(N_1, N_2) = e^{-(\frac{NW(N_1)+NW(N_2)}{\sigma})^2},$$

where N_1 and N_2 represent two nodes that need connecting. This formulation is designed to create low edge weights when two nodes are assigned high node weights and high edge weights when two neighboring nodes have low node weights. We want any path that traverses through the shape to stay along discovered ridges within the internal portion of the shape as much as possible.

7.3 Unfolding the Image

We began this long journey with the assumption that it would be necessary to use this approach for the purpose of unfolding elongated structures that had been twisted into a knot. We wish to put the final touches on our algorithm to do just that. Utilizing the graph of grid points with assigned edge weights, we now find a center path using an appropriate shortest path algorithm within this graph connecting two physical points along the specimen (say, the head and the tail). The shortest path algorithm can be selected based on the needs of the problem and we have found Dijkstra's algorithm to be sufficient for solving this problem.

Once a center path has been created from two selected points within the graph, we can straighten the surface our specimen. First, we map each point along the surface to the nearest point along the specimen. We do this by finding the closest point along the center path (excluding one end of the path) to every point in the surface. We reserve one end point in our center path as an anchor point by which we will rotate all of the other points in the surface. In the three dimensional case, the rotation of a surface point about a center path point is a two step process and described in Algorithm 4. Three required pieces of information are a surface point, the associated nearest center path point, and that center path's neighbor in the direction of the anchor point. To associate a rotated point to the proper location in the new surface, we need the distance between two grid points.

With that, our algorithm is complete.

Data: S: Surface, P: Center Path Points, delta: the distance between two path points

Result: S': The Surface, which each point in a new position

for i *in* $1..\|S\|$ **do**

$P_j \leftarrow NearestPoint(P, S_i)$;

$p \leftarrow P_{j-1} - P_j$ // Find the direction of P_{j-1} from P_j;

$c \leftarrow S_i - P_j$ // Find the direction of S_i from P_j;

$a \leftarrow \{-\|p\|00\}$ // Rotate vector p to align with the x axis;

$xydist \leftarrow \sqrt{\sum_{i=\{1,2\}}(p_i - a_i)^2}$ // Distance from p to a, ignoring z direction;

$xytheta \leftarrow |acos(\frac{2*\|a\|^2 - xydist^2}{2\|a\|^2})|$ // Find the angle between p and a, ignoring z direction;

$$R_z \leftarrow \begin{bmatrix} cos(xytheta) & -sin(xytheta) & 0 \\ sin(xytheta) & cos(xytheta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ // Create xy-rotation matrix;

$c \leftarrow R_z \times c$ // Rotate c along z axis;

$xzdist \leftarrow \sqrt{\sum_{i=\{1,3\}}(p_i - a_i)^2}$ // Distance from p to a, ignoring y direction;

$xytheta \leftarrow |acos(\frac{2*\|a\|^2 - xzdist^2}{2\|a\|^2})|$ // Find the angle between p and a, ignoring y direction;

$$R_y \leftarrow \begin{bmatrix} cos(xytheta) & 0 & -sin(xytheta) \\ 0 & 1 & 0 \\ sin(xytheta) & 0 & cos(xytheta) \end{bmatrix}$$ // Create xy-rotation matrix;

$c \leftarrow R_y \times c$ // Rotate c along y axis.;

$c \leftarrow c + i * delta$ // Place this new point in the correct location along straightened path ;

$S'_i \leftarrow c$;

end

Algorithm 4: Shape Unfolding for a shape defined by a surface in three dimensions with a determined center path.

CHAPTER 8

CONCLUSIONS

In this dissertation we examine the following problems:

- Is it possible to create an efficient shape segmentation algorithm based on localized surface normals of unorganized surface points and demonstrate that the approach is sufficiently accurate?
- Can we use localized spatial depth information to detect localized, reliable ridges in 3D space?
- Can we use what we learned from the previous two questions to untie a virtual knot?

I believe we have answered 'yes' adequately to all three questions.

In Chapter 3 we looked at the problem of segmenting an area based on hyperplanes discovered through localized Principle Component Analysis. We decided it was best to create a grid of unlabeled points surrounding our shape and allow a percolation algorithm to grow around that shape beginning from a distant point that is obviously exterior to the shape. The algorithm wraps itself around the shape, hugging as close to the shape's surface as allowed by the union of hyperplanes defined by the local PCA. There is an essential parameter called the threshold distance d that defines how large a 'bubble' can be before we say that a hyperplane blocks the progression of the algorithm. We attempted several approaches at giving this parameter an adequate definition and we settled upon an adaptive parameter that allows the algorithm to shrink the size of this parameter as the algorithm progresses. This adaptive approach allows us to recreate the original shape with an accuracy

Figure 8.1: Eel unfolded.

of 70% in most cases and as high as 90% in many cases without having to know anything about the internal structure of the shape being studied. Chapters 4 and 5 describe testing of this approach in two and three dimensions. We feel that the accuracy achieved by this approach is sufficient to justify a success.

In Chapter 6 we use the results of the chapter on image segmentation in order to best create a center line path from the head to the tail. We use the localized spatial depth statistic to identify the spatial depth image of our surface points, but only for the points determined to be interior to the shape. Using the spatial depth information, we created our own ridge detector based on the two dimensional ridge detector created by Lindeberg and the edge detector created by Canny. The ridges provided by our ridge detector turned out to be perfect for the needs of our problem.

In Chapter 7 we use what we learned from the previous chapters to create a complete algorithm that allows us to merge the approaches into a single cohesive algorithm for effectively creating a center line path down the middle of a shape. Using that center line path, we apply standard three dimensional rotation matrices and straighten the shape. We demonstrate this result in Figure 8.1 where we have taken an unorganized surface point cloud data representing an eel and passed it through all of the phases of our approach resulting in an unfolded eel.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. NSF MCB-1027989 and MCB-1027830. We would like to thank Aishat

Aloba for processing large data sets.

BIBLIOGRAPHY

[1] N. Amenta, S. Choi, and R. K. Kolluri, "The power crust," in *Proceedings of the sixth ACM symposium on Solid modeling and applications*. ACM, 2001, pp. 249–266.

[2] N. Amenta and R. K. Kolluri, "Accurate and efficient unions of balls," in *Proceedings of the sixteenth annual symposium on Computational geometry*. ACM, 2000, pp. 119–128.

[3] C. L. Bajaj, F. Bernardini, and G. Xu, "Automatic reconstruction of surfaces and scalar fields from 3d scans," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. ACM, 1995, pp. 109–118.

[4] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Transactions on Mathematical Software (TOMS)*, vol. 22, no. 4, pp. 469–483, 1996.

[5] H. Blum, "A transformation for extracting new descriptors of shape," *Models for the perception of speech and visual form*, vol. 19, no. 5, pp. 362–380, 1967.

[6] J.-D. Boissonnat, "Geometric structures for three-dimensional shape representation," *ACM Transactions on Graphics (TOG)*, vol. 3, no. 4, pp. 266–286, 1984.

[7] J. Canny, "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 6, pp. 679–698, 1986.

[8] A.-L. Chauve, P. Labatut, and J.-P. Pons, "Robust piecewise-planar 3d reconstruction and completion from large-scale unstructured point data," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 1261–1268.

[9] F. Chazal and A. Lieutier, "The "λ-medial axis"," *Graphical Models*, vol. 67, no. 4, pp. 304–331, 2005.

[10] Y. Chen, H. Bart, X. Dang, and H. Peng, "Depth-based novelty detection and its application to taxonomic research," in *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*. IEEE, 2007, pp. 113–122.

[11] Y. Chen, H. L. Bart Jr, S. Huang, and H. Chen, "A computational framework for taxonomic research: diagnosing body shape within fish species complexes," in *Data Mining, Fifth IEEE International Conference on.* IEEE, 2005, pp. 4–pp.

[12] Y. Chen, X. Dang, H. Peng, and H. L. Bart, "Outlier detection with the kernelized spatial depth function," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 2, pp. 288–305, 2009.

[13] J. C. Church, R. Schmidt, H. Bart Jr., X. Dang, and Y. Chen, "Straightening 3-d surface scans of curved natural history specimens for taxonomic research," *Studies in Computational Intelligence*, vol. 493, pp. 266–286, 2012.

[14] D. Coeurjolly and A. Montanvert, "Optimal separable algorithms to compute the reverse euclidean distance transformation and discrete medial axis in arbitrary dimension," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 3, pp. 437–448, 2007.

[15] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques.* ACM, 1996, pp. 303–312.

[16] J. Dehmeshki, H. Amin, M. Valdivieso, and X. Ye, "Segmentation of pulmonary nodules in thoracic ct scans: a region growing approach," *Medical Imaging, IEEE Transactions on*, vol. 27, no. 4, pp. 467–480, 2008.

[17] T. K. Dey and S. Goswami, "Tight cocone: a water-tight surface reconstructor," in *Proceedings of the eighth ACM symposium on Solid modeling and applications.* ACM, 2003, pp. 127–134.

[18] H. Edelsbrunner and E. P. Mücke, "Three-dimensional alpha shapes," *ACM Transactions on Graphics (TOG)*, vol. 13, no. 1, pp. 43–72, 1994.

[19] R. M. Haralick, "Ridges and valleys on digital images," *Computer Vision, Graphics, and Image Processing*, vol. 22, no. 1, pp. 28–38, 1983.

[20] D. Holz and S. Behnke, "Fast range image segmentation and smoothing using approximate surface reconstruction and region growing," in *Intelligent Autonomous Systems 12*. Springer, 2013, pp. 61–73.

[21] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, *Surface reconstruction from unorganized points*. ACM, 1992, vol. 26, no. 2.

[22] D. R. Karger, P. N. Klein, and R. E. Tarjan, "A randomized linear-time algorithm to find minimum spanning trees," *Journal of the ACM (JACM)*, vol. 42, no. 2, pp. 321–328, 1995.

[23] T.-Y. Law and P. Heng, "Automated extraction of bronchus from 3d ct images of lung based on genetic algorithm and 3d region growing," in *Medical Imaging 2000*. International Society for Optics and Photonics, 2000, pp. 906–916.

[24] T. Lindeberg, "Edge detection and ridge detection with automatic scale selection," *International Journal of Computer Vision*, vol. 30, no. 2, pp. 117–156, 1998.

[25] A. P. Mangan and R. T. Whitaker, "Partitioning 3d surface meshes using watershed segmentation," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 5, no. 4, pp. 308–321, 1999.

[26] N. J. Mitra, A. Nguyen, and L. Guibas, "Estimating surface normals in noisy point cloud data," *International Journal of Computational Geometry & Applications*, vol. 14, no. 04n05, pp. 261–276, 2004.

[27] G. Mühlenbruch, M. Das, C. Hohl, J. E. Wildberger, D. Rinck, T. G. Flohr, R. Koos, C. Knackstedt, R. W. Günther, and A. H. Mahnken, "Global left ventricular function

in cardiac ct. evaluation of an automated 3d region-growing segmentation algorithm," *European radiology*, vol. 16, no. 5, pp. 1117–1123, 2006.

[28] F. A. Pellegrino, W. Vanzella, and V. Torre, "Edge detection revisited," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 34, no. 3, pp. 1500–1518, 2004.

[29] Y. Qi, W. Xiong, W. K. Leow, Q. Tian, J. Zhou, J. Liu, T. Han, S. K. Venkatesh, and S.-c. Wang, "Semi-automatic segmentation of liver tumors from ct scans using bayesian rule-based 3d region growing," in *MICCAI Workshop*, vol. 41, no. 43, 2008, p. 201.

[30] C. Revol and M. Jourlin, "A new minimum variance region growing algorithm for image segmentation," *Pattern Recognition Letters*, vol. 18, no. 3, pp. 249–258, 1997.

[31] C. Revol-Muller, F. Peyrin, Y. Carrillon, and C. Odet, "Automated 3d region growing algorithm based on an assessment function," *Pattern Recognition Letters*, vol. 23, no. 1, pp. 137–150, 2002.

[32] P. L. Rosin, "Straightening and partitioning shapes," in *Visual Form 2001.* Springer, 2001, pp. 440–450.

[33] Y. Sato, S. Nakajima, N. Shiraga, H. Atsumi, S. Yoshida, T. Koller, G. Gerig, and R. Kikinis, "Three-dimensional multi-scale line filter for segmentation and visualization of curvilinear structures in medical images," *Medical image analysis*, vol. 2, no. 2, pp. 143–168, 1998.

[34] L. Xia, C.-C. Chen, and J. Aggarwal, "Human detection using depth information by kinect," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on.* IEEE, 2011, pp. 15–22.

[35] T. Zhang and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," *Communications of the ACM*, vol. 27, no. 3, pp. 236–239, 1984.

VITA

James Clark Church

EDUCATION

University of Mississippi, University, MS

- Master of Science (2007)
- Doctorate of Philosophy (2014)

University of Tennessee, Martin, TN

- Bachelor of Science in Computer Science (2004)

HONORS AND AWARDS

Eagle Scout Honor Award, Boy Scouts of America, July 2000

Competed in ACM International Programming Competition, 2002 – 2005

President of the Ole Miss Chapter of Upsilon Pi Epsilon, 2005

Best Oral Presentation for Mathematics, Computer Science and Statistics at the 2009 Conferences of the Mississippi Academy of Sciences.

School of Engineering PhD Graduate Achievement Award, University of Mississippi, 2014

TEACHING

Courses taught at Rhodes College:

CS 141 Section 1: Introduction to Computer Science, Fall 2010

CS 241 Section 1: Data Structures in C++, Fall 2010

Courses taught at the University of Mississippi:

CSCI 391 Section 1: Data Analysis using Python and R, Winter Session 2012

CSCI 112 Section 1: Computer Science II using Java, Second Summer 2010

CSCI 259/390 Section 1: Programming in C++, Spring 2010

CSCI 111 Section 3: Computer Science I using Java, Fall 2009

CSCI 490 Section 1: Python Web Development, First Summer 2009

CSCI 259/390 Section 1: Programming in C++, Spring 2009

CSCI 111 Section 1: Computer Science I using Java, Fall 2008

CSCI 112 Section 1: Computer Science II using Java, Second Summer 2008

CSCI 103 Section 3: Survey of Computing using Alice, Spring 2008

CSCI 490 Section 1: Python Web Development, Fall 2007

CSCI 112 Section 1: Computer Science II using Java, Second Summer 2007

CSCI 112 Section 1: Computer Science II using Java, Spring 2007

CSCI 112 Section 1: Computer Science II using Java, Fall 2006

Talks

"Crafting Regular Expressions." University of Mississippi. September 21, 2005

"The Google Maps API." University of Mississippi. February 1, 2006

"Web Application Development using AJAX." University of Mississippi. September 27, 2006

"Image Creation using Python and Numpy." University of Mississippi. October 17, 2007

"A Spatial Median Filter for Noise Removal in High Dimensional Data." ACM Mid-Southeast Fall Conference. November 16, 2007

"A Spatial Median Filter for Noise Removal in High Dimensional Data." IEEE SoutheastCon 2008. April 5, 2008.

"An Introduction to Sage." University of Mississippi. February 4, 2009.

"An Approximation Algorithm for Generating Neighborhood Graphs." Mississippi Academy of Sciences. February 26, 2009.

"Multiparadigm Programming in Scala." Consortium for Computing Sciences in Colleges (CCSC) 2009 Mid-South Conference. April 4, 2009. Given with Dr. H. Conrad Cunningham.

"Unfolding Elongated Solids Identified by Point Cloud Surface Data." University of Mississippi. September 7th, 2012.

"Straightening 3-D surface scans of curved natural history specimens for taxonomic research." 12th IEEE/ACIS International Conference, Niigata, Japan. June 17th, 2013.

PUBLICATIONS

Church, J. C., "A Spatial Median Filter for Noise Removal in High Dimensional Data," In *Proceedings of the ACM Mid-Southeast Fall Conference*, 1 page abstract, Gatlinburg, TN. November 2007.

Church, J. C., Chen, Y., and Rice, S. V. "A spatial median filter for noise removal in digital images." Southeastcon, 2008. IEEE (3-6 April 2008), 618-623.

Church, J. C., and Chen, Y. "A Random Projection Algorithm for Graph Approximation," In *Proceedings of the Mississippi Academy of Sciences 2009 Conference*, 1 page abstract, Olive Branch, MS. February 27, 2009.

Cunningham, H.C., and Church, J. C. "Multiparadigm Programming in Scala," In *The Journal of Computing Sciences in Colleges 2009 Conference*, 1 page abstract, Martin, TN. April 4, 2009.

Church, J. C., et al. "Straightening 3-D surface scans of curved natural history specimens for taxonomic research," In *Studies in Computational Intelligence, Vol. 493*, Book Chapter, Springer Publications, June 2013. The paper was selected to be among the best 20 papers of the 12th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2013) conference.