Spring 5-6-2014

Surface Reconstruction Using Differential Invariant Signatures

Sophors Khut
Macalester College, khut.sophors@gmail.com

Surface Reconstruction Using Differential Invariant Signatures

Sophors Khut

Department of Mathematics, Statistics and Computer Science
Macalester College
May 6, 2014

Advisor: Robert Thompson
Second Reader: Elizabeth Shoop
Third Reader: David Shuman

Abstract

This thesis addresses the problem of reassembling a broken surface. Three dimensional curve matching is used to determine shared edges of broken pieces. In practice, these pieces may have different orientation and position in space, so edges cannot be directly compared. Instead, a differential invariant signature is used to make the comparison. A similarity score between edge signatures determines if two pieces share an edge. The Procrustes algorithm is applied to find the translations and rotations that best fit shared edges. The method is implemented in Matlab, and tested on a broken spherical surface.

Acknowledgments

I would like to thank Professor Robert Thompson and Yiwen Hu for their vision and contributions to this project.

Contents

Chapter 1

Introduction

The motivation for this project is the work of Hoff and Olver, [8]. Using a curve matching algorithm based on differential invariant signatures, these authors developed an effective technique for the automatic reassembly of jigsaw puzzles. It was suggested by Olver that similar methods could be applied to reassemble surfaces. As possible test data, we obtained scans of a broken ostrich egg, [7], pictured in Figure 1.1. This thesis extends the algorithms of Hoff and Olver to three dimensions with the end goal of automatic reassembly of the broken ostrich egg. Though this goal is not met entirely, our method is successful in assembling artificial test data.

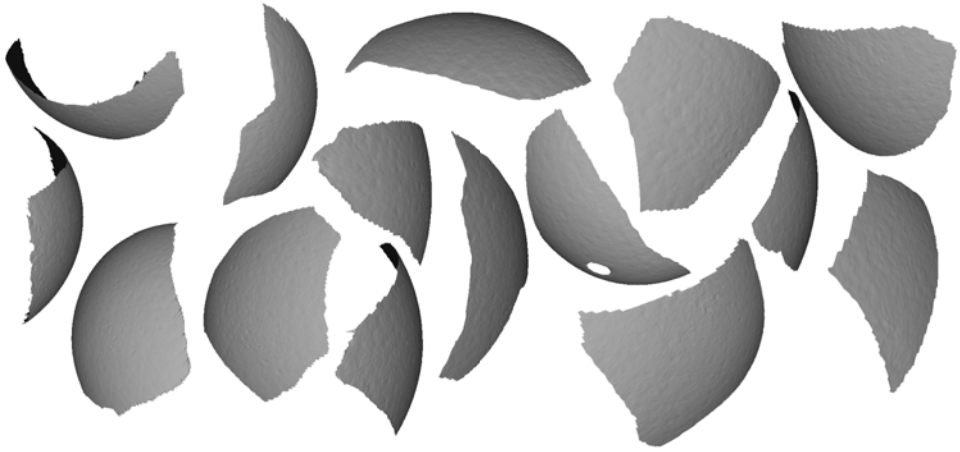

Figure 1.1: A digitized broken ostrich egg

The mathematical strategy for reassembling a broken surface is based on matching the edges or boundaries of the surface. Thus, reassembly becomes a problem of matching up three dimensional curves, like a jigsaw puzzle in space. In practice, the surface pieces may have different orientation and position, so their edges cannot be directly compared. Instead, we will use differential invariant signatures as a measure of comparison. The signature enables us to recognize the

shape of a curve regardless of its orientation and position. Additionally, edges are specified not by a continuous curve, but a discrete set of points. Hence it is necessary to use approximate discrete signatures computed from these points. The discrete signatures are compared via a similarity score to identify if the edges share a common portion. Portions of an edge are determined to be congruent if their signatures are close, and the similarity score is a measure of this closeness. Finally, matched edges must be reassembled via a rotation and translation in space, which is computed via the Procrustes algorithm.

Chapter 2 provides the mathematical background necessary for the method. This includes a discussion of differential invariants for space curves, the differential invariant signature, discrete signature approximation, similarity score computation and the Procrustes algorithm. Chapter 3 is dedicated to implementation. Pseudocode for all steps of implementation is provided in this chapter, and a Matlab implementation is available on Github at the following URL:

`https://github.com/robcth/surfaceReconstruction`

We apply our method to an artificial broken surface consisting of four pieces of a sphere. This application is discussed in Chapter 4. The algorithm successfully matches and reassembles the pieces. In Chapter 5 we discuss difficulties encountered and possible future work.

Chapter 2

Background

This project has roots in several branches of mathematics, including differential and computational geometry, linear algebra, and computer vision. This chapter provides an overview of the mathematical tools that will be used, including *signature*, *similarity score*, and the *Procrustes algorithm*.

The signature, $S = (\kappa, \kappa_s, \tau)$, allows us to be able to compare one piece to another without any concern about orientation and position in space. The signature of an edge is used as the metric of the comparison instead of the edge itself. Because the edges to be compared consist of a discrete set of points, it is necessary to approximate the signature at each point on the edge by a discrete numerical approximation. We first discuss below the construction of the signature and its approximation.

The second stage of this project is to identify matching edges by applying a signature comparison. To do this, the edges are decomposed into smaller arcs, called bivertex arcs. The level of matching of two bivertex arcs is then measured by calculating the *similarity score* of their signatures. The similarity score, which is presented in [2], measures the closeness of two curves. We discuss below in Sections 2.2 and 2.3 the bivertex arc decomposition and similarity score computation.

The final stage is to find the translation and rotation that reassembles matched pieces by a rigid motion reconstruction method known as the *Procrustes algorithm*.

2.1 Signature

The signature S of a three dimensional curve is a curve parametrized by the curvature κ, its derivative with respect to arc length κ_s, and torsion τ. The signature of a two dimensional curve is a special case of the signature of a three dimensional curve where torsion $\tau = 0$.

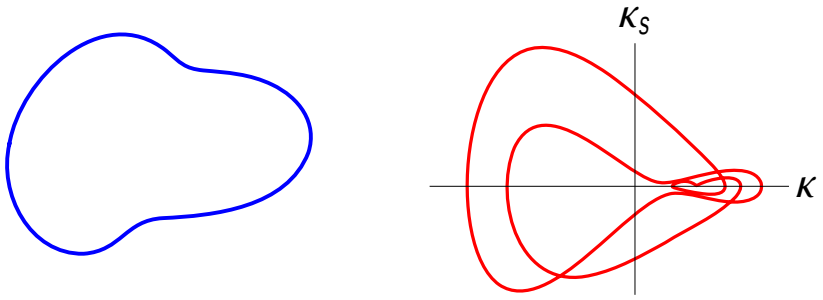

Figure 2.1: A two dimensional curve and its signature

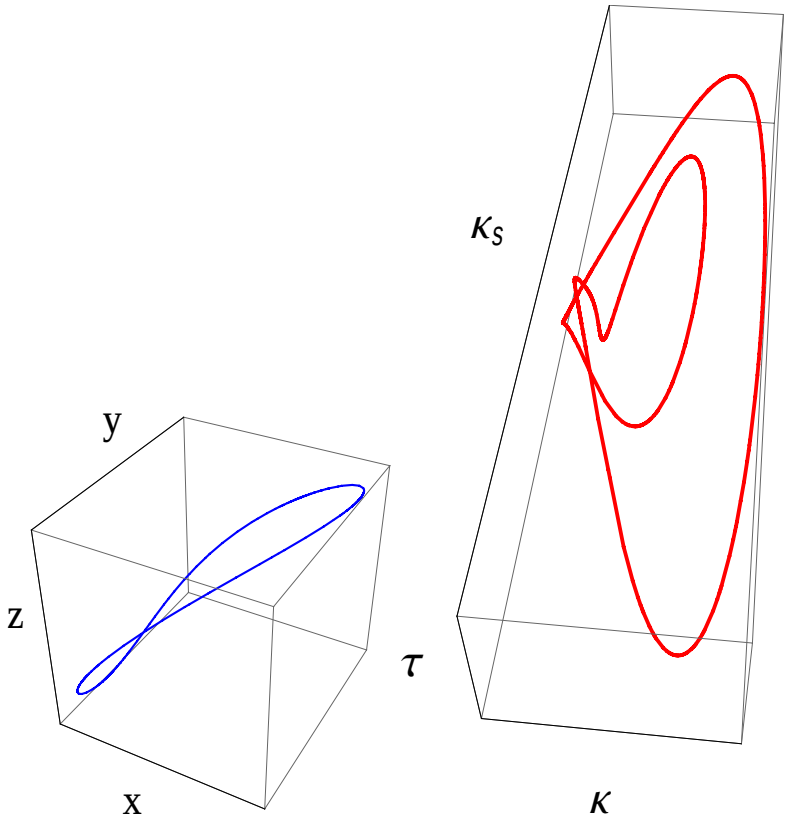

Figure 2.2: A three dimensional curve and its signature

With some restrictions that will be discussed in Section 2.2, it can be shown that two curves are related by a group transformation if and only if their signature curves are identical. A proof of this result appears in [1]. This implies that regardless of their orientation and position, two curves are congruent if and only if their signatures are identical. Since an understanding of the concept of signature requires understanding of curvature κ and torsion τ, we begin with a discussion of this topic.

2.1.1 Curvature and Torsion

Curvature κ measures the bendiness of a curve. Moving along the curve at constant speed, the direction of the tangent vector changes as the curve bends. The curvature measures the magnitude of the rate of change of the tangent vector. The torsion τ measures how sharply the curve twists out of a plane as we move along the curve. Moving along the curve at constant speed, the tangent vector will rotate or twist. The torsion measures the magnitude of this twisting of the tangent vector.

Let $r(t)$ be a parameterized curve representing the position vector of a moving particle. Arc length $s(t)$ of this curve is defined by

$$s(t) = \int_a^t \|r'(\sigma)\| \, d\sigma,$$

where the integration can start at any a in the domain of the parameterization. Then we can write t as a function of s, and write $r(s) = r(t(s))$. The curve $r(s)$ is then parametrized by its arc length. The unit tangent vector T, the unit normal vector N, and the unit binormal vector B are defined as follows:

$$T = \frac{dr}{ds}, \qquad N = \frac{\dfrac{dT}{ds}}{\left\| \dfrac{dT}{ds} \right\|}, \qquad B = T \times N.$$

The unit tangent vector T points in the direction of the curve, the unit binormal vector is in the same plane but perpendicular to T, and the unit normal vector N is perpendicular to both T and B. It is possible to use T, N, and B to move anywhere in three dimensional space with two motions. This is shown in Figure 2.3. One motion measures the rate of change of the unit tangent vector T and another measures the rate of change of the unit binormal vector B.

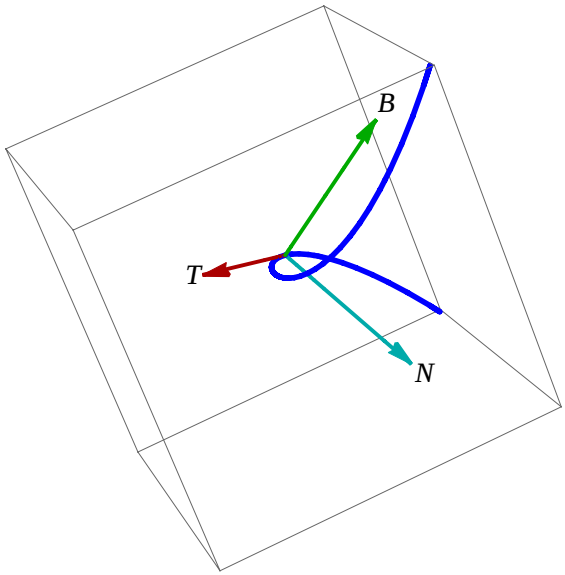

Figure 2.3: T, N, and B as a moving frame

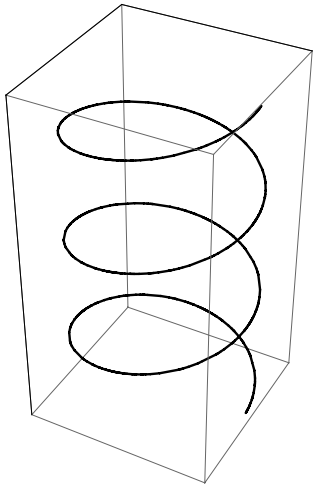

Figure 2.4: A helix, with constant positive curvature and torsion

The rate of change of the unit tangent vector is proportional to the unit normal vector N, and the coefficient of this proportionality is the *curvature* κ. Moreover, the rate of change of the unit binormal vector is also proportional to the unit normal vector N and the coefficient of the proportionality is negative of *torsion* τ.

$$\frac{dT}{ds} = \kappa N, \qquad \frac{dB}{ds} = -\tau N.$$

Example 2.1.1. *If $\tau = 0$, $\kappa = 0$, then the curve is a straight line. If $\tau = 0$, $\kappa = c > 0$, then the curve is a circle. If $\tau = c_1 > 0$, $\kappa = c_2 > 0$, then the curve is*

a helix, and it spirals upward at a constant rate and constant diameter (see Figure 2.4).

2.1.2 Signature Approximations

In practice, the edge curves of each piece are represented by many three dimensional data points, not a continuous curve or formula. Therefore, the signature can only be approximated at each point by a discrete numerical approximation. Our approach to this numerical approximation is described in [1] and [3].

In [1], the authors propose to find numerical approximations for κ and κ_s in terms of *joint invariants* to get less sensitive numerical approximations. A (Euclidean) joint invariant is a function of a collection of points that is independent of orientation and position of this collection in space. For example, the Euclidean distance of two points is a joint invariant of these points, the simplest joint invariant of the action of the Euclidean Group on $\mathbb{R}^3$. The differential invariants $\kappa, \kappa_s,$ and τ are then approximated in terms of the Euclidean distance.

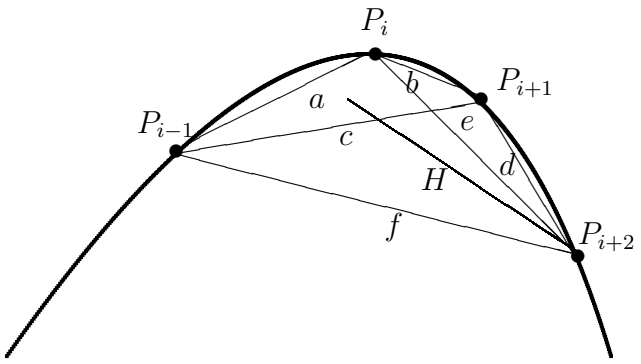

Figure 2.5: Consecutive points for signature approximations

It is shown in [3] that with four consecutive points on a curve, one can compute the numerical approximations for $\kappa, \kappa_s,$ and τ. The approximations of these differential invariants will be denoted $\tilde{\kappa}, \tilde{\kappa_s},$ and $\tilde{\tau}$. In Figure 2.5 four consecutive points, P_{i-1}, P_i, P_{i+1}, and P_{i+2} on a three dimensional curve are shown, together with the Euclidean distances of interest. We then denote by $d(P_i, P_j)$ the Euclidean distance between the point P_i and P_j. Let

$$a = d(P_{i-1}, P_i) \quad b = d(P_i, P_{i+1}) \quad c = d(P_{i-1}, P_{i+1}) \quad d = d(P_{i+1}, P_{i+2})$$
$$e = d(P_i, P_{i+2}) \quad f = d(P_{i-1}, P_{i+2})$$

The formulas used to approximate the signature are as follows:

$$\begin{aligned} \tilde{\kappa}(P_i) &= \frac{4\Delta}{abc} \\ \tilde{\kappa_s}(P_i) &= 3 \cdot \frac{\tilde{\kappa}(P_{i+1}) - \tilde{\kappa}(P_i)}{a+b+d} \\ \tilde{\tau}(P_i) &= 6 \cdot \frac{H}{def\tilde{\kappa}(P_i)}, \end{aligned} \tag{2.1}$$

where Δ is the area of triangle whose sides are a, b, and c (computed using Heron's formula), and H denotes the height of the tetrahedron with sides a, b, c, d, e, and f with respect to P_{i+2}.

To illustrate the method behind these signature approximation formulas, we give a proof of the second formula of (2.1).

Proposition 2.1.2. *The approximation to the derivative of curvature,*

$$\tilde{\kappa_s}(P_i) = 3 \cdot \frac{\tilde{\kappa}(P_{i+1}) - \tilde{\kappa}(P_i)}{a+b+d},$$

converges to the actual derivative of curvature, $\kappa_s(P_i)$, as a, b, d approach 0.

Proof. Begin by writing the approximation of κ at P_i as

$$\tilde{\kappa}(P_i) = \kappa(P_i) + \frac{1}{3}(b-a)\kappa_s(P_i) + \cdots, \tag{2.2}$$

an expansion proved in [1]. From Figure 2.5, we also have

$$\tilde{\kappa}(P_{i+1}) = \kappa(P_{i+1}) + \frac{1}{3}(d-b)\kappa_s(P_{i+1}) + \cdots. \tag{2.3}$$

Taking the difference of the (2.2) and (2.3) we arrive at

$$\tilde{\kappa}(P_i) - \tilde{\kappa}(P_{i+1}) = \kappa(P_i) - \kappa(P_{i+1}) + \frac{1}{3}(b-a)\kappa_s(P_i) - \frac{1}{3}(d-b)\kappa_s(P_{i+1}) + \cdots \tag{2.4}$$

Now we use

$$\kappa_s(P_i) \approx \kappa_s(P_{i+1}) \qquad \text{and} \qquad \kappa(P_{i+1}) - \kappa(P_i) \approx b\kappa_s(P_i)$$

to rewrite (2.4) as

$$\tilde{\kappa}(P_i) - \tilde{\kappa}(P_{i+1}) \approx -\frac{1}{3}(a+b+d)\kappa_s(P_i).$$

□

The proof of the τ formula is more complex, and the reader is encouraged to investigate [3] for further discussion of this and the other formulas from (2.1).

A comparison of the actual signature of the curve pictured in Figure 2.2 and the discrete approximation of this signature is shown in Figure 2.6.

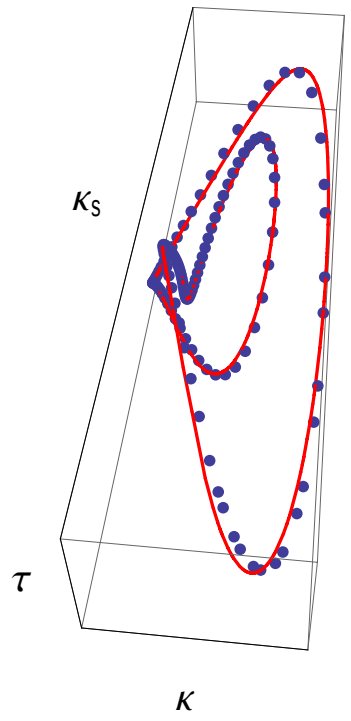

Figure 2.6: Actual signature versus discrete approximate signature

2.2 Bivertex Arcs

We will be using the closeness of the signatures of two edges as a metric of comparison. However, there are two issues that need to be addressed before applying this comparison. The first is that two pieces share only a portion of their edges, so we cannot simply compare the signatures of entire edges.

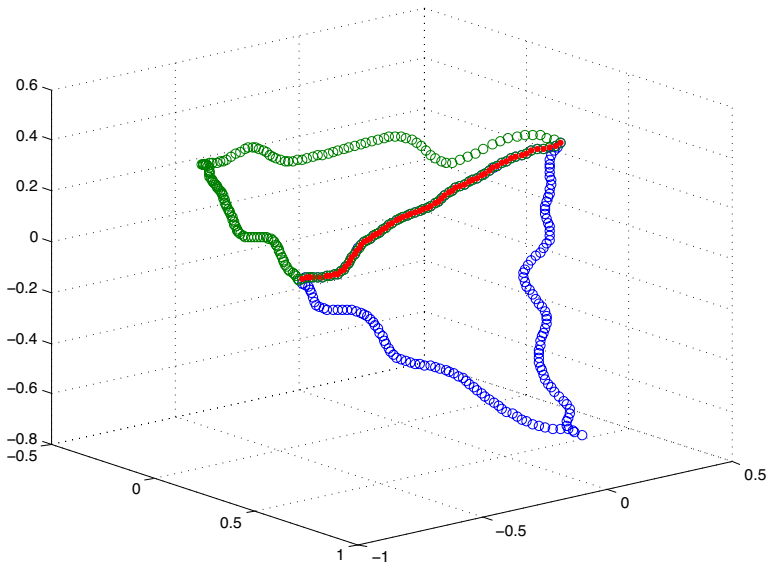

Figure 2.7: Two pieces share a common edge

The second is that it is possible to construct some closed curves that are not congruent, but have identical signature. This implies that with just information of the signature, we may not know entirely the shape of the curve. A method of constructing these curves is described in [4]. A family of non-congruent curves with identical signature is shown in Figure 2.8. The construction of these non-congruent curves motivates the method of segmentation into *bivertex arcs*, as first described in [2]. Consider the situation when a curve contains a portion of a line or circle. The curvature and torsion of line or circle is a constant, so their signatures consists of only a point. We then extend the length of the line or circle, which does not change the signature, but creates a non-congruent curve.

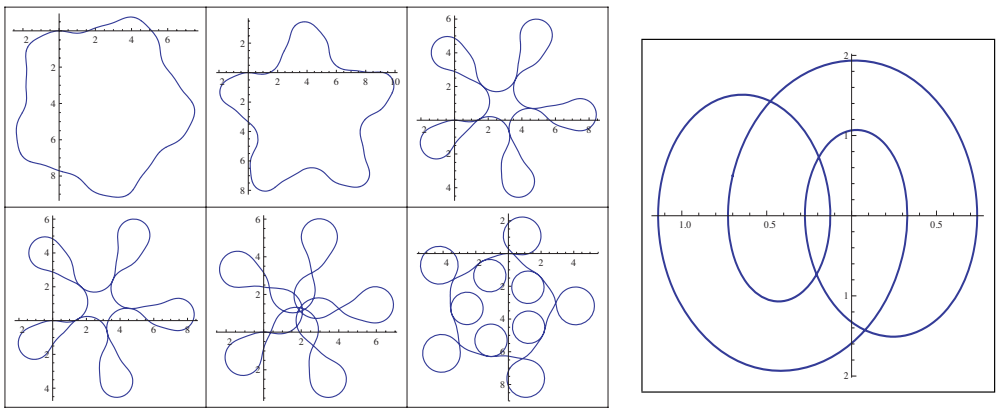

Figure 2.8: A family of curves and their common signature, [4]

Therefore, we want to decompose our curves into smaller arcs, none of which

contains a portion of a line or circle. This decomposition also helps us to create smaller pieces of the edge that will be compared. We can eliminate any lines or circles on the edge by excluding portions of the curve that have $\kappa_s = 0$. These smaller arcs on which $\kappa_s \neq 0$ are called *bivertex arcs.*

Definition 2.2.1. *A bivertex arc is an arc of a curve for which $\kappa_s = 0$ at the endpoints and $\kappa_s \neq 0$ at all other points.*

Theorem 2.2.2. *Two bivertex arcs are congruent if and only if their signatures are identical.*

Bivertex arcs provide both a practical and a theoretical advantage. In theory, two bivertex arcs are congruent if and only if their signatures are identical. The proof of this result for planar curves appears in [1]; for three dimensional curves the proof is very similar. In practice, it is difficult to compare bivertex arcs directly because the numerical approximation of κ_s never takes on the exact value of zero. However, if there is a portion where κ_s is approximately zero, κ_s may change signs. Therefore, in practice we check if κ_s changes sign instead of $\kappa_s = 0$. Using this approach, we compare the arcs that contain the most "curvature information", i.e. those whose curvature is changing the most.

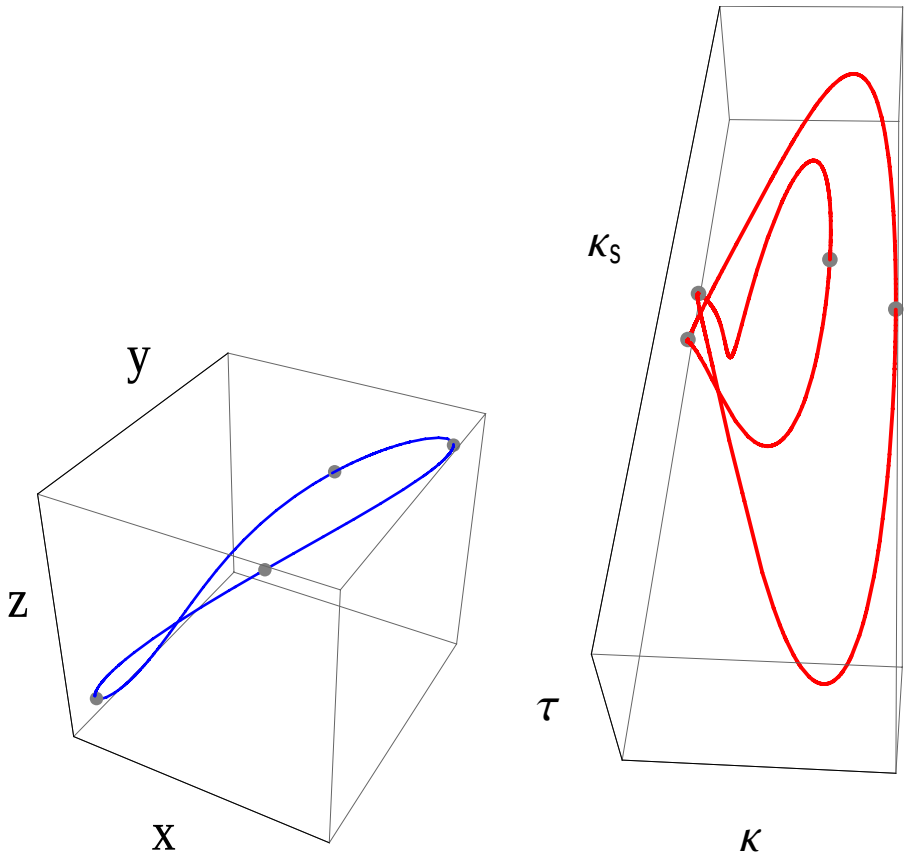

Figure 2.9: A curve and its signature with bivertex arc decomposition marked

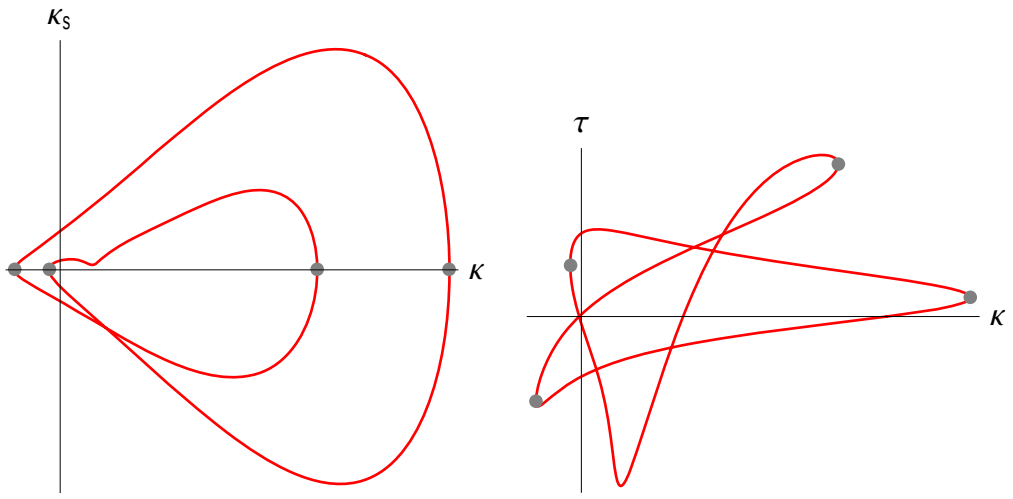

Figure 2.10: Two projections of the signature with bivertex arcs marked

2.3 Curve Similarity Score

The core of the surface reconstruction method is to identify if two pieces share a common edge by computing the similarity score of the signatures of their edges. The similarity score measures the closeness of two curves on a scale from 0 to 1. The approach described here for computing the similarity score is developed in [2].

2.3.1 Terminology and intuition

Before getting to the technical details of the *similarity score*, it is worth understanding the intuitive idea first. Suppose we want to compare two curves, which we will call the p-curve and the q-curve. Each curve is represented by a set of three dimensional data points. We measure closeness by comparing each point of the p-curve to each point on the q-curve. The intuitive idea is: for each point on the p-curve, if there is a point on the q-curve that is close to it, we should get a good *similarity score*. Therefore, we iterate through each point on the p-curve and measure the *strength* of comparison of that point to each point on the q-curve.

The *strength* of comparison of two points is inversely proportional to their distance. The closer the two points, the larger their *strength* is. If they overlap, their *strength* is ∞. Denote the *strength* of comparison of points p_i and q_j as $h(p_i, q_j)$. For each fixed point of the p-curve, we find overall strength, which is the sum of all strengths of that point to points on the q-curve. The overall strength is very large if there exists a point of q-curve that is very close to the fixed point of p-curve. We then rescale the overall strength, which lies in $[0, \infty]$, to lie in $[0, 1]$. The similarity score of the p-curve and the q-curve is then the average over all

points on the p-curve of the rescaled overall strength to each point on the q-curve.

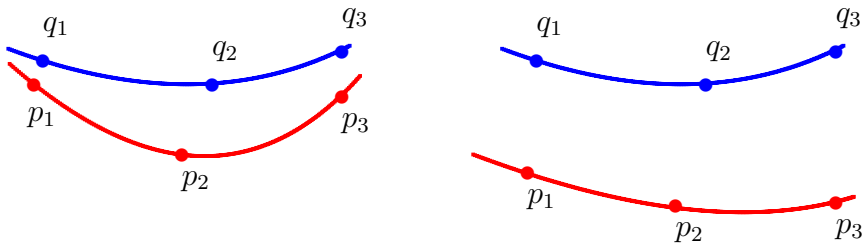

Figure 2.11: Two pairs of curves to be compared

Consider the two pairs of curves in Figure 2.11. For the left pair, $h(p_1, q_1)$ is large because p_1 is very close to q_1. However, $h(p_1, q_2)$ and $h(p_1, q_3)$ are smaller because their distances are larger. Then the *overall strength* of p_1 to the q-curve, defined to be the sum of the point-wise strengths, is large:

$$h(p_1, q\text{-curve}) = \sum_j h(p_1, q_j) \text{ is large.}$$

In general, if there exists a point on the q-curve close to p_1, $h(p_1, q\text{-curve})$ will be large. In contrast, for the second pair of curves in Figure 2.11, comparing p_1 to the q-curve results in a smaller overall strength:

$$h(p_1, q\text{-curve}) = \sum_j h(p_1, q_j) \text{ is small.}$$

Next, we rescale the overall strength to be in $[0, 1]$. The average over all points on the p-curve of the rescaled overall strength measures the *similarity score*, s, of the two curves:

$$s(p\text{-curve}, q\text{-curve}) = \underset{i}{\text{average}} \left(\text{rescaled } h(p_i, q\text{-curve})\right).$$

One can see that the average of the overall strengths for the pair of curves on the left will be larger than the pair on the right because, overall, more p-curve points are close to the q-curve. As a final consideration, since the similarity score is not symmetric, we also compute $s(q\text{-curve}, p\text{-curve})$ and take the minimum of the two values as our final similarity score.

2.3.2 Similarity Score Algorithm

The procedure of computing the similarity score is based on the intuitive ideas above. However, its implementation is more complex due to choices that can be made regarding the scale of comparison, the elimination of instances of division by zero, and the shape of the rescaling function. For each of these choices there are parameters that will adjust the value of the similarity score, and these parameters should be chosen so that the similarity score provides the most effective comparison.

The similarity score is described as a metric to measure the closeness between two signature curves. For this purpose, suppose that we wish to compare two curves B_1 and B_2. The approach to computing the similarity score has four ingredients: the *scale of comparison*, *separation*, *strength*, and *rescaling*.

We first define the *scale of comparison* D. The *scale of comparison* measures the maximal variation of curvature along the two curves and is computed via the following formula:

$$D = \max\left\{ \max(\kappa \in B_1) - \min(\kappa \in B_1), \max(\kappa \in B_2) - \min(\kappa \in B_2) \right\}.$$

We then compute the *separation* between two points p and q, where $p \in B_1$ and $q \in B_2$. Essentially, the separation measures the closeness between two points. However, it also checks to see if the Euclidean distance between p and q is not greater than or equal to the scale of comparison. The scale of comparison is used to serve as the cut-off value, and scale down the separation of two points. The separation between two points p and q is then defined as follows:

$$d(p,q) = \begin{cases} \dfrac{\parallel p - q \parallel}{D - \parallel p - q \parallel} & \text{if } \parallel p - q \parallel < D \\ \\ \infty & \text{if } \parallel p - q \parallel \geq D \end{cases}$$

Now we can compute the *strength*. The idea is that the closer the two points, the larger their *strength*. The *strength* is inversely proportional to some power of their separation. The *strength* of two points, p and q, is denoted $h(p,q)$:

$$h(p,q) = \begin{cases} \dfrac{1}{d(p,q)^{\gamma} + \epsilon} & \text{if } d(p,q) < \infty \\ \\ 0 & \text{if } d(p,q) = \infty \end{cases}$$

where $\gamma > 0$ is fixed, and $\epsilon > 0$ is a small constant. The ϵ is a cut-off to avoid infinities when p and q coincide. The constant γ is used to scale the *strength*.

Larger values of γ cause the strength to drop more quickly to zero as *separation* increases.

Next we compute the sum of the *strength* over q for fixed p and scale the result to $[0, 1]$ using a rescaling function $r(t)$. Then the final *similarity score* of two curves is the average over p of the rescaled values.

$$s(B_1, B_2) = \operatorname*{average}_i \left(\sum_j r\left(h(p_i, q_j)\right) \right) \tag{2.5}$$

A reasonable rescaling function is needed to make the curve similarity score lie in [0, 1]. The rescaling function used in our algorithm is

$$r(t) = \frac{t}{t + B}, \qquad 0 \le t \le \infty.$$

The parameter $B > 0$ affects the distribution of the score along the interval $[0, 1]$.

2.4 Procrustes Algorithm

After identifying that two pieces share a common edge, the next step is to reassemble these pieces. These pieces may have different orientation and position, so a rotation and translation needs to be computed that best aligns the pieces in space. To compute the translation, the centroid (center of mass) of each piece is computed, and the translation is simply the difference of the two centroids. To find the rotation, the centroids of both pieces are placed at the origin, and a rotation minimizing the inter-point distances of shared edges is computed. This procedure is pictured in Figure 2.12. All of this may be done using the Procrustes algorithm.

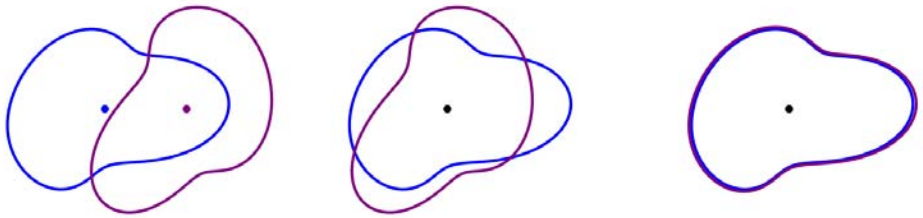

Figure 2.12: Translation and rotation to fit shapes together

Assume that we have two $n \times 3$ matrices A and B, representing collections of points in $\mathbb{R}^3$. The question to be solved is: how closely can B be rotated into A? This question is answered by solving the optimization problem:

$$\operatorname*{minimum}_Q \| A - BQ \|, \quad \text{where } Q^T Q = I.$$

The solution to this problem is a rotation matrix Q that "best" rotates B into A. This problem is generally known as the Procrustes problem.

For the practical use of the Procrustes algorithm, the matrices A and B represent the collections of points on matching curves. The first step is to translate both curves such that their centroids are at origin. We then solve the Procrustes problem for these translated curves. This problem can be solved simply via the Singular Value Decomposition (SVD), as later described in Algorithm 3.3.1.

Chapter 3

Methods

In this chapter we develop the implementation of the surface reconstruction method, which is presented in the form of pseudocode. The implementation is based on the work of [2] for computing *signature similarity scores* and [3] for the numerical approximation of differential invariants κ, κ_s, and τ. As described in Section 2.2, the bivertex arc decomposition is applied to each edge to create two sets of bivertex arcs. The signatures of the bivertex arcs are used in the comparison. After identifying possible matching pairs, these pairs are reassembled using the Procrustes algorithm, for which pseudocode is also presented. The method is implemented in Matlab.

3.1 Computing Approximations of Signatures

In this section, we give pseudocode for the implementation of the numerical approximations of differential invariants κ, κ_s, and τ using the formulas from Section 2.1.2. The differential invariants are approximated using only Euclidean distances in $\mathbb{R}^3$. Therefore, no result of the computation depends on orientation or position in space. The formulas for the signature approximation are described in Section 2.1.2 and pictured in Figure 2.1.

Denote by P_i a point on a curve, $dist(P_i, P_j)$ the distance between point P_i and P_j, and $A(P_i, P_j, P_k)$ the area of triangle with vertices P_i, P_j, and P_k. We can compute the numerical approximation of κ of a point with three consecutive points provided. The pseudocode is presented below.

Algorithm 3.1.1: The numerical approximation of κ

Data: Given three discrete points $P_{i-1}, P_i,$ and P_{i+1}
Result: Return $\tilde{\kappa}$ at point P_i
Compute $a = dist(P_{i-1}, P_i)$
Compute $b = dist(P_i, P_{i+1})$
Compute $c = dist(P_{i-1}, P_{i+1})$
Compute $\Delta = A(P_{i-1}, P_i, P_{i+1})$ using Heron's formula
Compute $\tilde{\kappa}(P_i) = \dfrac{4\Delta}{abc}$

To compute the numerical approximation of κ_s at a point, four consecutive points are required. The pseudocode is presented below.

Algorithm 3.1.2: The numerical approximation of κ_s

Data: Given four discrete points $P_{i-1}, P_i, P_{i+1},$ and P_{i+2}
Result: Return $\tilde{\kappa}_s$ at point P_i
Compute $a = dist(P_{i-1}, P_i)$
Compute $b = dist(P_i, P_{i+1})$
Compute $d = dist(P_{i+1}, P_{i+2})$
Compute $\tilde{\kappa_s}(P_i) = 3 \cdot \dfrac{\tilde{\kappa}(P_{i+1}) - \tilde{\kappa}(P_i)}{a + b + d}$

To compute the numerical approximation of τ at a point, four consecutive points are required. The pseudocode is presented below.

Algorithm 3.1.3: The numerical approximation of τ

Data: Given three discrete points $P_{i-1}, P_i, P_{i+1},$ and P_{i+2}
Result: Return $\tilde{\tau}$ at point P_i
Compute $a = dist(P_{i-1}, P_i)$
Compute $b = dist(P_i, P_{i+1})$
Compute $c = dist(P_{i-1}, P_{i+1})$
Compute $d = dist(P_{i+1}, P_{i+2})$
Compute $e = dist(P_i, P_{i+2})$
Compute $f = dist(P_{i-1}, P_{i+2})$
Compute $\Delta = A(P_{i-1}, P_i, P_{i+1})$ using Heron's formula
Compute V, the volume of pyramid base $P_{i-1}, P_i,$ and P_{i+1}
Compute height, $H = 3 \cdot \dfrac{V}{\Delta}$
Compute $\tilde{\tau}(P_i) = 6 \cdot \dfrac{H}{def\tilde{\kappa}(P_i)}$

These methods are used to compute the approximations of the signature at each

point on the edge curve of interest. It then produces a discrete signature curve, $S \subset \mathbb{R}^3$, used to perform similarity score computation.

3.2 Comparing Similarity Scores of Signatures

In this section, we will cover the implementation of the similarity score. The similarity score computation is applied to the signatures of the bivertex arcs. As previously mentioned, the edges are decomposed into smaller arcs on which $\kappa_s \neq 0$. In practice this is done by checking if $\tilde{\kappa}_s(P_i)$ and $\tilde{\kappa}_s(P_{i+1})$ change sign. If they do, P_i defines an endpoint of an arc. We iterate through all points, and decompose the edge using this method.

Algorithm 3.2.1: Curve decomposition

Data: Given an edge C_1
Result: Return a set of bivertex arcs
initialize $set\_arcs = \{\}$ (cell array in Matlab) ;
compute $kappa_s = numerical\ approximations\ of\ \kappa_s$;
initialize $number\_arcs = 1$;
initialize $length\_of\_arc = 1$;
for $p_i \in C_1$ **do**
    $set\_arcs\{number\_arcs\}(length\_of\_arc) = p_i$;
    if $kappa_s(p_i) \cdot kappa_s(p_{i+1}) < 0$ **then**
        $number\_arcs = number\_arcs + 1$;
        $length\_of\_arc = 1$;
    else
        $length\_of\_arc = length\_of\_arc + 1$;

Then each edge is decomposed into a set of arcs. Comparison of edges is done by comparison of sequences of these arcs. If two edges match together, they will both contain some number of arcs whose signatures are very close, as measured by the similarity score. Next we discuss the implementation of this similarity score.

3.2.1 Procedure of Similarity Score Computation

We first implement the scale of comparison of two arcs as described in Section 2.3.2. The method takes as inputs the sets of discrete signatures B_1 and B_2 of two

arcs and outputs the *scale of comparison* $D(B_1, B_2)$.

Algorithm 3.2.2: The *scale of comparison*

Data: Given two signatures, B_1 and B_2
Result: return *scale of comparison*, D
initialize $\mathit{first\_max\_kappa} = max(\kappa \in B_1)$;
initialize $\mathit{first\_min\_kappa} = min(\kappa \in B_1)$;
initialize $\mathit{second\_max\_kappa} = max(\kappa \in B_2)$;
initialize $\mathit{second\_min\_kappa} = min(\kappa \in B_2)$;
$D = max(\mathit{first\_max\_kappa} - \mathit{first\_min\_kappa}, \mathit{second\_max\_kappa} - \mathit{second\_min\_kappa})$;

We precompute the *scale of comparison* D, which will be used in the *similarity score* computation. Next we describe the measurement of the *separation* between points from B_1 and B_2. Pseudocode is given below.

Algorithm 3.2.3: The *separation* value of two points

Data: Given two points, $p \in B_1$ and $q \in B_2$
Result: return *separation*
Compute $\mathit{pointDiff} = dist(p, q)$;
initialize $D = D(B_1, B_2)$;
if $\mathit{pointDiff} < D$ **then**
    $separation = \dfrac{\mathit{pointDiff}}{D \text{ - } \mathit{pointDiff}}$;
else
    $separation = \infty$;

Next we describe the measurement of the *strength* of separation between points from B_1 and B_2 as defined in Section 2.3. The pseudocode is given below.

Algorithm 3.2.4: The *strength* value of two points

Data: Given two discrete points, $p \in B_1$ and $q \in B_2$
Result: return *strength* of p and q
initialize $\gamma = 0.5$;
initialize $\epsilon = 10^{-5}$;
initialize $separation = separation(p, q)$;
if $separation < \infty$ **then**
    $strength = \dfrac{1}{separation^{\gamma} + \epsilon}$;
else
    $strength = 0$;

Then we can compute the *similarity score* of the signatures, B_1 and B_2. The *similarity score* method takes two signatures B_1 and B_2 as the inputs, and returns their *similarity score*. The rescaling function is required to make the *similarity*

score lie in [0, 1].

Algorithm 3.2.5: The *rescaling* function

Data: Given a value, t
Result: return a *rescaled value*, r
initialize $B = 1000$;
$r = \dfrac{t}{t+B}$;

The parameter $B > 0$ affects the distribution of the score along the interval $[0, 1]$. In our case, we use $B = 1000$ as tested in [2]. Choosing larger values of B makes the score transition more slowly from 0 to 1.

The pseudocode for the *similarity score* computation is given below.

Algorithm 3.2.6: The *similarity score* of two signatures

Data: Given two signatures, B_1 and B_2
Result: return *similarity score*, the closeness of two signatures
initialize $\mathit{first\_score} = 0$;
initialize $\mathit{total\_strength} = 0$;
for $p_i \in B_1$ **do**
    for $q_j \in B_2$ **do**
        $\mathit{total\_strength} = \mathit{total\_strength} + \mathit{strength}(p_i, q_j)$;
    $\mathit{first\_score} = \mathit{first\_score} + \mathit{rescaling}(\mathit{total\_strength})$;
    $\mathit{total\_strength} = 0$;
$\mathit{first\_score} = \dfrac{\mathit{first\_score}}{\mathit{size}(B_1)}$;
initialize $\mathit{second\_score} = 0$;
initialize $\mathit{total\_strength} = 0$;
for $q_j \in B_2$ **do**
    for $p_i \in B_1$ **do**
        $\mathit{total\_strength} = \mathit{total\_strength} + \mathit{strength}(q_j, p_i)$;
    $\mathit{second\_score} = \mathit{second\_score} + \mathit{rescaling}(\mathit{total\_strength})$;
    $\mathit{total\_strength} = 0$;
$\mathit{second\_score} = \dfrac{\mathit{second\_score}}{\mathit{size}(B_2)}$;
$\mathit{similarity\_score} = \mathit{min}(\mathit{first\_score}, \mathit{second\_score})$;

As mentioned, each edge is represented by a set of arcs. Therefore, when comparing two edges, we need to compare each arc of the first edge to each arc of another edge. First, we construct a table to store the *similarity score* of each pair of arcs. Then we find the *highest similarity score* in this table. From experiments, we define the threshold of 0.90. If the *highest similarity score* ≥ 0.9, we conclude that these two edges share a common arc. Next we trace upward and downward along

the diagonal of this table starting from the index with the *highest similarity score*. We want to find a sequence of pairs with *similarity score* ≥ 0.9. Then we include these arcs in the matching part. This is because the matching part should be a sequence of consecutive arcs in the table.

Algorithm 3.2.7: Construct similarity score table for two sets of arcs

Data: Given two sets of bivertex arcs, *bivertex_arcs_1* and *bivertex_arcs_1*
Result: Return *table*, similarity score table
initialize *table* = [];
for $first\_arc_i \in$ *bivertex_arcs_1* **do**
    for $second\_arc_j \in$ *bivertex_arcs_2* **do**
        table(i, j) = $similarity\_score(first\_arc_i, second\_arc_j)$

Algorithm 3.2.8: The procedure of finding the matching part of two edges

Data: Given two edges C_1 and C_2
Result: Return $\tilde{\kappa}_s$ at point P_i
initialize *bivertex_arcs_1* = all bivertex arcs of C_1;
initialize *bivertex_arcs_2* = all bivertex arcs of C_2;
initialize *score_table* = *table*(*bivertex_arcs_1, bivertex_arcs_2*);
initialize *max_score* = *max*(*score_table*);
initialize *(r_max, c_max)* = *row* and *column* of *max_score*;
initialize *threshold* = 0.9;
initialize *start_row* = *r_max*;
initialize *start_col* = *c_max*;
initialize *end_row* = *r_max*;
initialize *end_col* = *c_max*;
if *max_score* $\geq$ *threshold* **then**
    while *score_table(start_row, start_col)* $\geq$ *threshold* **do**
        start_row = *start_row* − 1;
        start_col = *start_col* − 1;
    while *score_table(end_row, end_col)* $\geq$ *threshold* **do**
        end_row = *end_row* + 1;
        end_col = *end_col* + 1;

3.2.2 Example of Similarity Score Computation

In this section the signature and similarity score computation applied to three pairs of test curves. We begin by generating a small segment of a curve, then we add different levels of noise to create another curve for the comparison.

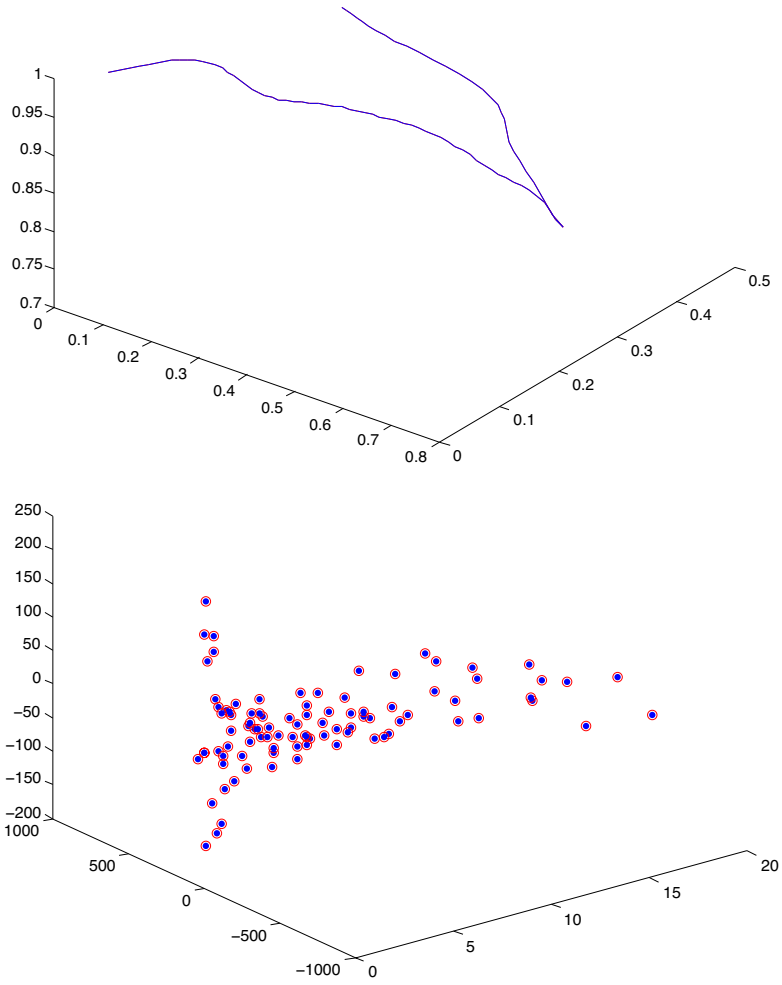

Figure 3.1: Curves with identical signature

The two identical curves in Figure 3.1 have identical signatures, each shown as blue and red points. It may be difficult to see from the figure, but the blue and red points overlap one another in space. Computing the bivertex arc decomposition and similarity score of these identical signatures produces a highest score of 0.9934 between bivertex arcs. The score is above the threshold, so these two curves would be said to share an edge.

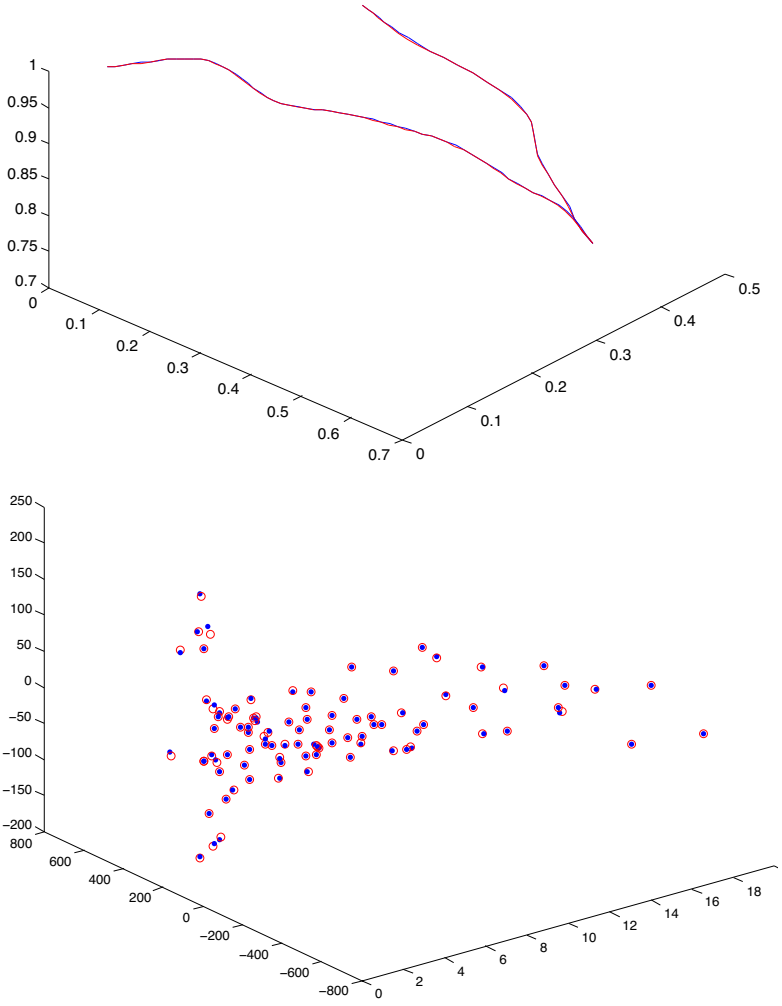

Figure 3.2: Curves with almost identical signature

The two curves in Figure 3.2 have almost identical signature: the blue points and red points are generally very close to each other in three dimensional space. Computing the bivertex arc decomposition and similarity score of these nearly signatures produces a highest score of 0.95 between bivertex arcs. The score is above the threshold, so these two curves would be said to share an edge.

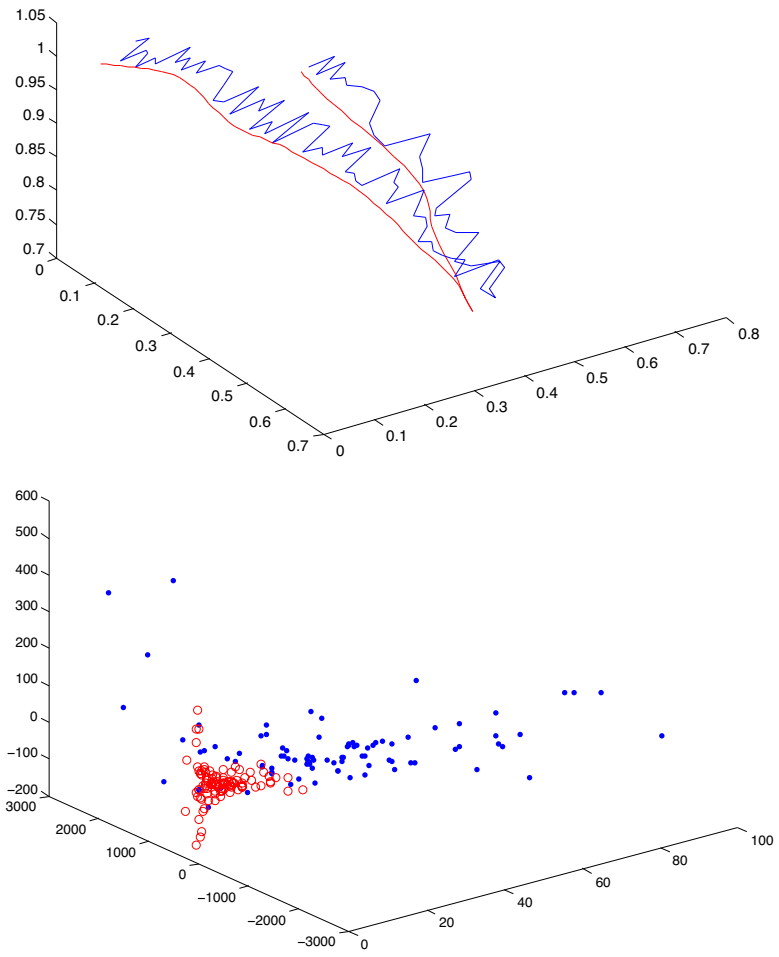

Figure 3.3: Curves with different signature

The two curves in Figure 3.3 have clearly different signatures. The blue points and red points are far away from each other, and they seem to split into two different groups based on the magnitude of their values. Computing the bivertex arc decomposition and similarity score of these nearly signatures produces a highest score of 0.66 between bivertex arcs. The score is below the threshold, so these two curves would not be said to share a common edge.

3.3 Reassembling Broken Pieces

The final stage of the surface reconstruction is to reassemble each piece of a broken surface. The matching pieces are in different orientation and position in space, so we need to translate and rotate until their matching part overlaps. The Procrustes algorithm solves this problem. We have already described the Procrustes algorithm

in Section 2.4, and we will implement it here.

Algorithm 3.3.1: Algorithm to find rotation [5]

Data: Given A and B $\in \mathbb{R}^{m \times n}$
Result: Return a rotation matrix Q
Compute $C = B^T A$
Compute the SVD $U^T C V = \sum$ and save U and V.
Compute $Q = UV^T$

Algorithm 3.3.2: Procrustes Algorithm

Data: Given *common edges* from two pieces, $common\_edge_1$ and $common\_edge_2$
Result: return *first_centroid, second_centroid, rotation*
initialize $first\_centroid = [0; 0; 0]$;
for $p_i \in common\_edge_1$ **do**
    $first\_centroid = first\_centroid + p_i$;
initialize $second\_centroid = [0; 0; 0]$;
for $q_j \in common\_edge_2$ **do**
    $second\_centroid = second\_centroid + q_j$;
$first\_centroid = \dfrac{first\_centroid}{size(common\_edge_1)}$;
$second\_centroid = \dfrac{second\_centroid}{size(common\_edge_2)}$;
compute $C = common\_edge_1 \cdot transpose(common\_edge_2)$;
compute **SVD** $U^T C V = \sum$ and save U and V;
$rotation = U \cdot transpose(V)$;

The Procrustes algorithm reassembles two pieces at a time. The method takes two inputs, the matching part of the first piece and the matching part of the second piece. It then finds the centroid of each common edge, and computes the rotation that maps $common\_edge_2$ to $common\_edge_1$. The *first_centroid*, *second_centroid*, and *rotation* then are used to reassemble two pieces. To reassemble two pieces, we first translate each piece to another using the centroid values from the Procrustes algorithm. Then we apply the rotation to rotate one piece to another, so that the matching part overlaps.

Chapter 4

Application

In this chapter, we apply our surface reconstruction method to an artificial broken surface. The artificial broken surface is generated in Matlab using Bèzier curves to generate curved edges, and a basic transformation to map these broken edges to the surface of a sphere. The algorithm successfully reassembles all pieces, and we discuss these results below.

4.1 Artificial Surface Data

We begin our experiment by generating an artificial surface broken into four pieces along smooth edges. The process of creating this surface is illustrated in Figure 4.1. Five points ("pins") are chosen inside a unit disk in $\mathbb{R}^2$. Between each of these pins a Bèzier curve is created using 18 randomly spaced control points. The result is a planar "puzzle", shown in the left of the figure. This puzzle is mapped to the sphere using the transformation

$$(x, y) \mapsto (x, y, \sqrt{1 - x^2 - y^2}).$$

To mimic real data some noise is added, and random rotations and translations are applied to scramble these four pieces in space as pictured in Figure 4.2.

These pieces of broken surface are used to test our surface reconstruction method. Bivertex arc decompositions are computed, and two pieces are compared at a time to see if they possess common bivertex arcs with high similarity score. After identifying those sequences of arcs with high similarity, the Procrustes algorithm is applied to reassemble the arcs (and together with them the rest of the piece).

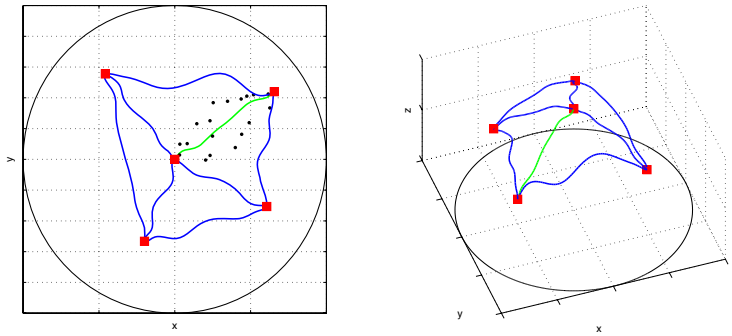

Figure 4.1: A planar "puzzle" and the broken spherical surface created from it

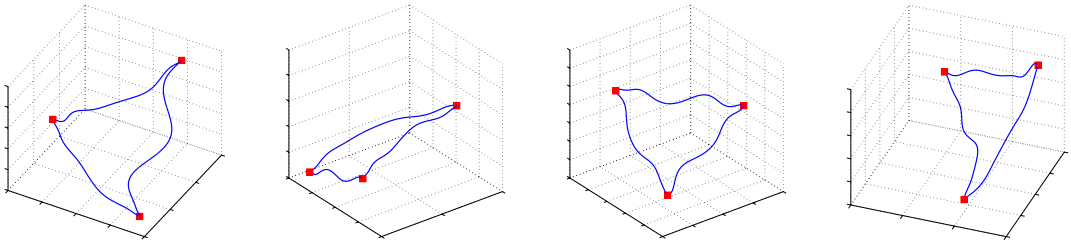

Figure 4.2: Each surface piece scattered in space

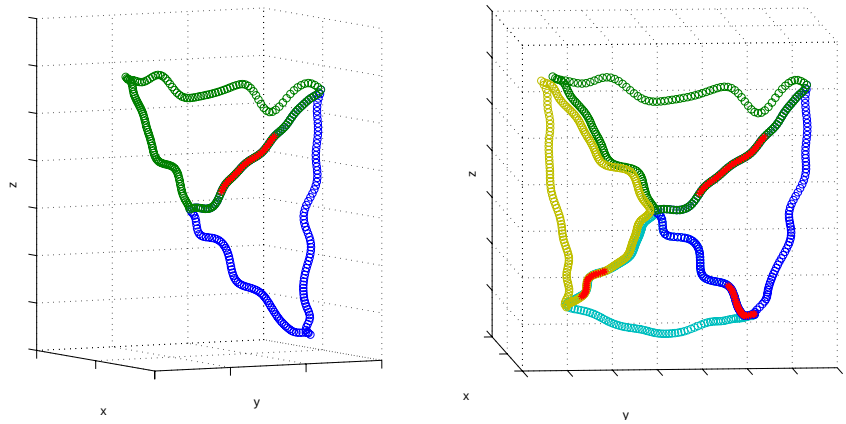

Figure 4.3: Two matched pieces and the complete reassembled surface

The end result of the reassembly is pictured in Figure 4.3. The left part of the figure shows two pieces of the broken surface reassembled. The red highlight along the matching edges is the sequence of matching bivertex arcs as found by the similarity score procedure. The Procrustes algorithm is applied to this highlighted

portion of the matching edges. In the right part of the figure is shown the final result with all pieces reassembled. It may be noticed that not all pieces are perfectly aligned. Because the Procrustes is applied to only two pieces at a time, there may be small errors that accumulate as assembly proceeds.

Chapter 5

Conclusion and Future Work

The major effort in this project was devoted to implementation and testing of the signature computation and comparison. The algorithm was shown to be effective at reassembling artificial broken surface data, but has not yet been applied to real data obtained from 3d scanning or other methods. To this end, we are currently working to apply our method to 3d scans of the broken ostrich egg, [7], discussed earlier and shown in Figure 1.1.

Each piece consists of a many points in space, i.e. a 3d point cloud. The strategy for reassembly consists of extracting the boundary curve of each piece, then applying our surface reconstruction method to these these boundary curves. The boundaries may be extracted using the free boundary of a surface triangulation of the egg piece, as shown in Figure 5.1. However, the boundaries of this broken egg are very rough. Rough or noisy data produces unusable differential invariants, so a smoothing strategy needs to be adopted before our algorithm can be applied. The naive approach is to use a smoothing spline, such as the Matlab function `spaps`. With a small smoothing parameter, smoothing splines appear to approximate the egg boundary without showing the same amount of noise, as can be seen in Figure 5.2. It remains to be seen whether these approximate but smoother boundaries are sufficient to reassemble the egg pieces. It may be worth experimenting with other ways of smoothing the rough data, such as curvature flow.

Another area deserving further study is the method for decomposing curves before applying signature comparison. We chose to use $\kappa_s \neq 0$, i.e bivertex arcs, for the decomposition because it helps avoid degenerate or less useful portions of the signature, and because the decomposition does not depend on position or orientation in space. In practice, the decomposition should be chosen so that the resulting arcs are large enough for meaningful comparison, but not so large as to overlap with other pieces. Another strategy for decomposition is to look for maximal points of curvature or torsion, called curvature "codons", [6].

It is also worth refining the approach to the similarity score. In this thesis

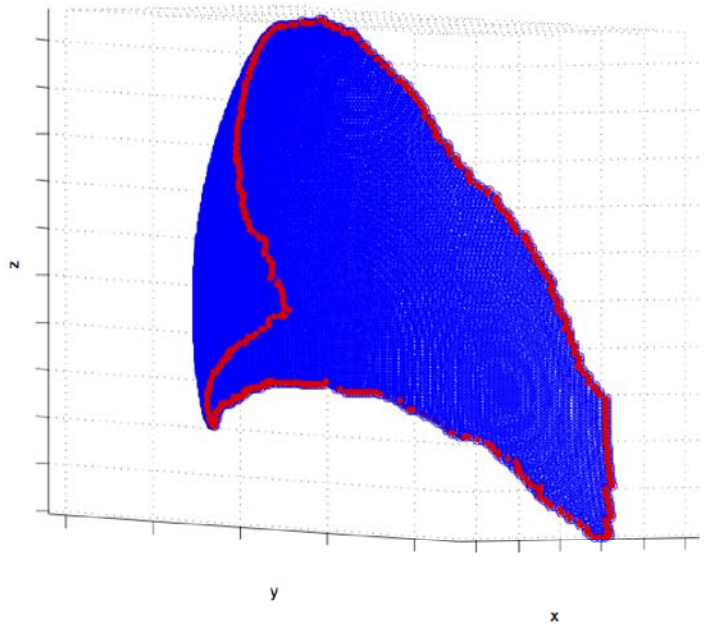

Figure 5.1: An egg piece and its approximate boundary

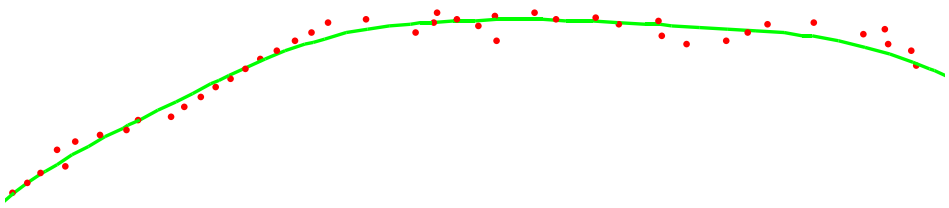

Figure 5.2: Smoothing spline interpolation of a portion of egg boundary

we adapt the signature comparison strategy from [2]. It can be seen in Section 2.3.2 that there are many parameters γ, R, ϵ, etc. introduced to avoid numerical problems and to make the most effective comparison. These parameters have been chosen and adjusted *ad hoc*. Perhaps a more thorough study, based on a collection of curve examples, could improve the selection of these parameters. Completely different approaches to comparison may also improve the algorithm. For example, the range of values of curvature or torsion could provide a quick conclusion that two curves are not congruent.

Finally, as can be seen in Figure 4.3, small errors resulting from assembling edges using the Procrustes algorithm can accumulate as more edges are assembled. To get rid of this error, small adjustments could be made after the coarse assembly of two pieces. This idea, called "piece locking", was implemented in [8]. A three dimensional version of piece locking would ensure a more tightly assembled surface.

Through *numerical signature approximations*, computation of *similarity score*

and the *Procrustes algorithm*, we were able to develop a mathematical strategy for reconstructing a broken surface. This method was successful in providing enough information to reassemble four pieces of a spherical surface. However, as we have described, there is much more work to be done to make the method more robust, and to achieve the goal of reconstructing a real broken egg.

Bibliography

[1] Calabi, E., Olver, P.J., Shakiban, C., Tannenbaum, A. and Haker, S., "Differential and numerically invariant signature curves applied to object recognition", Int. J. Comput. Vision, 26 (1998): 107-135.

[2] Hoff, Daniel J., and Olver, P. J., "Extensions of invariant signatures for object recognition", J. of Math. Imaging and Vision 45.2 (2013): 176-185.

[3] Boutin, M., "Numerically invariant signature curves", Int. J. of Computer Vision 40.3 (2000): 235-248.

[4] Musso, E. and Nicolodi, L., "Invariant Signatures of Closed Planar Curves", J. of Math. Imaging and Vision, 35 (2012): 68-85.

[5] Golub, G. H., and Van Loan, C. F., *Matrix computations*, 2012.

[6] Richards, W., Benjamin D., and Whittington, D. "Encoding contour shape by curvature extrema." JOSA A 3.9 (1986): 1483-1491.

[7] Amenta, N. and Bern, M., Digital scan of an ostrich egg, unpublished raw data.

[8] Hoff, Daniel J., and Olver, P. J., "Automatic Solution of Jigsaw Puzzles", J. of Math. Imaging and Vision, 49 (2014): 234-250.