THE AVERAGE OF A SET OF COMPATIBLE CURVES AND SURFACES

A Thesis
Presented to
The Academic Faculty

By

Mukul Sati

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing

Georgia Institute of Technology

August 2019

THE AVERAGE OF A SET OF COMPATIBLE CURVES AND SURFACES

Approved by:

Dr. Jarek Rossignac, Advisor
School of Interactive Computing
Georgia Institute of Technology

Dr. Greg Turk
School of Interactive Computing
Georgia Institute of Technology

Dr. Concettina Guerra
School of Interactive Computing
Georgia Institute of Technology

Dr. Thomas Kurfess
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Karen Liu
School of Interactive Computing
Georgia Institute of Technology

Date Approved: April 18, 2019

ACKNOWLEDGEMENTS

Words cannot convey my appreciation for my advisor Dr. Jarek Rossignac, but I will try. Dr. Rossignac has been a mentor and guide for both research method and research philosophy. He allowed me the freedom and luxury of working at my pace and owning my learning. Very importantly, he made me comfortable asking questions. Working with him exposed me to a variety of research problems and, while I grappled with each of them to varying levels of success, Dr. Rossignac always motivated me to strive to understand the essence of the problem. Thank you, dear sir, for making me fall in love (again) with geometry and math – a fascination that I am sure will last a life-time.

I would like to thank Dr. Thomas Kurfess for introducing me to, and encouraging me to get practical experience with manufacturing. Thanks also, to Dr. Roby Lynn for being ever-willing to answer my incessant, basic questions about machining, mechatronics and control. Also, I would like to thank Dr. Tommy Tucker and Dr. Chris Saldana for insights and assistance on the CNC control work that funded a significant portion of my graduate studies. My interactions with people in the M.E. department at Georgia Tech have significantly influenced my research interests.

I would like to thank my committee members, Dr. Greg Turk, Dr. Concettina Guerra, and Dr. Karen Liu for their valuable inputs and for their service. Along with my advisor, Dr. Turk and Dr. Liu were ever present during my doctoral education, were exemplary teachers, and model researchers. I was continually inspired by both their technical skills, and their calm and poised demeanor.

Thanks are also due to members of the graphics group at Georgia Tech for stimulating discussions. I would also like to thank my friends Sandeep and Aishwarya for non-remote counsel. Finally, I would like to thank all of my family for their blessings, support and guidance. Especially, Meghna for lovingly putting up with me, my parents for their unconditional love, and, my brother Saurabh for always being a willing listening ear.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The proposed thesis focuses on providing definitions, closed-form expressions and algorithmic solutions for computing averages of sets of shapes and associated standard deviation fields. By shapes, we mean lines, planes, curves, and surfaces. Note that these shapes are embedded in a common ambient space – by which we mean that their scales, orientations, and positions are fixed and important.

We establish sufficient conditions on sets of shapes for which our proposed solutions yield topologically valid results, and call such sets compatible. We propose, study and apply suitable generalizations of the statistical formulations of the average and standard deviation of sets of numbers to sets of compatible shapes. In particular, the solutions proposed extend naturally the popular concept of the medial axis to two and more curves in the plane or in three-dimensions and to two or more surfaces.

We demonstrate the usefulness of our developed algorithms for tasks such as: morphing between shapes, sketch specification by overdrawing, and, analyzing the path of the cutting tool of a CNC machine.

CHAPTER 1

INTRODUCTION

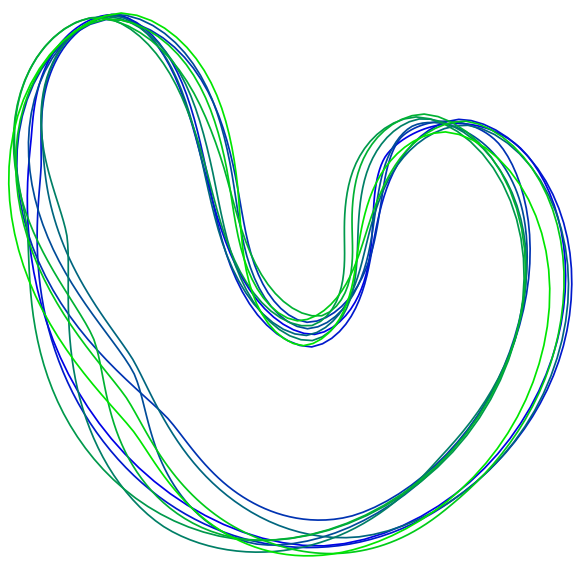

Figure 1.1: What is the average of a set of curves?

1.1 Thesis overview

What is the average of the set of closed planar curves of Fig. 1.1? The answer to this question is not obvious.

It is relatively easier to enumerate properties that we would expect the average of a set of curves to possess. For example, we would expect that the average depends on all the curves instead of only on a particular subset, and that the average passes through all the points that all the curves have in common.

But perhaps, it is wisest to first consider the questions: "why do we need the average of curves?" or, "of what use is the average of a set of curves?": Techniques for computing an average (or a measure of central tendency) are important because if we wish to represent a set of data by a single entity, then, the average is the ideal *representative*. In

algorithms that operate on large data, an average typically captures the global properties of the input elements and can serve as a summary description of the data (for purposes such as visualization, analysis, design) and also as a proxy for the entire data (for reducing the computational complexity of subsequent operations).

In this thesis we present algorithms that take as input a set of planar curves and yield a single representative curve. Our presented algorithms yield a curve that possesses properties expected of an average (Fig. 1.2), and thus, we call it the "average curve". We compute the average curve by snapping any one of the input curves.

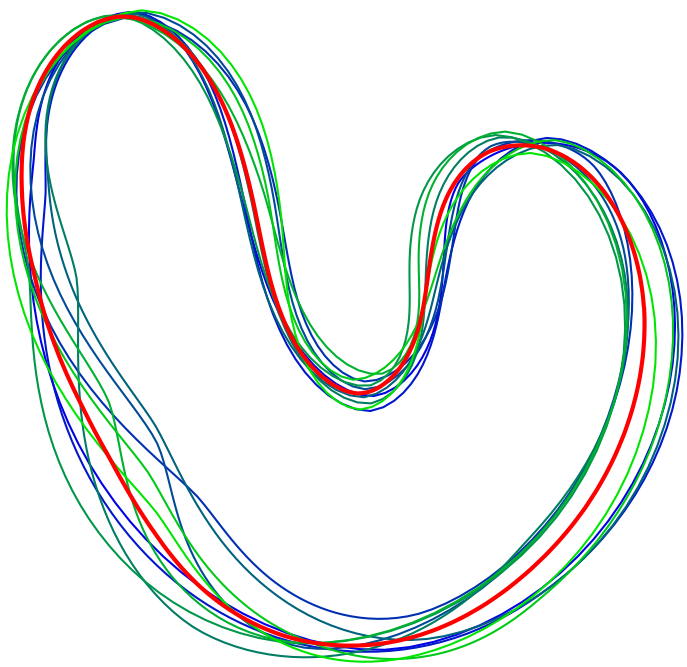

Figure 1.2: A set of input curves in the plane and the average curve (red).

Our developed 'Snap' algorithm, for a sample point, computes a query point on each input curve, computes the tangent lines to the curves at the query points, and projects the sample onto the average of the tangent lines.

We extend this approach to surfaces in 3D. To do so, we describe how to average planes, and then, use a plane average in a Snap algorithm that computes the average surface for a set of input surfaces(Fig. 1.3).

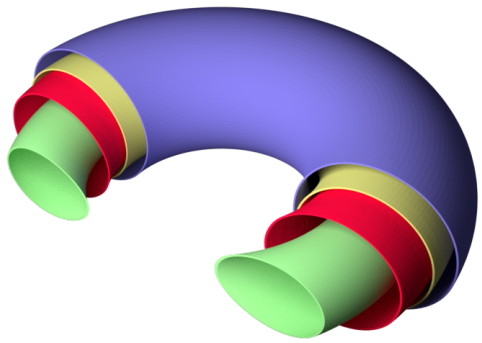

Figure 1.3: A set of input surfaces and the average surface (red). To better reveal the internal structure of the surfaces, we have clipped each surface with a clipping plane.

We further extend these two solutions to the problem of averaging curves in 3D. We describe how to average lines in 3D, and, use this result to compute the average space curve (curve in 3D) for sets of space curves (Fig. 1.4).

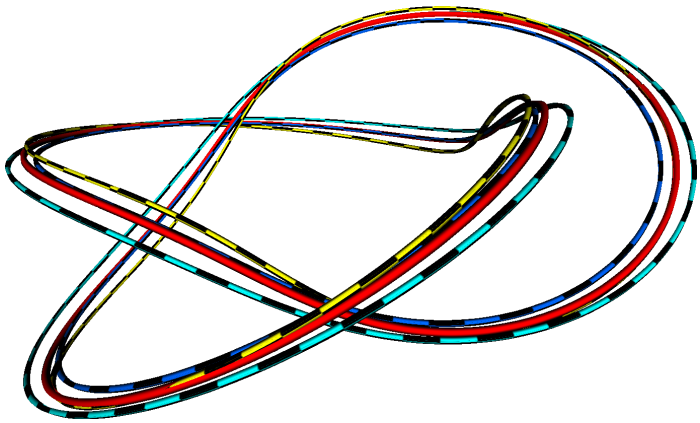

Figure 1.4: A set of input curves in 3D and their average curve (red).

It may be desirable to quantify the variability of the input set about the average. Indeed, in computationally driven natural science inquiries, efficient algorithms to compute both an average and a measure of variability about the average, of data collected during an experiment are key enablers of subsequent inductive reasoning. The algorithms we present in this thesis for computing the average shape (curve or surface) also compute a scalar field

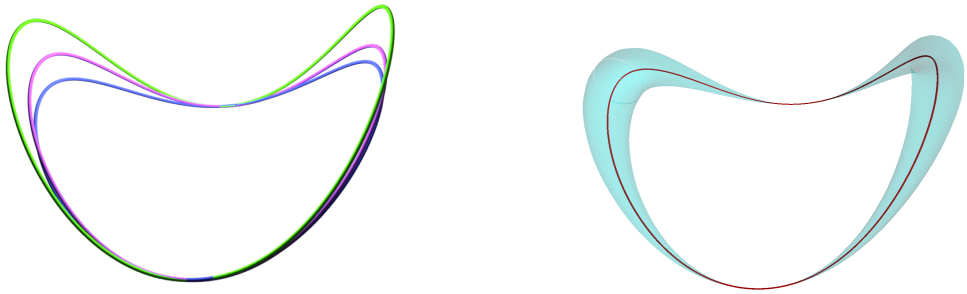

Figure 1.5: For a set of input curves (left), we compute not just the average (right, red), but, also a variability field over the average. The variability field is visualized here using a translucent, cyan-colored offset tube of variable radius. The radius of the tube at a particular point on the average is proportional to the variability value at that point.

that captures the variability of the set of shapes. The variability information is useful for visualization (Fig. 1.5).

We note here that we use the term 'shape' to mean objects (curves, surfaces) that are situated, or, embedded in the plane or in space. We bring this precision to the readers attention, because often, when it is noted that two shapes are roughly similar, the writer implies that 'intrinsic' properties of the shapes are similar regardless of their embeddings in space. A technique to average two circles of the same radii which discards how they are situated in the plane will yield a circle of the same radius as the result. However, the solutions we propose consider how these two circles are embedded in the plane to compute the average. Thus, our computed result is sensitive to changes in the relative positions of the circles (Fig. 1.6).

Thus, the approaches proposed here allow one to compute the average and variance of a set of situated curves and surfaces (Fig. 1.5).

Additionally, our algorithms compute, for each point on the average a corresponding point on each input shape (Fig. 1.7). By considering the set of points (one on each shape) that correspond to the sample point on the average as being in correspondence with each-other, we define the n-way correspondences between the input set of shapes. To compute

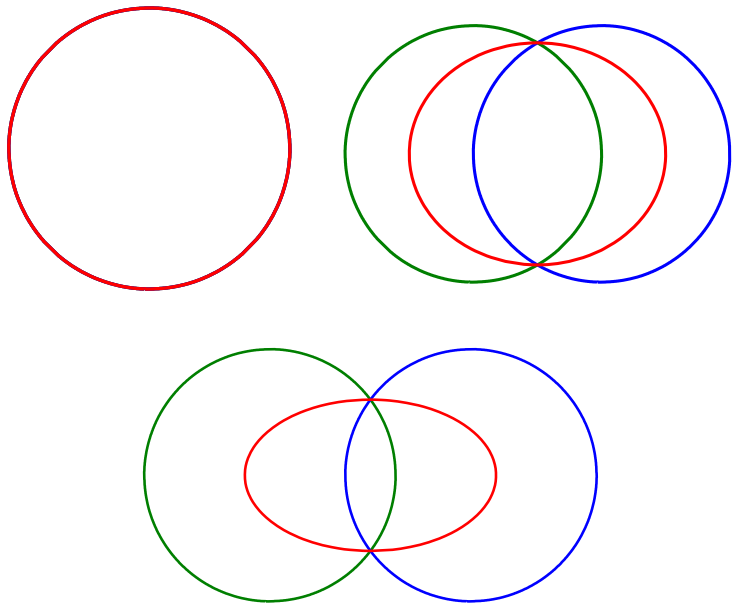

Figure 1.6: Top Left: Two identical input circles and their representative (red, coincident over the input circles). Top Right, Bottom: Two non-concentric, equal radius circles (blue, green) embedded at different locations in the plane and their representative (red).

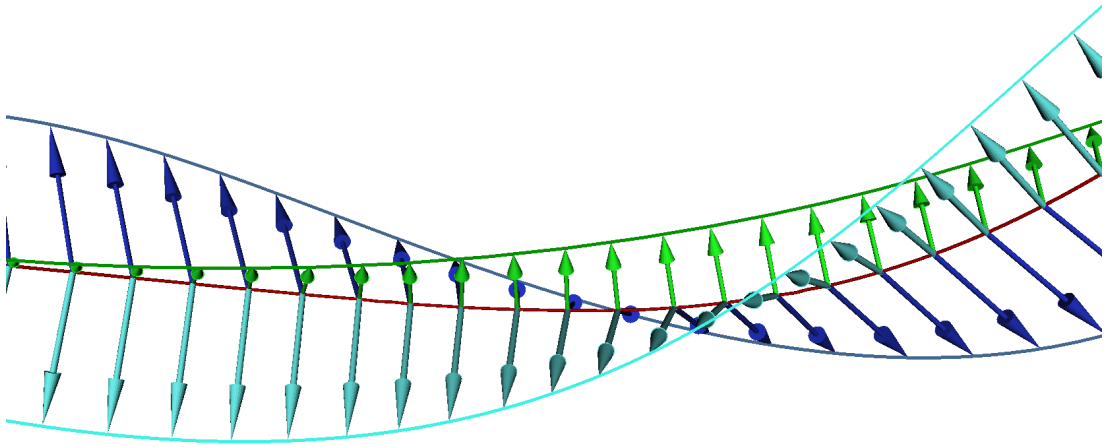

Figure 1.7: Aside from computing the average (red), we also compute for each point on the average, a corresponding point one each input shape (blue, green, cyan curves). The correspondences are visualized here as green, blue, cyan arrows that point from a point on the average to its corresponding point on each input.

the set of correspondences for a given point on one of the input shapes, we present a modification of the Snap algorithm.

This n-way correspondence has several potential applications. As an example, attribute data (such as texture) specified on one shape S_1 can be transferred to another shape S_2 by associating to each point of S_2 the attribute value associated with its corresponding point on S_1. Attributes defined on S_1 and S_2 can also be averaged in a similar manner.

Other applications of the ideas presented in this thesis include morphing and weighted averaging, sketch consolidation, data aggregation, and geometric modeling.

The *theoretical contribution* of our thesis is that we mathematically define the set of points that constitute the average. For example, for the case of input curves in a plane, we consider a scalar field over the plane and define the average as the points where a particular expression of the scalar field holds. What is an appropriate scalar field to consider and what is the expression for point-set containment? For a set of planar curves, these questions are answered in Chapter 5. Our definitions for the case of surfaces are presented in Chapter 7 and, for the case of space curves are described in Chapter 8.

Given a list of shapes (S_i), our definitions yield representatives that are *symmetric* – the average does not change if the list is reordered. Thus, our average is defined by the unordered input set $\{S_i\}$.

Additionally, our definitions have the property that, for a set of lines (resp. planes) they yield, as the average, a line (resp. plane). The average line or plane has a several appealing properties that an average should possess (Section 3.7).

The *computational contribution* of our thesis is that we present algorithms for computing approximations to the average that are simple and efficient. These algorithms leverage our closed form expressions for averaging planes and lines. Additionally, the algorithms also simultaneously compute correspondences and variability.

The algorithms we present for computing average shapes operate by starting with a seed shape (generally, but not necessarily, one of the inputs), sampling 'query' points on

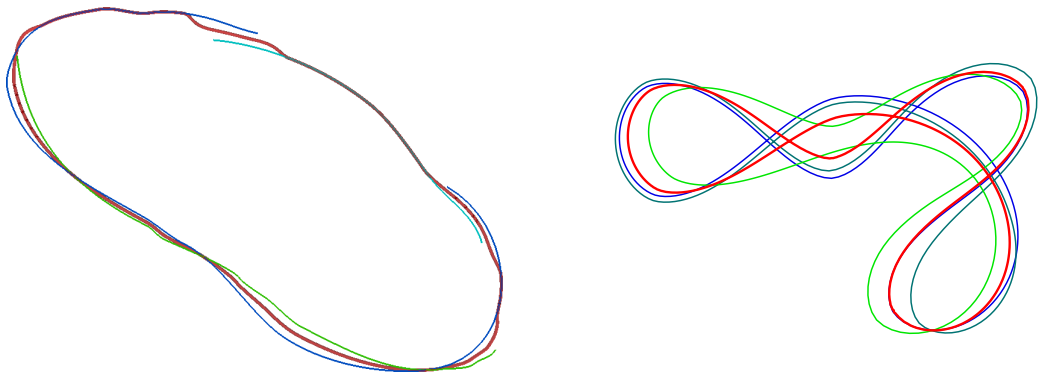

Figure 1.8: We present algorithmic modifications that allow computing representative curves for some configurations of inputs that contain curves with boundary (left) and that contain self-intersecting curves (right).

the shape, and, 'snapping' each point so that it lies on the average shape. The connectivity information for the computed samples on the average shape is taken to be the same as that of the query points on the seed shape.

A limitation of our approach is that it yields results that are useful for roughly parallel (compatible) sets of shapes. The class of compatible shapes exhibits sufficient variety to merit study. We also present algorithmic modifications that extend the applicability of our algorithms to some configurations of incompatible sets of shapes (see for example Fig 1.8). These include: a) shapes with boundary and b) self-intersecting shapes.

1.2 Thesis structure

The thesis is structured as follows: we envision the modern shoemaker's shop in Chapter 2 providing a motivating example for our work. We also discuss formulations for the average of points and scalar measures and highlight the symmetric nature of these formulations.

In Chapter 3, we discuss the simplest shape-averaging problem: averaging lines and planes. We discuss our formulations of the average of a set of lines (in 2D and 3D) and planes (in 3D) and show how these formulations yield average lines and planes that are

easily computed. We also derive expressions for the weighted average of a set of lines. Similar to the weighted average concept of a set of scalars, the concept of the weighted average for a set of lines yields an average that is more sensitive to changes in a line that is assigned a higher weight. The weighted average concept allows us to develop algorithms for morphing between shapes. The results presented in this chapter are both of independent value, and are used in our computational algorithms for averaging shapes.

In Chapter 4, we present a summary of works whose contributions we leverage, and also discuss prior works on averaging shapes.

The remainder of the thesis discusses our proposed algorithms for computing average curves and surfaces. In Chapter 5, we describe our approach to computing an average curve for a set of closed loop planar curves. In Chapter 7, we present our method of computing an average surface for a set of boundary-less, manifold surfaces.

Subsets of the contents of Chapter 5 and Chapter 7 have been published as the following joint papers respectively:

- Sati, Mukul, Jarek Rossignac, Raimund Seidel, Brian Wyvill, and Suraj Musuvathy. "Average curve of n smooth planar curves." Computer-Aided Design 70 (2016): 46-55.
- Sati, Mukul, and Jarek Rossignac. "Average and variance of a quasi-parallel family of surfaces." Computer-Aided Design 102 (2018): 61-71.

The common computational idea employed in our presented algorithms is that of 'snapping' a candidate point onto the average curve (or surface) by iteratively 'moving' it to lie on average lines (or planes) computed using local information. In Chapter 6 we discuss the performance and accuracy of our proposed 'snapping' approach.

In Chapter 8, we present an algorithm for averaging closed loop space curves. Curves in 2D and surfaces in 3D have a co-dimension of 1. That is, the dimension of the input shapes (1 for curves, 2 for surfaces) is one less than the dimension of the embedding space (2 for the Euclidean plane or 3 for Euclidean space). However, curves in 3D have a co-

dimension of 2. This difference requires us to modify both our mathematical formulation and the computational algorithm.

We demonstrate two practical applications of our curve-averaging: sketch specification by overdrawing and manufacturing process analysis.

To yield useful results for the sketching application, we needed to modify the algorithms developed for closed curves. We discuss in Chapter 9 these computational modifications for open curves.

In Chapter 10, we describe a CNC machining system that we have developed. We use the system to collect manufactured shape data, and, describe novel ways in which the collected data may be analyzed using techniques presented in our thesis. Specifically, we demonstrate how knowledge of both the average and the variability field facilitates assessing the quality of a manufacturing process and for informing its improvement.

An early version of the developed machining system is described in the joint publication:

- Lynn, Roby and Sati, Mukul and Tucker, Tommy and Rossignac, Jarek and Saldana, Christopher and Kurfess, Thomas. "Realization of the 5-Axis Machine Tool Digital Twin Using Direct Servo Control from CAM" National Institute of Standards and Technology (NIST) Model-Based Enterprise Summit, 2018.

In summary, our thesis focuses on providing definitions, closed-form expressions and efficient algorithmic solutions for computing representative shapes (or, average shapes) of compatible configurations of sets of situated shapes. We also compute all-pair correspondences and use them to represent and compute the local variability of the input set of shapes as a scalar field over the average shape.

CHAPTER 2

BACKGROUND

2.1 A motivating example: The shoe fitting problem

To clarify and motivate our goal, we first discuss a simplified version of a practical problem: How to improve the online search for shoes that fit a particular consumer?

Let us assume that many consumers scan their feet [1] every few years and that 3D surface models of these anonymized scans are made available online to all shoe manufacturers. Let us also assume that custom shoes are too expensive for the mass market and that a manufacturer decides to mass-produce only m differently shaped shoes for a particular shoe style. How should we choose/design these m shapes?

For each shoe-size, the Mondopoint Standard [2], which is used for military shoes and for ski boots, defines the mean-foot length and width for which a shoe is suitable. Instead of the mean, European standard EN 13402 defines intervals of foot length and width for which the shoe is suitable. Except for these two measurements (length and width), these standards do not consider other aspects of the foot shape. Some authors propose to incorporate a larger number of *discrete* measurements [3].

To take full advantage of the increasing availability of affordable metrology tools, instead of using discrete measurements to establish shoe sizes, we propose to use geometric (curve or surface) models of the shape of the foot. These models may be recovered from the scans. Hence, instead of *shoe sizes*, we will talk about standard *shoe shapes*. We assume that the scanned models are converted, using morphological filters [4], to a smooth outer envelope.

Consider the input set $\{S_i\}$ of shapes, with each shape S_i representing the envelope surface of a customers foot. We want to divide set $\{S_i\}$ into m clusters of shapes. m may,

for example, be dictated by economic constraints on how many different shoe shapes of a given shoe style the manufacturer is willing to mass-produce. For each cluster, we want to compute the cluster average shape (CAS) and, for each CAS, A_k, we want to design a shoe that best fits all customers whose foot is closer to A_k than to any other CAS. We argue that the optimal solution to this clustering problem selects the m CASes so as to maximize the expected fit, i.e., to minimize a measure, E, which we define as the average of a chosen measure of disparity between each S_i and its closest A_k.

To properly formulate this optimization task and the measure against which optimization algorithms can be validated, we need to have precise mathematical definitions: (1) of a suitable disparity measure between two shapes and (2) of the shape average of a set of shapes (i.e., a surface that is the average of a given set of surfaces or a curve that is the average of a given set of curves).

Thus, the availability of such shape statistics techniques could transform the online shoe retail industry and shoe design technology and deliver ergonomic benefits to everyone. Indeed, the theoretical definitions of shape averages and of disparity measures, and the efficient algorithms that we have developed have broad applicability across a spectrum of industrial and research practices in other domains, as highlighted in the remainder of this thesis (see for e.g. Ch. 9 and Ch. 10).

2.2 Preliminaries: Averaging numbers and points

A key concept expounded in this thesis is that of symmetric (i.e., fair) formulations and constructions for averaging n entities. A property, expression or construction that depends on a list of inputs but that is independent of the ordering of the elements is called *symmetric*. In this section, we discusses symmetry in the context of averaging numbers and points.

The average of a set of numbers

We start with what we do know – the average of two real numbers $\{x_1, x_2\}$ has a closed form expression: $x = \frac{x_1+x_2}{2}$. The value of this is not altered under the pair of simultaneous substitutions $x_1 \to x_2, x_2 \to x_1$. For n numbers $\{x_1, \ldots, x_n\}$, the value of the expression denoting the average $x = \frac{x_1+\ldots+x_n}{n}$ does not change with any permutation of the number set. We say that the average is invariant under permutations of the ordered list of numbers $(x_1, \ldots, x_n)$. It is due to this invariance to permutations that we say that the average is a property of the set of inputs.

The symmetric expression for the average of two numbers can be obtained by at least the following characterizations, which assume that the number line is equipped with the standard, oriented coordinate frame to measure (signed) distances. The average is the number x such that:

(a) *Zero set of signed distance*: The sum of the signed distances from it to the two numbers is 0: $|x_2 - x| - |x_1 - x| = 0$.

(b) *Number at equal distance*: It is at equal (unsigned) distance from the two numbers: $|x - x_1| = |x - x_2|$.

(c) *Summed squared distance minimizer*: The summed square distance from it to the two numbers is minimized: $x = argmin_{x\in\mathbb{R}} \sum_{i=1}^{2}(x - x_i)^2$. Note that the variance is given by a scaling of the summed square distance and thus, is also minimized at x.

We see however that the number at equal distance characterization does not extend to n real numbers. Consider the case of three distinct numbers. Given a distance d, for a given x_1, the number at distance d is one of $x_1 + d$ or $x_1 - d$. If we require that the number x be at distance d from x_1, x_2 and x_3, with $x_1 < x_2 < x_3$, then the condition of being at distance d from x_1 and x_2 yields $x = x_1 + d = x_2 - d < x_3 - d$. Thus, x cannot be at distance d simultaneously from 3 numbers in general. Now, we shall look at the case of the centroid

of a set of points in the plane, where, we notice that the zero-set definition does not remain valid.

The centroid of a set of points

We also do know how to compute the average of two points in the Euclidean plane, and, this is given by the simple closed form expression corresponding to the centroid. However, the method (a) that was described above for two numbers cannot be used to arrive at the expression for the centroid, as, the concept of a "signed distance", which was well-defined for numbers on the number line, is not defined anymore. The signed distance concept is well-defined for shapes that partition the space in which they are embedded into an exterior and interior. A necessary condition for such a partitioning is the shapes have co-dimension 1. However, points have a co-dimension of 2 with respect to the plane.

Also, the solution set characterized by (b) need not be a point. In the Euclidean plane, we see that for 2 points we will obtain the line that is the perpendicular bisector of the edge joining the two points, for 3 points, the solution set is the circumcenter point, and, for 4 points that are not the vertices of a cyclic quadrilateral, the solution set is empty.

Consider now an alternative formulation of the average: the point which minimizes the summed distances to the input points. For 3 points, the solution is the Fermat point rather than the centroid. For n points, the solution is termed the geometric median. Unlike the centroid which is computable using the simple coordinate-wise averaging formula, the geometric median does not possess a closed form. Rather, computation of an approximation of the geometric median minimizer generally requires iterative methods such as those described in [5].

It is only the characterization of the average given by (c) that yields the expression for the centroid of the set of points.

When we seek to apply the characterization that yields the centroid for n points to the problem of averaging n curves in the plane: the representative is the set of points where

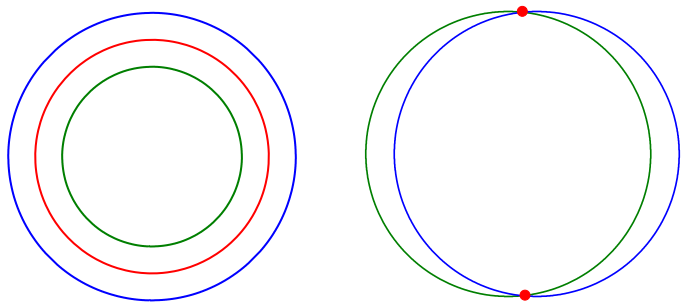

Figure 2.1: Left: Two concentric circles (blue, green) and the set of points where the summed squared distance attains a minimum value (red). Right: Two non-concentric circles (blue, green) – the minimum value of the summed squared distance is attained at points (red dots) where the circles intersect.

the summed squared distances to the input curves is minimized. This characterization does indeed yield useful results for concentric circles. However, in general, the solution is a (set of) isolated point(s) (Fig. 2.1).

The above discussions for averaging numbers and points, and the initial failed attempt to average curves serve to highlight that:

- For an entity to serve as an average of, or as a representative for, a set of entities, it should be symmetric.

- Determining the characterization that yields a useful representative for a particular class of input sets is not trivial. In particular, characterizations that yield useful results for a particular input class (for e.g., a set of numbers) need not be useful for another input class (for e.g., a set of points on the Euclidean plane).

CHAPTER 3

AVERAGING VECTORS, LINES AND PLANES

In this chapter, we present techniques for averaging vectors, lines and planes. First, we present known approaches to averaging a set of unit vectors in 2D and in 3D. Then, we present novel formulations for averaging sets of lines (in 2D and 3D) and planes (in 3D).

The results presented in this chapter are useful in and of themselves. Additionally, they are also used in subsequent chapters which describe our approach to computing average curves and surfaces.

3.1 Averaging a set of unit vectors in 2D

Given a set of unit vectors $\{n_i\}$, the following are well known approaches to compute an average vector n:

Normalized linear average: As a set of vectors in a vector space can be added and scaled, the algebraic expression for the average of a set of numbers, $\frac{\sum_{i=1}^{k} n_i}{k}$, is well-defined for a set of vectors as well. However, the expression may not yield a unit vector without normalization. Explicitly renormalizing, we may compute the average vector as:

$$n = \frac{\sum_i n_i}{\|\sum_i n_i\|}$$

Squared dot average: Considering the metric structure of the Euclidean plane and Euclidean space, we can compute n as the unit vector that maximizes $\sum_i (n_i \cdot n)^2$, where $\cdot$ denotes the Euclidean dot product. That is, n is the unit vector which maximizes the summed squared cosine of the angle between it and the inputs. Equivalently, n is the unit vector which minimizes the summed squared sines of the angles between it and the inputs.

In an orthogonal frame in 2D, each n_i is represented using its coordinates (x_i, y_i). Using coordinates (x, y) for n, we obtain:

$$\begin{aligned} n &= \underset{(x,y)\in\mathbb{R}^2}{\operatorname{argmax}} \sum_i (xx_i + yy_i)^2 \\ &= \underset{(x,y)\in\mathbb{R}^2}{\operatorname{argmax}} \sum_i x_i^2 x^2 + y_i^2 y^2 + 2x_i y_i xy \end{aligned}$$

Assuming $n = (\cos\alpha, \sin\alpha)$, α can be computed using the condition:

$$\tan(2\alpha) = \frac{2\sum x_i y_i}{\sum x_i^2 - y_i^2} \tag{3.1}$$

Spherical average: n is the unit vector that minimizes the summed squared spherical distances to the input vectors. In 2D, this means that n is the vector that minimizes the summed squared angles to the vectors.

3.2 Averaging a set of unit vectors in 3D

Normalized linear average: This is the same as the expression for 2D:

$$n = \frac{\sum_i n_i}{\|\sum_i n_i\|}$$

Squared dot average: Using the matrix notation, we arrive at a unified expression for n as:

$$n = \underset{x\in\mathbb{R}^2}{\operatorname{argmax}}\, x^T \Big(\sum_i [n_i \otimes n_i] \Big) x$$

Here $\otimes$ denotes the outer product. In case $\Big(\sum_i [n_i \otimes n_i] \Big)$ is non-negative definite with a unique largest singular value n is then directed along the vector corresponding to the largest singular value and is such that it has a positive dot product with each input vector.

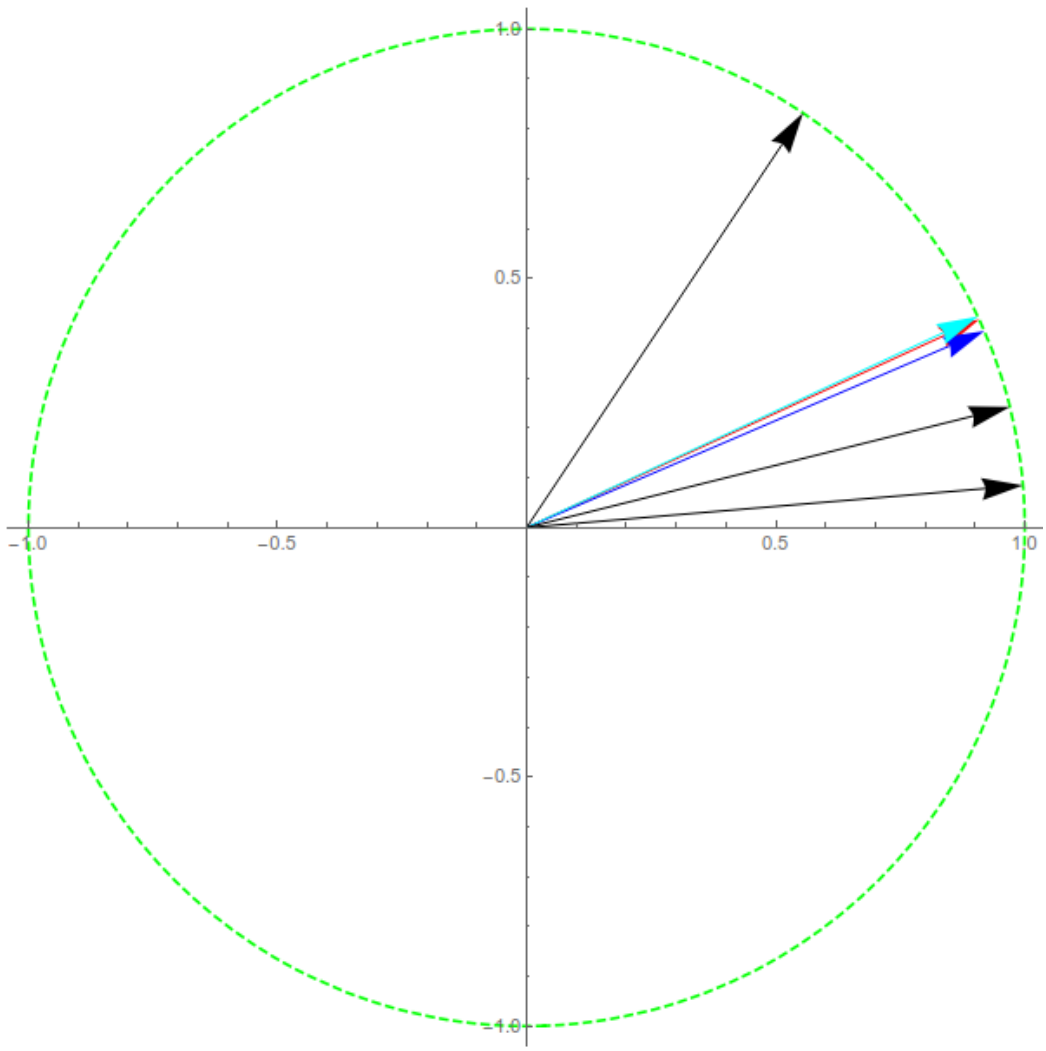

Figure 3.1: For a set of three unit vectors (black), the normalized linear average (red), the squared dot average (blue) and the spherical average (cyan) are shown.

Spherical average: The definition for n is the same as the case for 2D: n is the unit vector that minimizes the summed squared spherical distances to the input vectors. We refer the reader to [6] for an iterative algorithm.

3.3 Averaging a set of lines in 2D

We present two formulations for the average of a set of lines in 2D. These formulations yield different results. In general configurations, both formulations yield a line. For each

formulation of the average, we also provide a property for validating whether the set of input lines are compatible.

One of our formulations identifies the average as the set of points that are the zero-set of a scalar field defined over the plane. Thus, we term this average line the zero-set average line (zAL). The second formulation describes the average as the set of points that are the valley-set of a scalar field over the plane. Thus, this average is termed the valley-set average line (vAL).

An oriented line l is represented by a point p on l and a unit direction vector t (the unit tangent). That is, $l = (p, t)$. Equivalently, in 2D, $l = (p, n)$ where n is a 90 degree counter-clockwise rotation of t (i.e., n is the unit normal).

For a set of oriented lines to be compatible, we require them to be closest projection compatible: for every input line l_i, the closest projection function Π_{ij} that maps a point on l_i to its closest point on the line l_j should be an orientation preserving bijection for every other input l_j. That is, if u and v are ordered (in accordance with the orientation of l_i) points on l_i, then $\Pi_{ij}(u)$ and $\Pi_{ij}(v)$ are ordered in accordance with the orientation of l_j.

3.3.1 Zero-set average line (zAL)

For a given set of oriented lines $\{l_i\} = \{(p_i, n_i)\}$, the zAL is the zero level-set of the summed signed distance field Z to $\{l_i\}$. That is, given a point p, the signed distance from p to each l_i is computed and summed to obtain the value $Z(p)$. p lies on the average line if $Z(p) = 0$.

For the given set of lines,

$$Z(p) = \sum_i (p_i p \cdot n_i)$$

$$\therefore Z(p) = 0 \Rightarrow \sum_i (p_i p \cdot n_i) = 0$$

To see that this expression identifies a set of points on a line, note that this is a linear expression in p. More precisely, replacing $p_i p$ by $op + p_i o$, where o denotes the origin of the chosen coordinate system, we have:

$$Z(p) = 0 \Rightarrow op \cdot (\sum_i n_i) = -\sum_i (p_i o \cdot n_i) \tag{3.2}$$

Thus, for a given set of lines, a point p is on the zero-set average if the dot product of p's position vector op with the vector $(\sum_i n_i)$ is a constant.

Note that thus, the set of such p's lie on the line with normal directed along the normalized linear average $n = \sum_i n_i$ (Sec. 3.1).

3.3.2 Valley-set average line (vAL)

For a set of oriented lines $\{l_i\} = \{(p_i, n_i)\}$, the summed squared distance function is $Q(p) = \sum_i (p_i p \cdot n_i)^2$. Using Cartesian coordinates (x, y) for p, (x_i, y_i) for p_i, and (u_i, v_i) for n_i, and writing $Q(x, y)$ for $Q(p)$, we obtain $p_i p \cdot n_i = (x - x_i)u_i + (y - y_i)v_i$ and thus

$$\begin{aligned} Q(x, y) = \sum_i (&u_i^2 x^2 + v_i^2 y^2 + 2u_i v_i xy - 2u_i(u_i x_i + v_i y_i)x - \\ &2v_i(u_i x_i + v_i y_i)y + 2u_i v_i x_i y_i + u_i^2 x_i^2 + v_i^2 y_i^2) \end{aligned} \tag{3.3}$$

$$= c_{2,0}x^2 + c_{0,2}y^2 + c_{1,1}xy + c_{1,0}x + c_{0,1}y + c_{0,0} \tag{3.4}$$

Q is a quadratic polynomial. Hence, we can use the following matrix notation with x now denoting the vector $[x\ y]^T$:

$$Q(x) = x^T A x + b^T x + c \text{ where}$$

$$A = \begin{bmatrix} c_{2,0} & \frac{c_{1,1}}{2} \\ \frac{c_{1,1}}{2} & c_{0,2} \end{bmatrix}, b = \begin{bmatrix} c_{1,0} \\ c_{0,1} \end{bmatrix}, c = c_{0,0}.$$

Note that, the coefficients for the second degree terms in x, y only involve the coordinates of the normal vectors $\{n_i\}$ and are independent of position coordinates $\{p_i\}$. Similar to the case for squared dot average of vectors, A can be written as $A = \sum_i n_i \otimes n_i$.

In the selected cartesian coordinate system, we have the following expressions for the gradient ∇Q and Hessian H of Q:

$$\nabla Q(x, y) = \begin{bmatrix} 2c_{2,0}x + c_{1,1}y + c_{1,0} \\ c_{1,1}x + 2c_{0,2}y + c_{0,1} \end{bmatrix}$$

$$H(x, y) = \begin{bmatrix} 2c_{2,0} & c_{1,1} \\ c_{1,1} & 2c_{0,2} \end{bmatrix} = 2A$$

Condition for compatibility: A set of oriented lines in 2D are compatible wrt. the valley-set average when both eigenvalues of H are non-negative and there exists a unique largest eigenvalue of H. This condition is in addition to closest-projection map compatibility mentioned earlier.

The vAL of a set of lines is a height-valley of their summed squared distance height field Q. We use the definition of [7] for the valley of a height-field: it is the set of points where a minima of the height-field is attained along a principal direction of the height-field. For a set of compatible lines, we seek the height valley defined by $\nabla Q \cdot e_2 = 0$, where e_2 is the eigenvector of the Hessian H of Q that corresponds to the larger eigenvalue of H. For compatible lines, the larger eigenvalue is unique, and, thus, the vAL is well-defined.

Because H is a constant over the plane, so are its eigenvectors e_i. We have a closed form expression for the eigenvectors. For deriving it, consider the action of H as a linear transformation. H maps lines through the origin to lines through the origin, or to the origin point. An eigenvector e_i of H support the lines through the origin that H fixes. The eigenvector e_i itself is just scaled by H and thus, for every such vector e_i, $\exists\lambda$ such that $He_i - \lambda e_i = 0$.

Now, consider the linear transformation $H - \lambda I$ where I is the identity matrix. This transformation acts on the vector e_i as $He_i - \lambda I e_i = He_i - \lambda e_i$, which, we know evaluates to the zero vector. Thus, $H - \lambda I$ maps e_i to the zero vector, and, it holds that, either $e_i = 0$, or, that, the transformation $H - \lambda I$ maps some non-zero vector to 0. Assuming the latter case, this means that $H - \lambda I$ 'compresses space' by at-least a dimension, mapping the plane to at-least a line, or, more generally to a set that has zero measure / volume as a subset of $\mathbb{R}^2$. This geometric fact is expressed as the vanishing of the determinant of $H - \lambda I$.

The condition $det(H - \lambda I) = 0$ gives a quadratic characteristic equation in λ. The larger eigenvalue λ_2 is given by the larger root.

$$\begin{aligned} \det(H - \lambda I) &= 0 \\ \Rightarrow \lambda^2 - 2(c_{0,2} + c_{2,0})\lambda + 4c_{0,2}c_{2,0} - c_{1,1}^2 &= 0 \\ \Rightarrow \lambda_2 = c_{0,2} + c_{2,0} + \sqrt{(c_{0,2} - c_{2,0})^2 + c_{1,1}^2} \end{aligned}$$

Now, e_2 is a vector for which it holds that $He_2 = \lambda_2 e_2$ or $(H - \lambda_2 I)e_2 = 0$. The matrix $H - \lambda_2 I$ has zero determinant and thus, is not full-rank. Thus, $H - \lambda_2 I$ has linearly dependent rows, and, can be written as:

$$H - \lambda_2 I = \begin{bmatrix} v_1 \\ \beta v_1 \end{bmatrix}$$

Our requirement that $(H - \lambda_2 I)e_2 = 0$ will be true if $v_1 \cdot e_2 = 0$. That is, if e_2 is a vector orthogonal to v_1. Substituting λ_2 back into the equation, and using x, y to denote the coordinates of e_2, the following relation between x and y is required for orthogonality :

$$y = \frac{\left(\sqrt{(c_{0,2} - c_{2,0})^2 + c_{1,1}^2} + c_{0,2} - c_{2,0}\right)x}{c_{1,1}} = m_1 x$$

Substituting $x = 1$ in the above equation, we obtain an eigenvector with eigenvalue λ_2.

The vAL condition is:

$$e_2 \cdot \nabla Q = 0 \tag{3.5}$$

$$\Rightarrow \begin{bmatrix} 1 & m_1 \end{bmatrix} \begin{bmatrix} 2c_{2,0}x + c_{1,1}y + c_{1,0} \\ c_{1,1}x + 2c_{0,2}y + c_{0,1} \end{bmatrix} = 0 \tag{3.6}$$

This expands to:

$$y = -\frac{c_{1,1}\left(c_{0,2} + c_{2,0} + \sqrt{{c_{1,1}}^2 + (c_{0,2} - c_{2,0})^2}\right)x}{{c_{1,1}}^2 + 2c_{0,2}\left(c_{0,2} - c_{2,0} + \sqrt{{c_{1,1}}^2 + (c_{0,2} - c_{2,0})^2}\right)} - \frac{c_{1,0}c_{1,1} + c_{0,1}\left(c_{0,2} - c_{2,0} + \sqrt{{c_{1,1}}^2 + (c_{0,2} - c_{2,0})^2}\right)}{{c_{1,1}}^2 + 2c_{0,2}\left(c_{0,2} - c_{2,0} + \sqrt{{c_{1,1}}^2 + (c_{0,2} - c_{2,0})^2}\right)}$$

$$\Rightarrow y = m_2 x + c$$

Thus, the vAL is a line. When we compute the product of the slope of this line with m_1, we see that $m_1 m_2 = -1$:

$$\begin{aligned} m_1 m_2 &= -\frac{\left(\sqrt{(c_{0,2} - c_{2,0})^2 + c_{1,1}^2} + c_{0,2} - c_{2,0}\right)}{c_{1,1}} \frac{c_{1,1}\left(c_{0,2} + c_{2,0} + \sqrt{{c_{1,1}}^2 + (c_{0,2} - c_{2,0})^2}\right)}{{c_{1,1}}^2 + 2c_{0,2}\left(c_{0,2} - c_{2,0} + \sqrt{{c_{1,1}}^2 + (c_{0,2} - c_{2,0})^2}\right)} \\ &= -1 \end{aligned}$$

Thus, the vAL is a line $l(p, t)$ that has for tangent a vector t orthogonal to e_2. The line passes through the point $p = (-\frac{2c_{0,2}c_{1,0} - c_{0,1}c_{1,1}}{4c_{0,2}c_{2,0} - c_{1,1}^2}, -\frac{c_{1,0}c_{1,1} - 2c_{0,1}c_{2,0}}{c_{1,1}^2 - 4c_{0,2}c_{2,0}})$, which minimizes Q, Thus, the vAL is a line and its location can be computed in closed form.

Note that the set of such p's lie on the line with normal directed along the squared dot average $n = \text{argmax}_{x \in \mathbb{R}^2} x^T \sum_i [n_i \otimes n_i]$.

Thus, for both the vAL and the zAL, the direction of the average line can be computed with knowledge of just the normals $\{n_i\}$ to the input lines $\{l_i\}$. Fig. 3.2 shows the vAL and the zAL of a set of four lines in two configurations. Note that the vAL and zAL are identical when all lines are parallel.

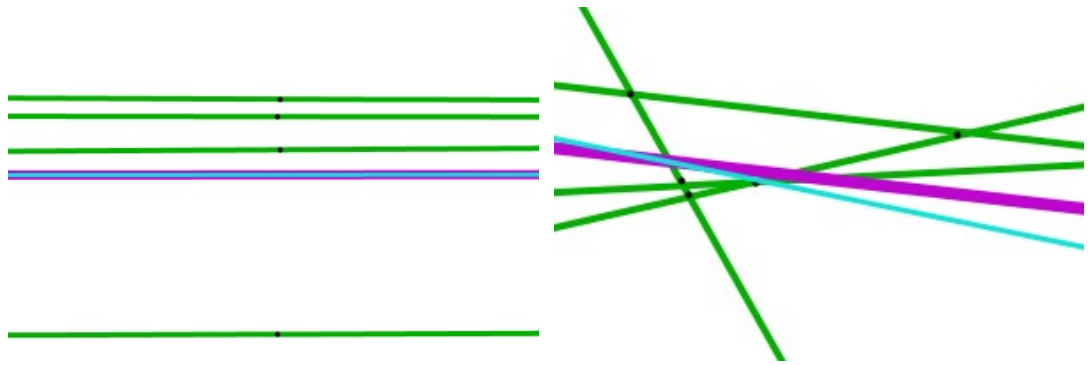

Figure 3.2: vAL (magenta) and zAL (cyan) for four parallel lines (left) and for non-parallel lines (right) oriented towards the right.

3.4 Averaging a set of planes in 3D

An oriented plane P is represented by a point p on P and a unit normal to the plane n. Similar to the case for lines in 2D, we present two formulations for the average of a set of planes in 3D. The formulations yield different results for set of input planes in general configuration. For compatible planes, both formulations yield a plane as the average.

As in the case for lines, for a set of oriented planes $\{P_i\}$ to be compatible, we require them to be closest projection compatible: for every input plane P_i, the closest projection function Π_{ij} that maps a point on P_i to its closest point on P_j should be a bijection for every other input plane P_j.

3.4.1 Zero-set average plane (zAP)

For a set of planes $\{P_i\}$, let $S_i : \mathbb{R}^3 \to \mathbb{R}$ given by $S_i(p) = (p_i - p) \cdot n_i$ denote the signed distance function to P_i. Consider the summed signed distances scalar field Z defined as $Z(p) = \sum_i S_i(p)$. The zAP is defined as the zero level set of Z.

$$Z(p) = \sum_i (p_i p \cdot n_i)$$

$$\therefore Z(p) = 0 \Rightarrow \sum_i (p_i p \cdot n_i) = 0$$

This is a single linear equation in three variables, and, thus is satisfied by points on a plane – the zero set average plane (zAP). Replacing $p_i p$ by $op + p_i o$, where o denotes the origin of the chosen coordinate system, we have:

$$Z(p) = 0 \Rightarrow op \cdot (\sum_i n_i) = -\sum_i (p_i o \cdot n_i)$$

Thus, the zAP has as normal the vector directed along $\sum_i n_i$.

3.4.2 Valley-set average plane (vAP)

Consider a set of planes, $\{P_i\}$. Let $S_i^2 : \mathbb{R}^3 \to \mathbb{R}$ defined as $S_i^2(p) = ((p_i - p) \cdot n_i)^2$ denote the squared distance function to P_i.

Consider the summed squared distances scalar field Q given by $Q(p) = \sum_i S_i^2(p)$. In vector-matrix notation, $Q(p) = op^T Aop + b^T op + c$, with op (and op_i appearing in expressions below) denoting the vector from the origin of the selected coordinate system to the point p (or p_i, accordingly), and, where:

$$A = \sum_i n_i \otimes n_i$$

$$b = \sum_i -2(op_i \cdot n_i) n_i$$

$$c = \sum_i (op_i \cdot n_i)^2$$

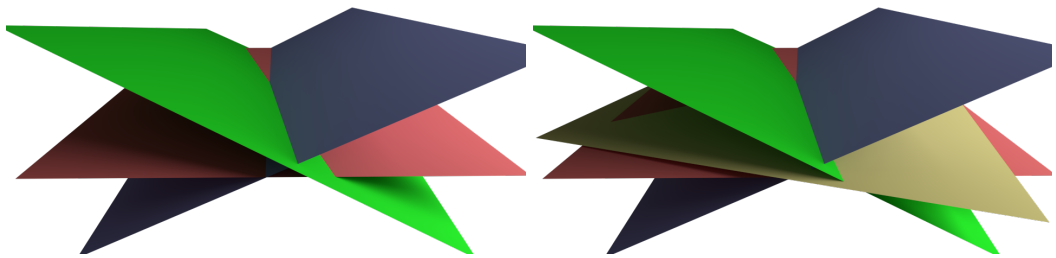

Figure 3.3: *Planar averages for a set of planes:* The left figure shows an input set of planes to be averaged. For this input set, the vAP and zAP are nearly identical. So, in the right figure, we display just the vAP (light yellow colored) overlaid over the inputs.

Condition for compatibility: A set of oriented planes in 3D are compatible wrt. the valley-set average when all three the eigenvalues of the Hessian H of Q are non-negative and there exists a unique largest eigenvalue.

The valley-set average of Q is a height valley of Q. A point p is on the valley-set average of Q if $\nabla Q(p) \cdot e_1 = 0$, where e_1 is the eigenvector corresponding to the largest eigenvalue λ_1 of the hessian H of Q.

The condition $\nabla Q \cdot e_1 = 0$ evaluates to a linear expression, and, thus, the condition is satisfied by points on a plane – the valley average plane (vAP).

Note that, similar to the case for 2D lines, for both the vAP and the zAP, the normal to the average plane can be computed with knowledge of just the normals $\{n_i\}$ to the input planes $\{p_i\}$. The vAP of a set of planes is shown in Fig. 3.3. The vAP and zAP formulations both produce a plane as the average. For families of two planes, their results are identical. For larger families of non-parallel planes, the vAP and zAP may differ (Fig. 3.4). The vAP is less sensitive to normal perturbations.

3.5 Averaging a set of lines in 3D

Both the zero-set average, and the valley-set average may by defined for lines in 2D and planes in 3D. The concept of signed-distance exists for these entities, as they are co-

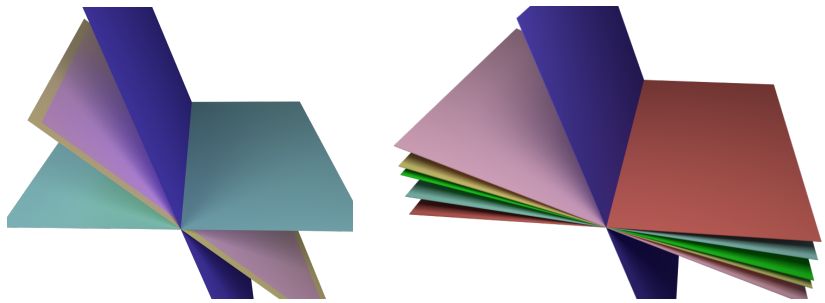

Figure 3.4: Left: zAP (pink) and the vAP (yellow) coincide for two planes (a slightly larger portion of the vAP is shown). Right: Given four input planes, zAP is more affected by the blue outlier than vAP.

dimension one manifolds that partition their ambient space. However, this is not true for lines in 3D. Thus, we can not extend the zero-set definition to average a set of lines in 3D.

For the valley-set definition, we consider the valley of the height-field Q that evaluates to the summed squared distances to the input lines $\{l_i\}$ at each point in space. Differently from the case for lines in 2D, we consider the valley-set determined by the vanishing of the Q's gradient in two directions. Thus, as per the terminology of [7], we extract for Q, a valley-set of type 1 which yields a one dimension solution set. We show that for compatible, configurations of lines, the valley-set is a line and we declare it to be the average of these lines.

Below we obtain expressions for the differential quantities of interest in characterizing the valley-set of Q. The squared distance Q_i between a point p, and a line l_i through point p_i, directed along t_i is computable by expressing it as the distance between p and the point q_i on l_i that is at a minimum distance from p, and then, computing $\|pq_i\|_2^2$ using Pythagoras theorem.

$$q_i = p + pp_i - \langle pp_i, t_i \rangle t_i$$

$$Q_i(p) = \|pq_i\|_2^2 = \|pp_i\|_2^2 - \langle pp_i, t_i \rangle^2$$

In any particular coordinate system, where op is represented by its coordinates x and op_i by x_i, the squared distance function Q_i to line L_i, and its gradient ∇Q_i evaluate to:

$$\begin{aligned}
Q_i(x) &= x^T x - (x^T t_i)^2 - 2x^T x_i + 2(x^T t_i)(x_i^T t_i) + x_i^T x_i - (x_i^T t_i)^2 \\
\nabla Q_i(x) &= 2x - 2(x^T t_i)t_i - 2x_i + 2(x_i^T t_i)t_i \\
&= 2(I - t_i \otimes t_i)(x - x_i) \\
&= A_i(x - x_i), \text{ where } A_i = 2(I - t_i \otimes t_i)
\end{aligned}$$

Note that A_i is symmetric, and, as each Q_i is a quadratic function in the coordinates, so is their sum $Q = \sum_i Q_i$. The gradient and the Hessian H of Q:

$$\nabla Q(x) = \sum_i A_i(x - x_i) = \Big(\sum_i A_i\Big)x - \sum_i A_i x_i \tag{3.7}$$

$$H(x) = \sum_i A_i \tag{3.8}$$

Condition for compatibility: A set of lines $\{l_i\}$ in 3D are compatible if the Hessian H of the summed squared distance function to $\{l_i\}$, has non-negative eigenvalues with a unique smallest eigenvalue.

For a compatible configuration of lines, the 1-dimensional valley-set of Q is a line and it contains point(s) where Q is minimized. Consider the following coordinate frame: denote by e_i the (orthogonal) eigenvectors of H, with corresponding eigenvalues λ_i ordered such that $i < j \Rightarrow \lambda_i >= \lambda_j$. Construct an orthonormal frame v_1, v_2, v_3 from the eigenvectors e_1, e_2 corresponding to the larger eigenvalues, and express Q in coordinate functions of this frame.

Then, for a point to be on the type 1 valley, Q should attain a minimum at that point, along e_1 and e_2. In our chosen coordinate frame, the above directional minima condition is written as, $\frac{\partial Q}{\partial x} = 0$ and $\frac{\partial Q}{\partial y} = 0$, with the second derivative requirement for minima along

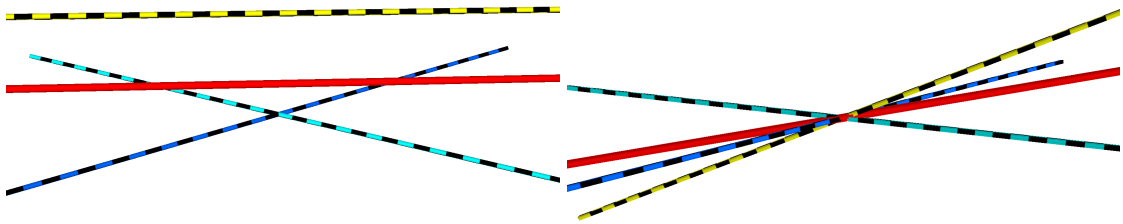

Figure 3.5: Three input lines (stippled), and their average computed using the proposed formulation (red). Left: Two input lines (cyan and blue) intersect and, the yellow line is parallel to their bisector. Their average (red) is parallel to the yellow line and to the bisector of cyan and blue. It is twice closer to that bisector than to the yellow line. Right: The average passes through the common point of intersection of all the input lines.

these directions: $\frac{\partial^2 Q}{\partial x^2} > 0$ and $\frac{\partial^2 Q}{\partial y^2} > 0$. As Q is positively unbounded and quadratic, it can either attain its minimum at a unique point, or, have a connected set of points as its minima.

First, consider the case that Q attains its minimum at a unique point p_{min}. Then, the valley condition is indeed satisfied at p_{min} (additionally, at such points, $\frac{\partial Q}{\partial z} = 0$ and $\frac{\partial^2 Q}{\partial z^2} > 0$). The valley condition then leads to two linear equations in three unknowns, that is satisfied along a line – the valley average line.

In the case that Q does not attain its minimum at a unique point, then, as λ_2 and λ_3 are strictly positive, H is not strictly positive definite and $\lambda_1 = 0$. Parallel lines belong to this case. Here, ∇Q is 0 along a one dimensional set. As the expression for ∇Q is affine in the coordinates, for the degenerate case, its vanishing is along a line – the valley average line.

In any particular coordinate frame, taking the dot product of the representation of ∇Q with that of e_1, e_2, applying the valley condition $\nabla Q(x) \cdot e_i = 0,\ i \in \{1, 2\}$ while substituting terms from Eqs. 3.7, 3.8, we obtain the following expression:

$$\nabla Q(x) \cdot e_i = 0,\ i \in \{1, 2\} \tag{3.9}$$

$$Hx \cdot e_j = \left(\sum_i A_i x_i \right) \cdot e_j \text{ for } j \in 1, 2 \tag{3.10}$$

Some results for the average line computed using this formulation are shown in Fig. 3.5.

3.6 Weighted average of a set of lines in 2D

3.6.1 Weighted zero-set average line (weighted zAL)

For a given set of oriented lines $\{l_i\} = \{(p_i, n_i)\}$, the weighted zAL is the zero level-set of the weighted summed signed distance field Z to $\{l_i\}$.

For the given set of lines,

$$Z(p) = \sum_i w_i(p_i p \cdot n_i)$$

$$\therefore Z(p) = 0 \Rightarrow \sum_i w_i(p_i p \cdot n_i) = 0$$

Replacing $p_i p$ by $op + p_i o$, where o denotes the origin of the chosen coordinate system, we have:

$$Z(p) = 0 \Rightarrow op \cdot (\sum_i w_i n_i) = -\sum_i w_i(p_i o \cdot n_i)$$

p is on the weighted zero-set average if the dot product of p's position vector op with the vector $(\sum_i w_i n_i)$ is a constant. The set of such p's lie on the line with normal directed along $n = \sum_i w_i n_i$.

3.6.2 Weighted valley-set average line (weighted vAL)

For a set of oriented lines $\{l_i\} = \{(p_i, n_i)\}$, the summed weighted squared distance function is $Q(p) = \sum_i w_i(p_i p \cdot n_i)^2$. Q is a quadratic polynomial. Hence, we can use the following matrix notation:

$$Q(x) = x^T A x + b^T x + c$$

Here $A = \sqrt{w}n_i \otimes \sqrt{w}n_i$. The weighted vAL is the valley of Q, and thus, is characterized by $\nabla Q \cdot e_2 = 0$, where e_2 is an eigenvector of the Hessian $H = 2A$ of Q that corresponds to the larger eigenvalue of H.

3.7 Properties of the average line and the average plane

For a list of compatible lines (resp. planes), the average line (resp. plane) that is computed using the formulations presented above satisfies the following set of symmetry and invariance properties:

- *Closure:* The average is a line (plane).
- *Exactness:* For a single line (plane), the average is the line (plane) itself.
- *Co-incidence:* The average of a list of co-incident (incident on a common point p) lines (planes) is co-incident with them at p. In particular, for a list of identical lines (planes), the average is exact again.
- *Order invariance:* The average is a set property invariant under permutations of the list.
- *Similarity invariance:* The construction of the average line (plane) commutes with similarity transformations.
- *Parameterization invariance:* The average does not depend on the parameterization of the lines (planes), but depends only on their spatial embedding.

3.8 Geometric interpretations

Interpreting the vAL

The level sets of the summed squared distance function for a set of compatible 2D lines are either lines, or, ellipses. When the level sets are lines, the height field surface $z = Q(x, y)$ is a parabolic cylinder, and, the vAL is the line of points (along the axis of the cylinder)

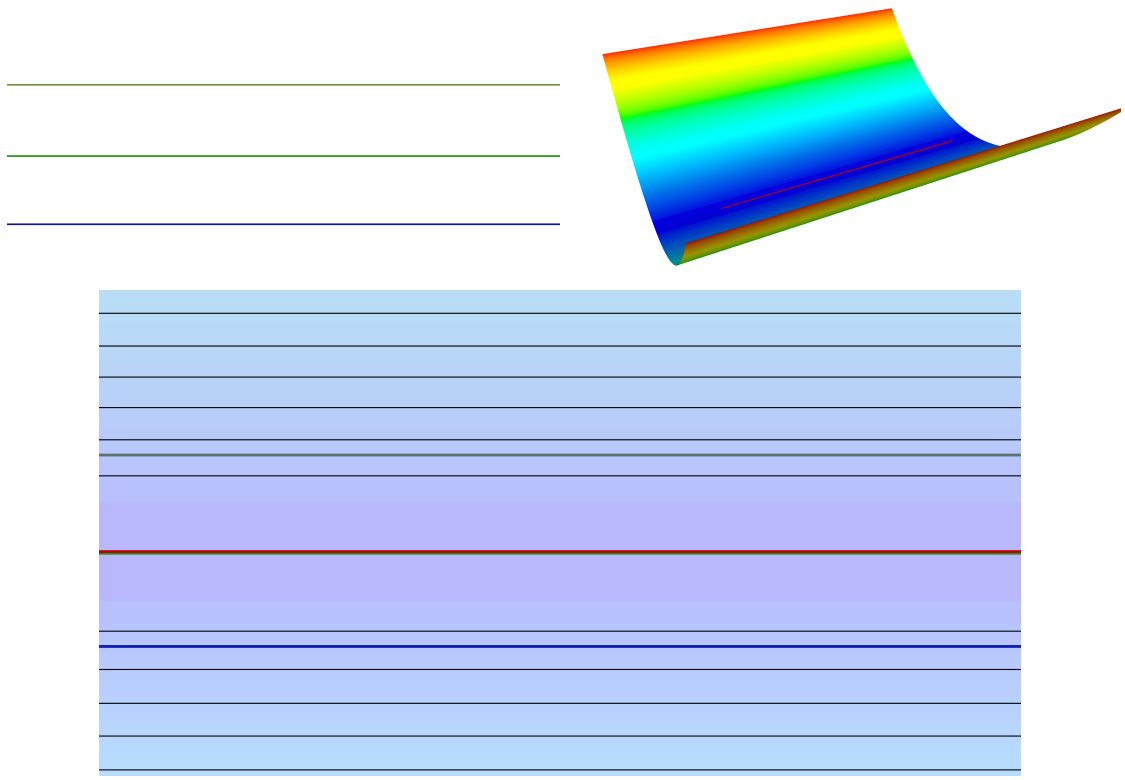

Figure 3.6: Top Left: Three parallel input lines. Top Right: The average line (red) and the summed squared distance field visualized over it. Bottom: The level sets (black) of the summed squared distance field are lines, and, the average line (red) is the line on all points of which, the gradient is zero.

where the gradient vanishes. When the level sets are ellipses, the vAL is directed along the larger principal axis, and, passes through the unique point where Q is minimum.

Fig. 3.6 and Fig. 3.7 visualize the geometric entities important for understanding the vAL for two configurations of line of interest to us: nearly parallel lines, and parallel lines.

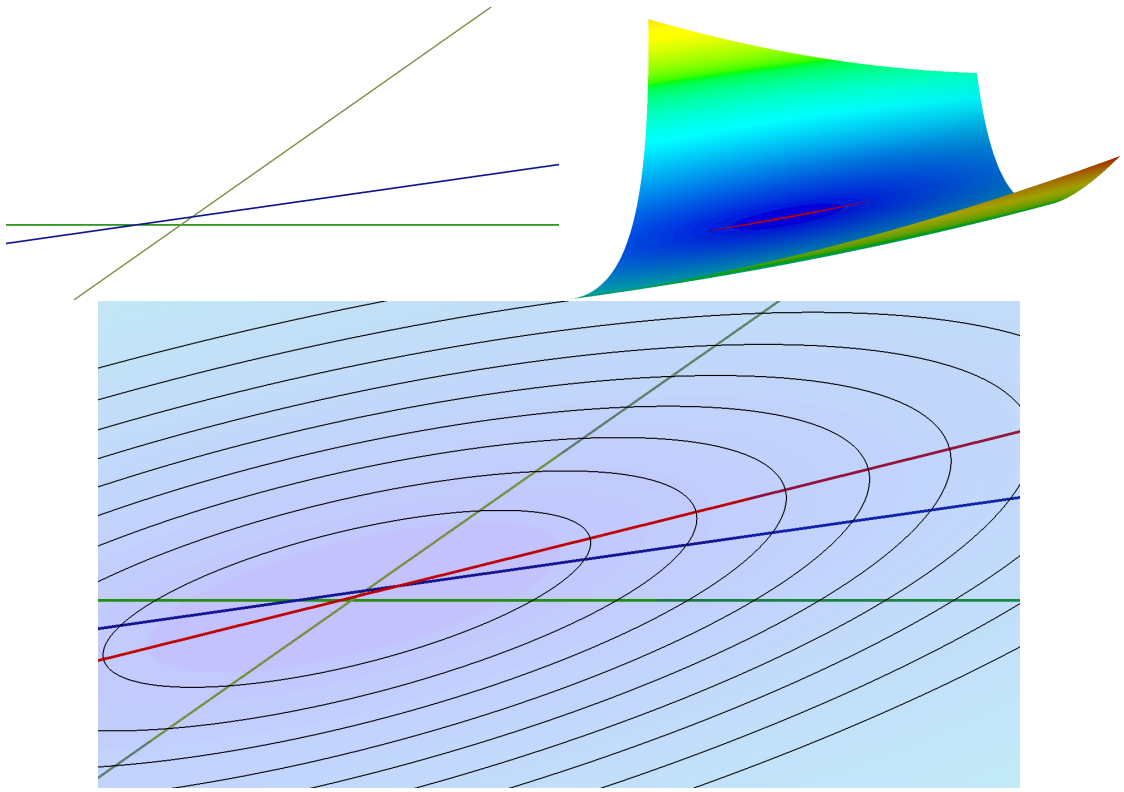

Figure 3.7: Top Left: Three input lines. Top Right: The average line (red) and the summed squared distance field visualized over it. Bottom: The level sets (black) of the summed squared distance field are ellipses, and, the average line (red) is directed along a principal axis.

CHAPTER 4

PRIOR ART

In this chapter, we first present formulations for averaging sets of certain mathematical objects. Then, we discuss prior art related to averaging shapes.

4.1 Averaging

Averaging values and points: The average of a set of n numbers is the sum of the numbers divided by n. In an affine space, fixing a frame of measurement, the coordinates of the centroid of a set of points $\{p_i\}$ is obtained by averaging each coordinate of the point-set. Note that, the centroid is the unique zero of the vector field $V(x)$ defined as $V(x) + x = \frac{1}{n}p_i$. An extension of this idea [8] is used to define the Riemannian center of mass. Such definitions of the average require only an affine structure. On the Euclidean line, the additional metric structure allows for a reinterpretation of the average of a set of numbers as the minimizer of the variance (see Sec. 2.2). The Fréchet mean extends this concept to metric spaces. If $\{p_i\}$ belong to a metric space, the Fréchet variance at any point is defined as the sum of squared distances to $\{p_i\}$, and, the Fréchet mean is defined as the point which minimizes the variance.

Averaging functions using point-wise correspondence: Given functions $\{f_i\}$ that map from a common domain to (subsets of) a space where a notion of averaging exists, an average can be constructed as the function $f(x) = Average(f_i(x))$. For example, for functions mapping to the Euclidean space, one candidate for the Average function is $Average(f_i(x)) = \sum_i \frac{f_i(x)}{n}$.

4.2 Averaging shapes

One could envision using an average of functions to compute the average of a set of shapes. For example, for a set of curves, if all input curves are star-shaped closed-loops, one could use polar coordinates, $r_i(\theta)$, of each curve around the centroid C_i of its kernel [9] and define the average curve using polar coordinate distance $m(r_i)$ around the centroid of the points C_i. A similar approach may be used in 3D [10].

Unfortunately, in many applications, some input shapes may not be star-shaped. Thus, in a number of prior works, averaging shapes involves two steps:

- *Establishing correspondence*, which, for curves, amounts to parameterizing each one as $C_i(s)$, and
- *Constructing samples* on the average curve that are each defined in terms of corresponding points $C_i(s)$, often (although not always) as their centroid.

Thus, we first review prior art that establishes correspondences between shapes.

4.2.1 Establishing correspondence

Arc-length correspondence: For a set of curves, points that have the same normalized arc length can be put in correspondence. However, curvature variations in the inputs can lead to poor results as, in constructing the average sequence of points, the parameterization used for each curve is arrived at independently (using only the arc-length information of that particular curve), without consideration of the parameterizations of the other curves (Fig. 4.1).

Landmark correspondence: Sparse correspondences may be specified by the user and used as constraints for obtaining dense correspondences, or, correspondences can be automatically extracted by detecting salient feature points such as points with high curvature or points which possess a similar response to a feature detector [11, 12, 13]. When the set of shapes are known to belong to a particular category, such as human faces, template defor-

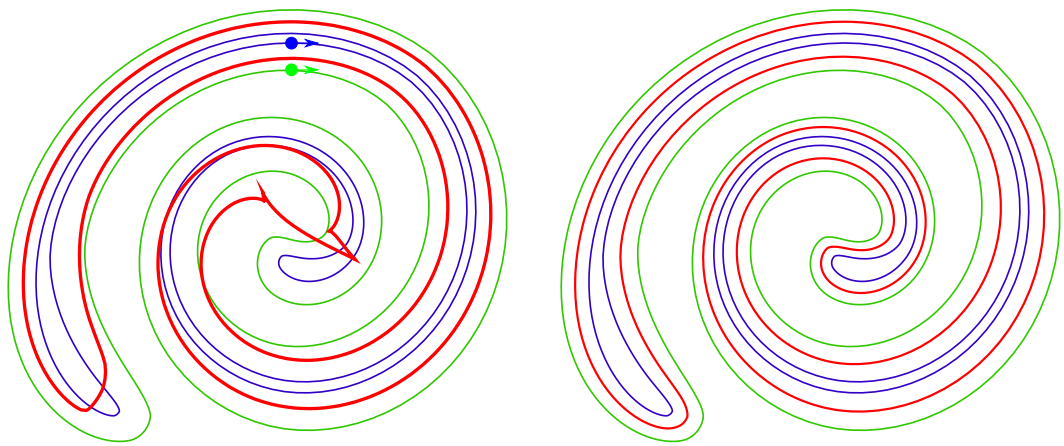

Figure 4.1: Left: The average (red) of two curves (blue & green) computed using corresponding points when the curves are parameterized by arc-length (the arc length value of 0 is assigned to the dotted locations and increases along the directions indicated). Right: The average (red) computed by techniques presented in this thesis.

mation techniques may be used [14, 15].

Fast correspondence finding using dynamic programming: A class of correspondence finding approaches consider finding correspondence between sets of discrete samples on the inputs. The edit-distance [16] quantifies differences between strings of symbols of size m and n by the optimal number of edit operations (insert, delete, change) that are required to transform one to the other. A dynamic-programming approach for computing the recursive edit-distance formulation in time $O(mn)$ is presented in [17].

For a pair of curves, seeking sampled correspondences, and building in assumptions about the monotonic nature of desired correspondences, we can obtain globally optimal correspondences using a similar dynamic-programming solution. Given a matching cost (measure of disparity for a pair of samples, one on each input curve), the dynamic time warping algorithm [18] can be used for finding correspondences between (or an alignment of) samples on two curves. For the case of two closed-loop curves, these algorithms require an initial starting correspondence between a pair of samples. Assuming that each curve is represented by n samples, an optimal cyclic alignment using edit operations (insert, delete,

change) costs $O(n^3)$ ([19], Lemma 3.1). The authors of [19] also discuss an algorithm for reducing this cost to $O(n^2 \log(n))$. In [20], the authors present an $O(n^2 \log(n))$ algorithm for computing optimal cyclic-correspondences using dynamic time warping.

Minkowski correspondence: The Minkowski average establishes correspondence between points with the same normal. Real-time solutions have been proposed for displaying weighted Minkowski averages of two [21] or more [22] polyhedra. Unfortunately, Minkowski maps fail to satisfy desirable properties. For example, the Minkowski average of a non-convex set S with itself is not S.

Tangent based correspondences: In [23], the authors use a dynamic programming approach to find correspondences between two curves that yields the best possible alignment of tangent vectors at sampled locations.

Normal and radial correspondences: One could use one of the input shapes, say S_1, as base and express each point of the other shapes, S_i, as the normal offset [24] of a point p on S_1 along the normal at p, $n_0(p)$. Thus, $S_i(p) = p + r_i(p)n_0(p)$. The average shape S may then be defined as $S(p) = p + m(\{r_i(p)\})n_0(p)$ using the arithmetic mean $m(\{r_i(p)\})$ of the offset distances. For the correspondences defined in this manner to be unique, it is required that the shapes be normal compatible [25]. The above idea also works for radial offsets [24], which places the point p in correspondence with its closest point of S_i. In both cases, the result depends on which input shape is chosen as base.

Correspondences via a common ('base') shape: It is possible to find point-wise correspondences between multiple shapes by finding for each point on each shape, a corresponding point on a common 'base' shape. Then, the set of points $\{p_i\}$ (one of each shape) that correspond to the same base shape point p are in correspondence. In this vein, in [26], Bier

and Sloan involve a simple, user-selected, intermediate surface to support a two-step texture mapping process. In [27], maps for a set of genus-0 meshes are obtained by mapping each mesh onto a sphere.

The 'base' shape that establishes correspondences may also be defined procedurally, and, in a level of detail hierarchical representation [28], correspondences computed on a coarse scale can be used to inform correspondences between the finer scale representations. In [29], maps between triangle meshes are derived from their progressive simplifications and from establishing correspondence at the crudest versions. In [30], maps between surfaces are obtained by decomposing each surface into the same structure of patches.

Correspondences through physically based deformation models: [31] phrases the computation of correspondences between two polygonal curves as a morphing problem. It models work done for stretching and bending segments and corners, and then, computes vertex correspondences that minimize the work that has to be done for deforming one curve to another using an edit-distance type dynamic-programming formulation.

4.2.2 Extracting features from a scalar field

One way of constructing a representative shape is indeed by first establishing point-wise correspondences explicitly. However, as described in Sec. 6.3.3, the average can also be computed by constructing a symmetric scalar field and extracting a set of points using this field. This section discusses prior-art for approximating distance fields, and for extracting zero-sets and valley-sets.

Computing distance fields: There are very few shapes for which the distance function has a closed form expression. Indeed even computing the distance from a particular point to an ellipse requires the solution of a quartic [32] or a transcendental equation [33], and, is best done using iterative methods.

Rather, one method of computing approximations to the distance function for a shape is using a spatial grid. A discrete grid representation of an embedded shape is as the set of grid-cells it stabs. For shapes represented implicitly, for example, such a representation can be computed by the zero-set extraction methods discussed below. For triangle-meshes, such a discrete representation can be computed by performing intersection tests for each triangle of the mesh against a grid-cell. The separating axis theorem for convex shapes (for two convex shapes, there exists a hyperplane that separates them) can be used to speed up these computations [34].

One approach to computing approximate distance fields from a discrete representation is the use of a 'propagating stencil' constructed for the grid and repeatedly 'applied' to each grid cell. For a particular grid cell, the application step first centers the stencil over each grid cell, and computes the element wise products of the centered stencil with the neighbors of the grid cell (mathematically, if A denotes a table of current distance values centered around the current grid cell presented in a matrix, and, B denotes the values of the propagating stencil, then $C[i,j] = A[i,j]B[i,j]$). The new distance value to the grid cell is assigned as the minimum of the entries in the resulting table of values $\min_{i,j} C[i,j]$. Note that, thus, the value at a grid cell is not expressed as the linear combination of its neighbors as is the case during the application of a linear filter. Sequential and parallel approaches to computing the 'distance transform' have been proposed [35, 36]. Different stencils are proposed, and their approximation accuracy studied, in [37].

Fast marching methods [38] compute approximations to the distance field for a discrete shape representation by intelligently propagating distance information away from the shape. These methods use an idea similar to the Dijkstra's algorithm and, make a pass over the set of grid-cell, ordered by distances that are updated online as the pass progresses.

Zero-set extraction: Given a grid on the vertices of which a function f has been evaluated, the marching cubes [39, 40] and dual-contouring [41] algorithms allow the extraction of

zero-crossing contours by constructing, for each grid-cell, local geometry in accordance with the evaluated f values.

Valley-set extraction: There are different characterizations of the valley-set of a 2D height-field. Arthur Cayley [42] and J.C. Maxwell [43] give an integral formulation of the valley-set (called lines of watercourse in [43]) as the slope line (solution curves under gradient flow) emanating at a point an infinitesimal distance away from a saddle point in a direction of decreasing gradient and approaching a minima of the height-field. To obtain a closed slope line (i.e., containing all it's limit points), both the saddle point and the minima point, which are fixed points of the gradient flow are included. Similarly, the ridge-set (called lines of watershed in [43]) is the closure of the slope line emanating from near a saddle point and terminating near a maxima of the height-field. Note that a valley-line on a height-field is a ridge line on the negative height-field.

These descriptions are very instructive, and are a precursor to modern Morse theory [44] – the study of topological properties of a manifold through the use of functions over the manifold.

The watershed transformation [45, 46] used in image segmentation, given a set of seed points at low heights (for an image, the height at a pixel is the intensity of the pixel), simulates the 'flooding' of a 2D height-field and computes the watershed lines as the set of points where the flooding would cause a water body that contains one seed to become connected with a water body containing another seed. The watershed lines are the boundaries of different objects in the image.

In [47], the above formulation is used for extracting ridges from 2D height-fields represented by their samples on a grid (gridded digital terrain models). [47] finds saddle points with sub-pixel precision on a filtered digital terrain model, and, computes the integral curve of the gradient vector field by an adaptive step size Runge-Kutta method [48] (Ch.

17) which requires at each 'time-step', computing approximations of the derivative of the height-field.

In [7], the valley of a height-field is defined as the set of points where a minima of the height-field is attained in a particular principal direction of the height-field. This characterization, along with local bi-cubic fits to a scalar field represented using a grid is used in [49] to compute grid cells that intersect with a ridge or a valley.

Given two vector-fields represented by samples on a grid, [50] presents a 'parallel-vectors' operator for finding grid-cells where the two vector-fields are parallel. The pair of vector-fields are interpolated to the interior of the cells. Depending on the interpolation function used, either analytic solutions, or, Newton-Rhapson iterations are used for finding if a point inside the grid-cell satisfies the valley as characterized by [7].

The authors of [51] define the ridge and valley-set as the locus of points with maximal curvature on each contour line.

The different characterizations are critiqued in [52], which notes the merits of the definitions of [42]. For compatible configurations of lines, the various definitions of the valley agree on the solution set – the vAL as discussed in Ch. 3.

As opposed to computing valleys of a scalar field, works such as [53, 54] seek to trace ridge and valley curves on meshes and surfaces. Valleys on surfaces are defined using frames local to the surface, as opposed to that on height-fields where, there exists, a particular distinguished 'up' direction.

Field evolution and stationary field zero-set extraction: The works presented above compute a representation of, or extract features from, a static scalar field. Active-contour models [55] use level-set [56] methods for evolving surfaces and curves. Starting with an initial shape representation and a cost functional, one computes: (a) an approximation to the signed distance field to the initial shape Q, and, (b) equations for iteratively updating Q so as to reduce the evaluation of the cost functional for the shape corresponding to $Q = 0$. Q

is evolved as per the update equations, and (typically the) zero set for the steady state of the evolved field is extracted as the evolved shape.

Particularly relevant amongst such approaches is an approach for elastic shape averaging. Given two input shapes, a deformation measure quantifies the energy expended in deforming one to the other. In [57], the authors define a deformation measure between two shapes and compute as the average, the shape that minimizes the summed deformation to the inputs. The authors use a staggered optimization process that alternatively updates an estimate of the average shape and the deformation measure from it to each input shape using grid based discrete representations.

4.2.3 Medial axis and ball morph

The medial axis [58, 59] of a shape is the set of points with multiple closest projections to its boundary. The medial axis has numerous applications [60], and, is a very well-studied object in computational geometry [61, 62, 63].

While the medial axis is unstable – minor alterations to a shape can alter its medial axis significantly – filtering approaches such as [64, 65] provide a practical method for capturing the medial symmetry set of a shape.

The use of the medial axis concept for symmetric correspondence establishment between two ball-compatible shapes is described in the 'ball-map' [66]. The correspondences established by the ball-map have been used to develop morphs between compatible pairs of shapes [67]. We present the essential ideas of this work in Sec. 6.2.

The medial axis of the symmetric difference of two 'compatible' 2D curves corresponds to the average curve defined in this thesis. Hence, our definition of the average generalizes this use of the average to more than two curves. Our algorithm for computing a representative generalizes (to more than two shapes) previously proposed sampling approaches [68, 69] of the medial axis.

4.2.4 Staggered, iterative optimization

Algorithms of the k-means clustering family [70] seek to compute an unknown object that is determined by local spatial information by a fixed point iteration. The expectation maximization algorithm [71] also solves for both an unknown set of optimal parameter values, and, a set of latent variables when the conditions on both the sets of unknowns are too hard to be solved for simultaneously. The algorithm assigns a guess value to the latent variables, and solves for the optimal parameter values for the current guess. The latent variables are then re-estimated using the parameter values just computed.

The moving least squares [72, 73] method for scattered data interpolation, given a set of data-points and associated scalar values, solves, for any query point, the weighted least squares problem for the data-points to obtain coefficients for a polynomial. The polynomial's evaluation at the query point furnishes the interpolated scalar value.

A principal curve [74] is a curve defined by sample point data through a similar staggered iterative process of approximating the curve locally by deciding contributing points for a local region, and, using the approximated curve for redistributing contributions from points.

In registration using iterative closest projections (ICP) [75], given a nominal surface and a cloud of points, a rigid transformation that 'best' aligns the point cloud to the surface is computed by iterating the staggered process of a) computing correspondences and b) computing and applying best aligning transformation. Various errors can be minimized when seeking the 'best' aligning transformation. Particularly relevant to this thesis is the minimization of the summed squared distances from each point of the point-cloud to the tangent plane at the point of closest projection on the nominal surface [76]. Important observations about the geometry of the squared distance function to surfaces are presented in [77] which further motivates the use of the squared distance to tangent planes for ICP.

4.2.5 Guaranteed homeomorphic approximations

Two shapes are homeomorphic if there exists a homeomorphism between them. While this definition is existential, given a procedure for mapping between two shapes, one can study conditions on the shapes for which the mapping establishes a homeomorphism between the two shapes. In this manner, conditions for the closest projection map between two shapes to be a homeomorphism between them are presented in [25].

[78, 79] reconstruct curves and surfaces from a set of point-clouds by providing constructions based on the Voronoi diagram [80] of the point-set. Assuming that the point-clouds are obtained by sampling an input (a smooth curve or surface), they provide conditions on how densely the point-clouds should sample the input such that their reconstruction is homeomorphic to the input.

Given a surface and a set of points, [81] discusses conditions for the surface to be homeomorphic to the restriction of the Voronoi cells of the Voronoi diagram of the set of points to the surface.

4.3 Relation of our contributions to prior-art

The approaches proposed in this thesis characterize the representative shape as zero-sets and valley-sets of appropriate height fields. Thus, these sets could be extracted using the ideas from Sec. 4.2.2. However, we use the derived expressions for the zero-set and valley-set of a set of lines and planes to suggest a grid-free, iterative approach.

In our thesis, a novel theoretical contribution is proposing a definition of the average shape for a set of compatible shapes as the height valley of the summed squared distance functions to the shapes. Armed with this definition, we investigate lines and planes, arriving at compatibility conditions which will ensure that the computed average is unique.

We note in Sec. 4.2.2 that techniques to compute distance fields typically grid the embedding space, and, techniques to compute the zero-set or the valley-set of a scalar field assume a dense-grid representation for their operation.

In our thesis, we consider the case where the scalar field is a function of the distance fields induced by a set of input geometric objects. For this case, we propose an alternate computational scheme. We do not compute a dense approximation of the distance fields for each shape. Rather, we construct local approximations to the distance fields during the Snap operation. Given a sampling location, the distance field to each shape is approximated as the distance to the tangent plane on the input shapes that corresponds to the sampling location.

Thus, the computational ideas of our work are similar to works on dynamic, iterative algorithms presented in Sec. 4.2.4. Our algorithm staggers correspondence finding and projection onto a local average approximation.

Some key differences from prior art are that: a) we enforce symmetry (wrt. permutations of the inputs) in selecting the data we use for computing our local field, b) create a directional field by using the summed squared distance to lines and planes rather than to points, c) we use the field to compute a local implicit surface to project onto, which is not necessarily given by the zero-crossing of the field (e.g., the valley formulation) and, d) we use local geometric information about the input to decide the direction of projection, while still being able to argue for the symmetry of our construction.

The closest projection queries can be made computationally efficient by using connectivity information of the input shapes. As the correspondence finding step is separated from the local approximation step, this approach has the following benefits:

- We can separately consider questions about the topological validity of the set of correspondences yielded by the correspondence finder. Using the results of prior art [25], we have (admittedly strong) a-priori conditions on the inputs for ensuring that the closest projection correspondence finder yields homeomorphic correspon-

dences. Aside from the a-priori compatibility conditions, we can also propose post computation checks that decide the usefulness of the computed result – we propose that the result be considered useful if it establishes homeomorphic closest projection maps, or, for polygonal inputs, when the result is a curve or surface that well approximates each of the inputs.

- The correspondence step may be altered to not necessarily yield closest projection correspondences (see Sec. 6.5), but yield correspondences that also consider the connectivity of the input shapes.
- We can alter the field whose zero-set or valley-set we seek, while using the unaltered correspondences. This idea is similar to weighted moving least squares, but, rather than weighting based on spatial proximity of the query point, we weigh each field induced from a shape either globally to effect morphing (see Sec. 5.5), or locally to effect smooth joins between curves (see Sec. 9.3.2).

CHAPTER 5

AVERAGING CLOSED, COMPATIBLE PLANAR CURVES

Our algorithms for computing average shapes start with a seed shape. This is, generally, one shape in the input set. Also note that the result is independent of which shape is chosen as seed. First, we sample 'query' points on the seed shape. Each of these query points is mapped to a point on the average. We assume that these 'snapped' points on the average have the same connectivity as that of the query points on the seed shape. Thus, for each type of shape (curve, surface), we need to develop 'Snap' methods that snap query points onto the average.

In this chapter, we first describe the approach for snapping query points onto an average curve for a set of compatible, closed-loop, planar curves. Then, we describe a modification 'Trace' that replaces the global closest projection computation step of Snap with a local query for better computational complexity.

5.1 Snapping for curves in the plane

The Snap method uses the fact that the average for a set of compatible lines is a line (Sec. 3.3). The Snap algorithm is presented in Alg. 1. At each 'Move' iteration (Fig. 5.1), we:

- For each curve C_i, compute the closest point p_i on it from the current p. p_i is said to be in closest projection correspondence with p as it is the image of p under the closest projection map (the map that associates p with its closest projection on C_i)
- Compute the average line by averaging tangent lines to the input curves at the points of correspondences $\{p_i\}$.
- Update p by projecting it onto the average line.

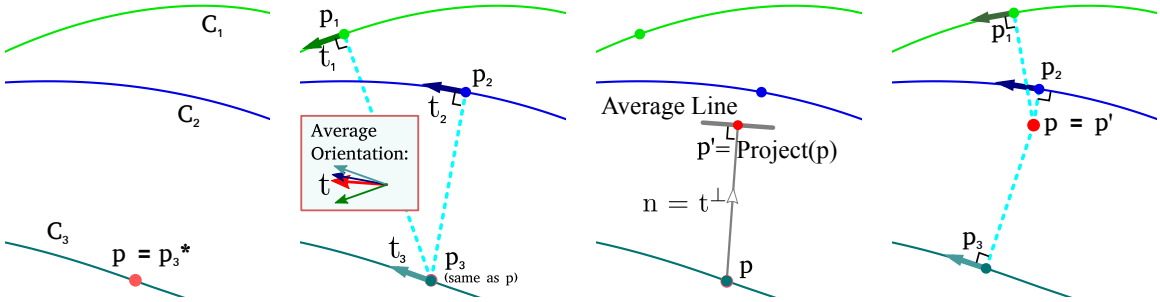

Figure 5.1: A 2D sketch of the initial steps of the Snap procedure. From left to right: (a) Shows three input curves $\{C_i\}$ and the initial location of p, which is initialized to a sample p_3^* on the seed curve C_3. Each 'Move' iteration of Snap will update p. (b) and (c) demonstrate the two phases in a single Move iteration. (b) Shows the closest projections $\{p_i\}$ of p, and the associated tangents $\{t_i\}$. The inset shows the direction of the "average" tangent t in red. (c) Shows how we project p to the average line (gray) by moving orthogonal to the average line's direction t, i.e., along the normal to the average line n. p' will become the new p for the next iteration. (d) Shows the projections of the new p onto the input curves and marks the start of the next Move iteration.

Algorithm 1 Snap method for closed planar curves

```
procedure SNAP(p_0, ℂ = {C_i})
    p = p_0
    repeat                                          ▷ Sequence of Moves
        for i ∈ {1, ... |ℂ|} do
            p_i ← ClosestProjection(p, C_i)
            l_i ← (p_i, t_i)                        ▷ Line tangent to C_i at p_i
        p ← Project(p, AverageLine({l_i}))          ▷ Project p on average line to {l_i}
    until Converged(p, {C_i})
    return p
```

The key sub-routine invoked by Snap is Project, which projects onto the average line. Depending on whether we use the zAL or vAL formulation to compute the average line, we call the resulting curve computed the zero-set average curve (zAC) or the valley-set average curve (vAC). In the next section, we describe the details of Project.

5.2 Projecting onto the average line

Given a set of lines $\{l_i\}$, the average line can be computed using either the zAL or vAL formulation of Sec. 3.3. Subsequently, a point p can be projected onto the computed line.

However, as we know of a condition (for example, for the vAL, the summed signed distance should evaluate to zero) that must hold for a point to be contained on the average line, we can Project onto the average line without computing its explicit representation.

Avoiding computing the average explicitly also aids numerical stability. For the vAL, explicitly computing the average line $l(p, t)$, requires computing a point on it, and its direction. As the vAL passes through the point where the summed square distance to the set of lines $\{l_i\}$ is minimized, this minima point can be used as p.

However, this method fails when the lines l_i are (nearly) parallel, because the minimizer may be (nearly) at infinity, which leads to numeric instability. Rather, note that the direction of the average tangent t, and thus, the average normal n can be computed robustly using only information about $\{t_i\}$. Additionally, we also have implicit conditions for a point to lie on the average line (the point should be on the zero-set of the signed distance function (Eq. 3.2), or the valley of the squared distance function (Eq. 3.5)). Thus, our projection approach directly formulates projecting p onto the average line as finding the vector $v = sn$ such that the required conditions are satisfied at $p+v$ (this implies that $p+v$ is on the average line).

Our project algorithms for both the zAL and the vAL compute the desired v and are presented below.

5.2.1 Projecting on the zAL

The details of *Project* for projecting onto the zAL of lines is shown in Algorithm 2. Note that we do not require normalization of the computed n.

5.2.2 Projecting on the vAL

The details of *Project* for projecting onto the vAL of lines is shown in Algorithm 3. The process computes the average normal n (using the formulation of Sec. 3.1) and the displacement along it to lie on the valley-set average line. In our notation, vector $n_i = (n_i.x, n_i.y)$.

Algorithm 2 Project on the zAL

```
procedure PROJECT(p, {l_i} = {(p_i, n_i)})
    n = (0, 0)
    for each line l_i do
        n += n_i
    for each line l_i do
        num += (pp_i · n_i)
    den = n · n
    p = p + (num/den) n
```

Algorithm 3 Project on the vAL

```
procedure PROJECT(p_0, {l_i} = {(p_i, n_i)})
    a = 0; b = 0; c = 0; n = 0; d = 0
    for each line l_i do
        a += n_i.x^2
        b += n_i.y^2
        c += n_i.x n_i.y
    α = atan2(2c, a − b)/2
    n = (cos α, sin α)
    for each line l_i do
        num += (p_0 p_i · n_i)(n · n_i)
        den += (n · n_i)^2
    p = p_0 + (num/den) n
```

5.3 Tracing: Accelerated Snap

Snap (Alg. 1) calls the ClosestProjection algorithm. The algorithm computes, for a given query point p and an input curve C_i a point p_i that is on C_i and is closest to p. We say that p_i is the point on C_i that corresponds to p.

A ClosestProjection algorithm would, for a piecewise approximation to a curve, need to query each piece for the closest point, and, take the minimum over these. Thus, for example, for a piecewise linear closed-curve approximation, the complexity of the ClosestProjection algorithm is linear in the number of vertices in the approximation.

Our Trace algorithm described below (Alg. 5) builds a polygonal approximation of the average sequentially along the average curve, a vertex at a time. For any input curve C_i, Trace uses the result of the previous correspondence computation step as the starting point for a local search for the new corresponding point, thereby improving computational efficiency.

Algorithm 4 Incremental Snap method

```
1: procedure INCSNAP(p, ℂ = {C_i}, {u_i})
2:     repeat                                          ▷ Sequence of Moves
3:         for i ∈ {1, ... |ℂ|} do
4:             p_i ← IncrementalClosestProjection(p, C_i, u_i)
5:             l_i ← (p_i, t_i)                        ▷ Line tangent to C_i at p_i
6:         p ← Project(p, AverageLine({l_i}))          ▷ Project p on average line to {l_i}
7:     until Converged(p, {C_i})
8:     return p, {p_i}
```

We compute the first (*seed*) vertex for snapping as the left-most point of the input set. This step requires traversing each curve once, before the tracing starts. Other, more robust, solutions may be used. For example, one may consider a small set of seed candidates randomly chosen on the input curves and select the one for which the minimum distances to the other curves have the smallest sum. The seed vertex is snapped to obtain a sample p

Algorithm 5 Trace

```
procedure TRACE({C_i})
    u ← left-most point of {C_i}
    {u_i} ← ClosestProjection(u, {C_i})
    {l_i} ← {(u_i, t_i)}                          ▷ Line tangent to C_i at u_i
    PAC.AppendVertex(Snap(u))
    while not Closed do
        t ← TangentToAverageLine({t_i})
        u, {u_i} ← IncSnap(u + ϵt)
        PAC.AppendVertex(u)
```

on the average curve, and, the correspondences $\{p_i\}$ (closest projections to each $\{C_i\}$) are noted.

Then we repeat the following process. We consider the sample u last appended to the list of samples. We use the corresponding points $\{u_i\}$ to u on each $\{C_i\}$ and the tangent lines $\{l_i\}$ at $\{u_i\}$ to compute the tangent of the average curve using the vAL or zAL formulation[1]. t is the tangent to the average curve at u. We compute a candidate point $p = u + \epsilon t$ as an extrapolation of u along the tangent to the average curve. Then, we compute a new sample v by snapping p and append it to the list of samples.

IncrementalClosestProjection: For trace, our IncSnap algorithm has the parameter signature $IncSnap(p, \{C_i\}, \{u_i\})$ and is invoked with $p = u + \epsilon t$. Note that it accepts 'previous' correspondences – correspondences $\{u_i\}$ for the point u last appended to the list of samples. For a curve C_i, the $IncrementalClosestProjection$ step of IncSnap does not perform a global search over C_i for the closest point p_i to p. Instead, we start the search at the corresponding point u_i of u (the previous snapped point) and stop the search at the first point p_i along C_i where the distance to p reaches a local minimum.

Trace stops (i.e., Closed becomes True), when the distance from the new vertex v to the first vertex of the list of samples is less than ϵ. We test whether v projects onto the edge created from the first two samples on the average. If so, we do not append it.

[1]Note that we do not need to recompute u_i and t as these are saved as the results of the previous snap.

The sampling density and hence the number of samples appended to the PAC are controlled by the step size ϵ. For example, we set it to 1/300 of the average length of the input curves if we wish to create an average curve with about 300 vertices. A large step size may reduce the smoothness of the average. One may of course subdivide the resulting curve for a smoother result.

5.4 Experimental results

In this section, we first report performance measurements and show a variety of qualitative results. Then, we discuss an application the approach presented here to morphing.

Convergence of Snap: While Snapping a point, if two successive Move steps yield locations u, v such that the distance between u and v is less than $10^{-8}bl$ where bl is the length of the smaller side of the bounding box of the union of the input curves, we declare convergence. The average number of move iterations that were needed to converge to the vAC from an arbitrary point p in the plane were 2.03.

Overall performance: We used a machine with 8 GB of RAM and a 2.2 GHz quad-core processor. Our single threaded implementation performs an average of 15,590 Snap operation per second on an input set of 768 vertices. Performance drops to 8,104 operations when we double the vertex count.

Incremental solution: Switching to locally computing the closest projections as described in Trace, the performance improved by a factor of 7 when the input has 768 vertices, by a factor of 14 when the vertex count is doubled, and by a factor of 38 when it is quadrupled.

Diversity of compatible input sets handled by Snap: We tested Snap on a large number of configurations of input curves. A typical set of examples are shown in Fig. 5.2.

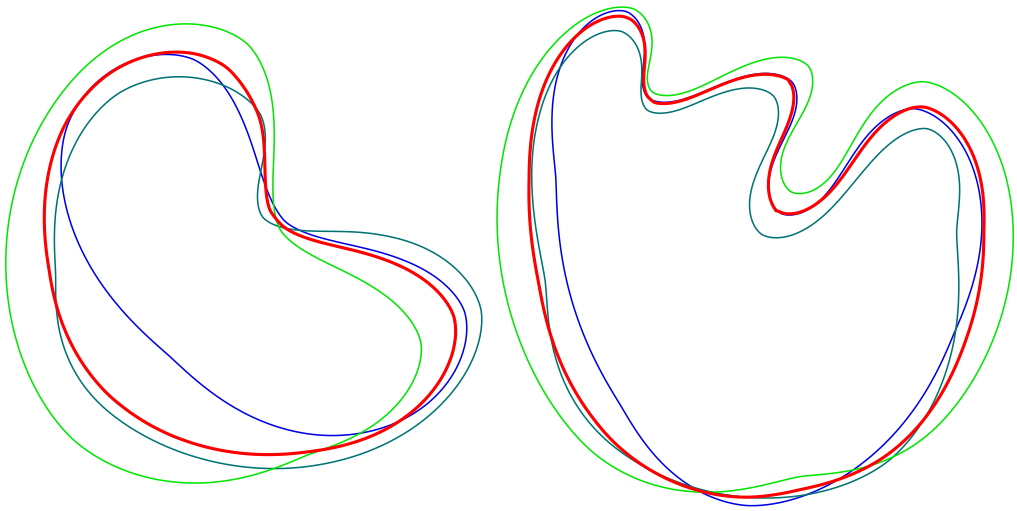

Figure 5.2: Example of sets of compatible inputs with their vAC (red).

Snap and Trace yield a Jordan curve for a variety of configurations, including configurations where the curves are far from similar (see for example Fig. 5.3).

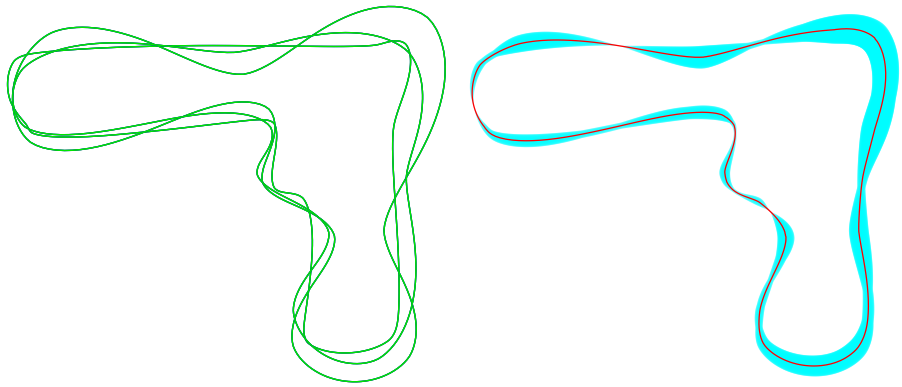

Figure 5.3: Three input curves (green) and the resulting vAC with inflation (cyan).

5.5 Application: Mixing shapes via weighted averaging

To compute a weighted vAC (resp. zAC) of $\{C_i\}$ with weights $\{w_i\}$ that sum up to 1, we modify the Project algorithms to, instead of projecting onto the average line, project onto the weighted average line (Sec. 3.6).

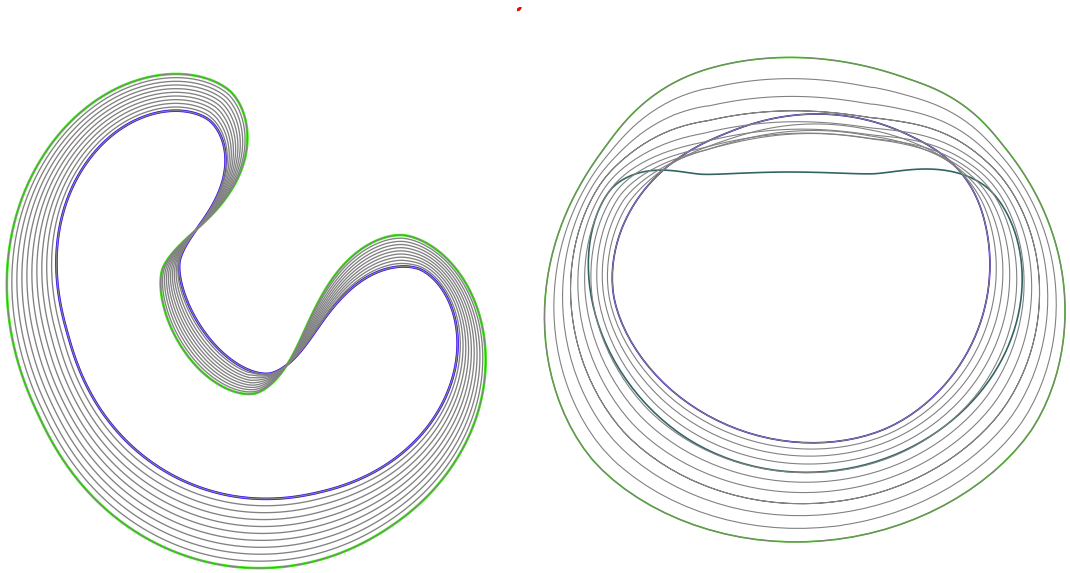

Figure 5.4: Left: Frames (black) of an interpolating morph between two input curves produced by using a weighted vAC formulation. Right: Frames of an animation defined by three control curves using quadratic Bézier weights of $(1-t)^2, 2t(1-t)$ and t^2. As expected, the central curve is not interpolated.

The choice of the weights or their variation with time or some other parameter depends on the application. On application is the design and animation of the morph between two curves. Fig. 5.4 shows a superposition of a series of frames of such an interpolation. These may also be used as a pattern of curves or as a tool to parametrize the gap.

Our solution extends to morphs defined by more than two curves, for example by using the paradigm proposed in [22] for a Bézier path in the space of polyhedra. But here, instead of using a weighted Minkowski sum, we use a weighted sum of (signed or squared) distance fields and trace the corresponding average curve.

As scaling and addition of (squared) distance fields behave as their equivalent operators on scalars or points, various formulations developed for motions or (parametric or subdivision) curves defined by control points may be trivially adapted to produce motions or patterns of curves that are defined by several control curves. For example, we can use Bézier weights or time parametrized weights derived from the Neville algorithm [82], so as to ensure that the animation interpolates all control curves with proper timing.

CHAPTER 6

DETAILS AND DISCUSSION OF THE SNAP METHOD

The Snap method was shown to yield an average curve for a set of input curves in Ch. 5. How does Snap work? What are properties of the average shape computed by Snap? This chapter presents some answers to such questions.

Snap computes an average for a set of nearly-parallel, closed-loop, planar curves. To be concise, we first establish helpful terminology. A curve without boundary is a closed curve. A smooth, closed-loop, planar curve may be formally described as the image of the circle $\mathbb{S}^1$ under a smooth function $f : \mathbb{S}^1 \to \mathbb{R}^2$. A Jordan curve is a simple, closed loop, planar curve without self-intersections. Formally, for a curve encoded by f to be Jordan, we require f to be an embedding, or, equivalently, we require for $f(\mathbb{S}^1)$ (the image of $\mathbb{S}^1$ under f) to be in a continuously invertible correspondence with $\mathbb{S}^1$. We say that a Jordan curve is homeomorphic to $\mathbb{S}^1$, and that f is a homeomorphism from $\mathbb{S}^1$ to the curve. We denote the set of input curves as $\{C_i\}$, and, a generic input curve of the set as C_i. The face bounded by a Jordan curve C_i is denoted as F_i. Unless qualified, the term curve in this chapter refers to a smooth Jordan curve, and, a set of curves refers to a set of compatible, smooth, Jordan curves.

Snap uses the closest point queries to compute corresponding points on the input curves. However, it is not the case that if a set of points $\{p_i\}$ on the curves $\{C_i\}$ are in correspondence established by our scheme, then p_i is the closest projection of p_j on curve C_i. In this chapter, we first discuss the nature of 'closest projection' correspondences established by Snap, noting how they are similar to the correspondences established by the medial axis. The discussion leads us to consider, in Sec. 6.2, how the medial axis of a shape relates to the average curve for the case of two curves, and to prior art [66] that constructs an aver-

age curve for a pair of Jordan curves. Then in Sec. 6.3, we present our formulation of the average which generalizes the construction of [66] to n smooth Jordan curves.

We then discuss the approximation which Snap uses to compute the average shape in Sec. 6.4 and discuss properties that average curves (and shapes) computed using Snap possess. Finally, leveraging the understanding of Snap gained from the discussion, we present modifications to Snap that improve the configuration of inputs that Snap yields useful results in Sec. 6.5, and, present a practical check for validity of the computed result in Sec. 6.6.

6.1 The asymmetry drawback of closest projection correspondences

Upon termination, Snap yields, for each point on the average, a corresponding point on the input. Snap constructs correspondences between the input curves $\{C_i\}$ that have the property: if point p_i on curve C_i is in correspondence with point p_j of curve C_j, then, there exists a point p and discs $B_i(p, \|p - p_i\|)$ and $B_j(p, \|p - p_j\|)$ such that $B_i \cap C_i = p_i$ and $B_j \cap C_j = p_j$. Snap computes an appropriate p.

Rather than invoking the existential quantifier in the above formulation, one can construct the correspondences first, and then using the correspondences to compute a point on the average. Here is an approach for 'directly' constructing the average curve:

- *Correspondence finding:* Compute corresponding points on the inputs.
- *Point-wise averaging:* Compute a point on the average using the corresponding points.

Finding meaningful correspondences, for some measure of meaningfulness, is a difficult problem, as evidenced by numerous prior works on the subject [83, 84] (see also Ch. 4).

Snap seeks to employ a particularly natural correspondence establishment scheme that uses closest point correspondences (while Snap uses closest projections, it is not the case

that if a set of points $\{p_i\}$ on the curves $\{C_i\}$ are in correspondence then p_i is the closest projection of p_j on curve C_i).

Given two curves C and C_i, the closest projection correspondences are established by associating every point p on C with its point(s) of closest projections on C_i. Formally, the closest projection map $\Pi_i : C \rightarrow C_i$ encodes closest projection correspondences established between each point of C and the closest point(s) on C_i.

Closest projection correspondences are particularly natural for shapes embedded in an Euclidean domain, as these correspondences do not consider parameters other than the Euclidean distance.

In general, Π_i may not be a homeomorphism (Fig. 6.1). However, sufficient conditions for Π_i to be a homeomorphism and thus, for C and C_i to be homeomorphic are defined in [25]. We discuss the condition in Sec. 6.4.1.

Assuming that the closest projection maps from a particular C_i to all other curves $\{C_j\}$ are homeomorphic, then, these maps do yield for each point p_i of C_i, a useful set of corresponding points $\{p_j\} \cup p_i$. Why then can we not specify a point on the average directly as the average of corresponding points $\{p_j\} \cup p_i$?

We cannot as the closest projection map is asymmetric – let p_2 be the closest projection on shape C_2, of a point p_1 that lies on shape C_1. Then, p_2 does not, in general, have p_1 as its closest projection on C_1. Thus, using the correspondences established by the closest projection map, the computed average depends on the particular choice of the 'seed' input curve (in the above description, the curve C_i), and is, thus, not a property of the set of inputs (Fig. 6.2).

Thus, if we wish to use the closest projection map to assist in computing an average curve that is symmetric, we must use an indirect approach. In the next section, we describe how prior art [66] constructs a symmetric center-curve and uses it to establish closest projection correspondences for the case of 2 curves.

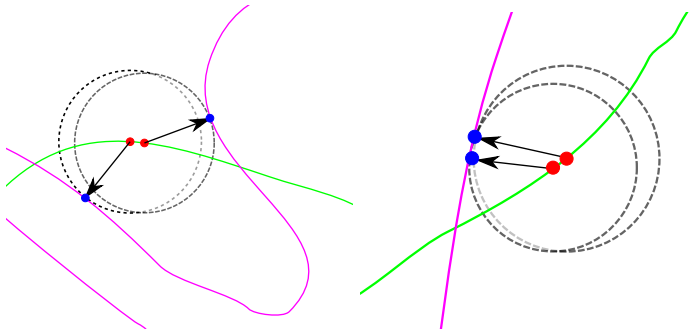

Figure 6.1: Left: Both images show portions of two curves (magenta, green). A configuration where the closest projection maps two red points on the green curve (red) to blue points that are far from each other along the magenta curve. Our presented approach identifies such configurations as invalid. Right: A configuration where the closest projection map is well behaved.

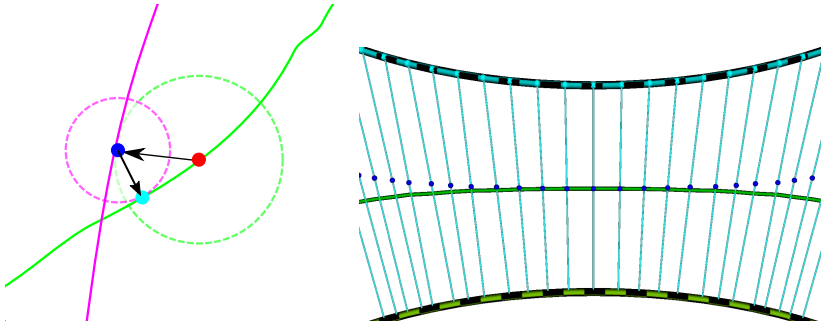

Figure 6.2: When the average is computed using the correspondences established by the closest projection from one of the input curves, chosen as the base, the average and the correspondence map depend on the choice of the base because the closest projection map is asymmetric (Left). In general, if the closest projection of a point p_1 (red) on one curve (C_1, green) onto another curve (C_2, magenta) is p_2 (blue), then, the closest projection of p_2 onto C_1 is not p_1. Right: Samples on one average curve obtained by selecting the cyan input as base are shown as blue dots, along with the cyan edges to their closest projections. The average curve obtained by selecting the green input as base is shown in green.

6.2 Ball-map: symmetric center-curve and closest projection correspondences for 2 curves

For two curves, consider the 'ball-center-curve', points on which are the center of 'maximal-balls'. Maximal balls could be explained as being obtained by, starting with a zero-radius disc centered at a point p_1 on the first curve, and, increasing the radius of the disc while keeping it in tangential contact with p_1 till the disc also makes tangential contact with a point p_2 on the second curve. This construction, the 'Ball-map' is described in [66], and, the center-curve has been used in prior art [67] for correspondence establishment and morphing for a pair of b-compatible input curves.

Given two curves, the ball-center-curve can also be described in terms of the medial axis [58] of a shape constructed from the two curves using set-theoretic operations. As the medial axis is a well-studied object in computational geometry [61], this description is instructive.

Given a Jordan curve C_i, for any point p in the plane, one can compute the closest projections $\{p_i\}$ of p on C_i. The medial axis of the interior of the face F_i bounded by C_i is the set of points in F_i which have more than one closest projection on C_i.

Is the ball-center-curve the medial axis of some shape derived from the inputs? For this, we define the *gap* of a set of Jordan curves $\{C_i\}$ as $\mathbb{G}(\{C_i\}) = (\cup F_i)^- \setminus (\cap F_i)^\circ$, where H^- and H° denote respectively the topological closure and interior[1] of set H and $H \setminus h$ is the set of elements in H that are not in h sets [85] (Fig. 6.4).

The ball-center-curve of a pair of curves C_1, C_2 is the union of the medial axis of the interior of their gap and their common intersection (Fig. 6.4). A morphological formulation is that the center-curve is the intersection of the medial axis of the ϵ-thickening of the gap for all $\epsilon > 0$.

[1]The closure and interior operators are necessary to deal with cases where the gap includes lower-dimensional portions. For example, when all input curves are identical, the gap is one-dimensional and is identical to these curves.

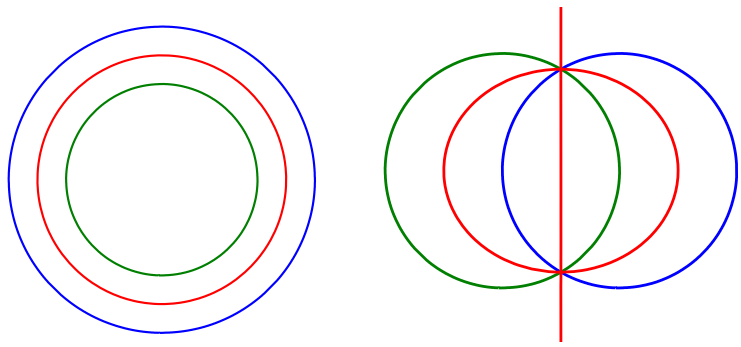

Figure 6.3: Left: Two concentric circles (blue, green) and the set of points at equal distance from each (red). Right: Two non-concentric circles (blue, green) and the set of points equidistant from each (red)

For ball-compatible pair of curves, rather than computing ball-maximal-discs by starting from the first curve, computing them by starting from the second curve leads to the same set of ball-maximal discs. Thus, ball-maximal discs and their centers are symmetric objects. For 2 curves, we propose to use as the average of 2 curves, their ball-center-curve.

6.3 Scalar fields for averaging for n curves

For more than 2 curves, one can seek to extend the above concepts defined for two curves by extracting the center-curve of the gap (recall that the for a set of n Jordan curves $\{C_i\}$, the gap is defined as $\mathbb{G}(\{C_i\}) = (\cup F_i)^- \setminus (\cap F_i)^\circ$, where H^- and H° denote respectively the topological closure and interior). We call this extension the *gap-center-curve*. However, the gap-center-curve is not a good candidate for the average as it does not change when a section of an input curve that is in the interior of the gap is varied while being constrained to lie in the interior of the gap. Thus, similar to the discussion of Sec. 2.2, we seek a useful generalization of the concept of the ball-center-curve to n curves.

6.3.1 Zero-set of summed signed distance function

First, we consider an alternate method to describe the ball-center-curve for 2 curves that is analogous to the number at equal distance. Consider the set of points at equal distance

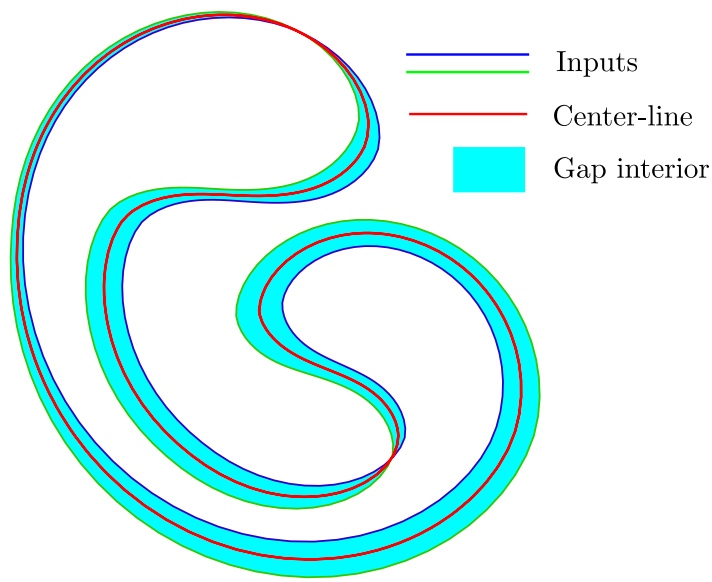

Figure 6.4: A depiction of the terms mentioned in the textual description: two curve inputs (blue, green), the interior of the gap (cyan) and their center-curve (red).

from two curves. For concentric circles, this set is the ball-center-curve. However, two co-incident circles, all the points in the plane meet the criteria of being at equal distance to the inputs, and, for two non-concentric circles, the subset identified is shown in Fig. 6.3.

Despite this, in each case, the ball-center-curve is indeed a subset of the set of points at equal distance. We refine our formulation to 'clip away' the extra points: the ball-center-curve for 2 curves is the set of points in $\mathbb{G}(C_1, C_2)^\circ \cup \Big(C_1 \cup C_2\Big)$, that are at an equal distance from C_1 and C_2.

Now, we describe how using 'signed distances' leads to a simpler formulation. A Jordan curve C_i partitions the plane into an interior ($F_i \setminus C_i$) and an exterior. Thus, given C_i, we can assign to any query point p in the plane, a signed distance $S_i : \mathbb{R}^2 \to \mathbb{R}$, whose evaluation $S_i(p)$ at p is such that $|S_i(p)|$ is equal to the distance between p and C_i, and, $\mathrm{sign}(S_i(p))$ is negative, positive or zero, for points in the interior of F_i, in the complement of F_i, and, on C_i respectively.

Consider the summed signed-distance function $Z = \sum_i S_i$. For a pair of concentric circles, the point-set at equal distance is identical to the set of points where $Z = 0$.

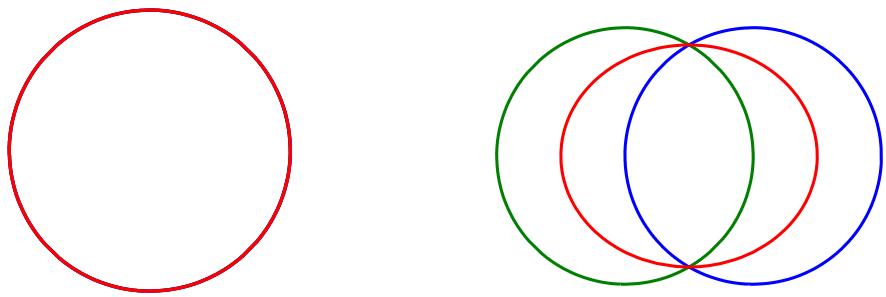

Figure 6.5: Left: For two coincident circles, the set of points that are the zero-set of the summed signed distance function coincides with the input (red). Right: For two non-concentric circles (blue, green), the points identified by $Z = 0$ (red) is a subset of the set of points that are at equal distance from each circle, and, is identical to the center-line.

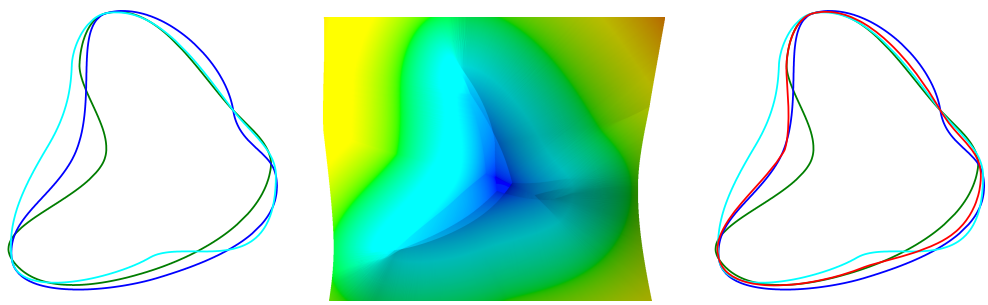

Figure 6.6: Left: Three input curves (blue, green, cyan). Center: The summed, signed-distance height-field Z. Left: The $Z = 0$ level set (red) overlaid over the inputs.

Additionally, for a set of curves, Z is never 0 in the interior of the common bounding region $(\cap F_i)^\circ$ and in the exterior region that is not bounded by any curve $(\cup F_i)^c$. Thus, Z is not in $\mathbb{R}^2 \setminus \Big((\cup F_i)^c$. But, $\mathbb{R}^2 \setminus \Big((\cup F_i)^c \cup (\cap F_i)^\circ\Big) = \mathbb{G}(\{C_i\})^\circ \cup (\cup C_i)$. Thus, for two curves, $Z = 0$ identifies points in the plane that are at equal distance from each curve, and that are contained in the set $\mathbb{G}(\{C_i\})^\circ \cup (\cup C_i)$.

The above is precisely the clipping based characterization of the ball-center-curve and, hence, the ball-center-curve for two curves is the zero-set of the summed signed distance function to them (Fig. 6.5).

The zero-set definition extends directly to n curves. Thus, we have the following possible characterization of the average: the average is the zero-set of the summed signed-distance field (Z). Fig. 6.6 visualizes the field Z and its zero-set for a set of n curves.

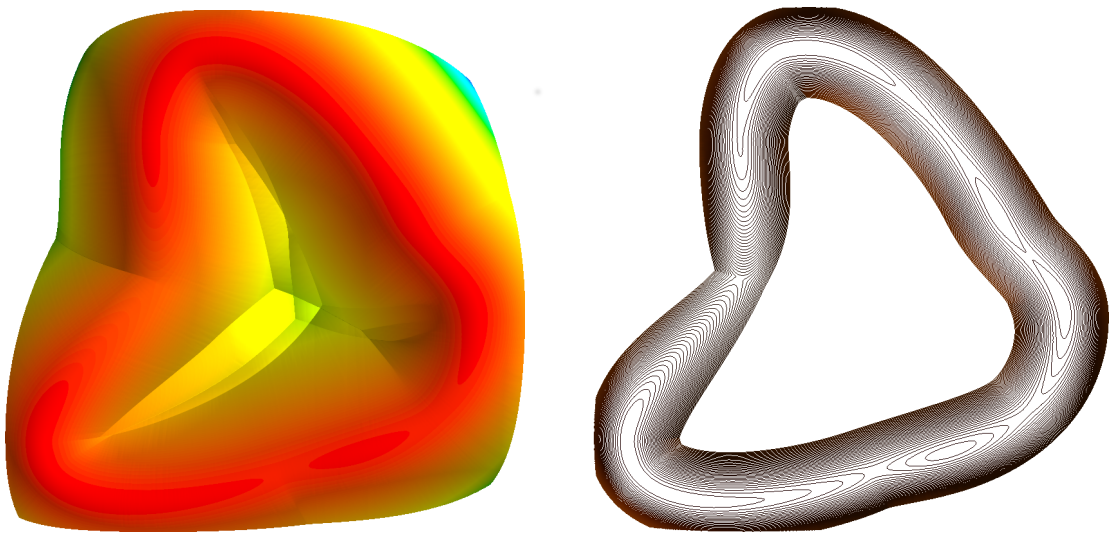

Figure 6.7: Left: The negative of the summed squared distance field Q for the three input curves of Fig. 6.6. Right: Contour lines for finer elevation differences for the top of the height-field $-Q$ are shown.

6.3.2 Valley-set of summed squared distance function

As we have seen (Sec. 2.2), formulations of the average that are equivalent for the case of 2 entities can yield different results for n entities. So, we consider the question: are there other formulations that yield the ball-center-curve? We ask this question particularly because, the zero-set formulation requires the concept of signed distances and thus, the zero-set characterization of the average can not be applied to average curves in 3D. We seek an alternative generalization that allows averaging 3D curves.

As mentioned in Sec. 2.2 (see also Fig. 2.1), considering points where the summed squared distances to the inputs is minimized yields isolated point-sets. However, we need a 1D solution set as our average (i.e., we need an average curve).

We have seen in Ch. 3, an approach to averaging compatible lines that extracts a valley-set of the summed squared distance field Q. Thus, it is reasonable to consider a similar formulation for curves.

Fig. 6.7 depicts the height-field $z = -Q(x, y)$ that at any point (x, y) in the plane evaluates to the negative of the summed squared distance to the input curve-set.

Points on the valleys of the height-field Q, or equivalently, points on the ridge of $-Q$ are given as per [7] by the points where minima of Q (or a maxima of $-Q$) is attained along a principal direction (along a direction given by an eigenvector of the Hessian of Q). This relaxation of seeking a generalized minimum – a minimum along a particular direction identifies a larger point-set.

6.3.3 General approach to averaging using scalar fields

The two approaches that we discussed are both are instances of a general approach to computing symmetric, representative point-sets for a set of shapes.

Constructing symmetric point-sets: Given a set of input shapes embedded in a common ambient space, a process for constructing a point-set that is defined by a set property is to:

1. Construct a 'symmetric' scalar field such that its evaluation at every point in the ambient space does not depend on the ordering of the inputs.
2. Express point-set containment as the satisfaction of a particular property of the scalar field.

The set of points so identified is a property of the scalar field and thus, independent of orderings of the input.

Constructing average point-sets: Points on the average are symmetric and, additionally, they also possess properties that make them a good candidate for representing the inputs, including a) containing points of coincidence of the inputs, b) being parallel to each input, and at an average distance from the inputs for parallel inputs and input sets which are offsets of one another and c) being affected by a variation of each input curve.

The approaches we presented above for computing representative point-sets for a set of lines or planes use different 'symmetric' fields:

- The summed signed-distance field to the inputs: $Z(p) = \sum_i S(p, C_i)$

- The summed squared distance field to the inputs: $Q(p) = \sum_i d^2(p, C_i)$.

The symmetry for each of these fields follows from the commutativity of addition over $\mathbb{R}$ in both cases. The resulting point-set is symmetric. The resulting point-set for either formulation also possess the above mentioned desirable properties amongst others (for the case of two curves, each yields the ball-center-curve) and hence, the results are average point-sets.

6.4 Snapping using tangent lines

To compute points on the zero-set average curve, one could use a spatial grid to approximate the summed signed distance function using techniques such as fast marching and contour extraction techniques to extract the zero-set. Similarly, to compute points which satisfy the differential characterization of the valley we can compute the summed squared distance function using fast marching, and extract the valley-set (see Sec. 4.2.2).

In contrast, the Snap approach of Alg. 1 (Ch. 5) seeks to sample the zero-set of Z or the valley-set of Q by iterative projections onto the zero-set / valley-set of the scalar field constructed using tangent lines to the input curves at the points of closest projection. These local field are denoted in the below discussion by Z_l and Q_l. In Sec. 3.3, the conditions for a point to lie on the zero-set of Z_l or valley-set of Q_l are shown to be satisfied along a line – the zAL or vAL, and, projections onto them can thus be carried out as presented in Sec. 5.2.

Thus, Snap does not require a grid. Rather, it relies on the assumption of compatibility of the input curves to obtain, for a query point p, the set of points $\{p_i\}$ on $\{C_i\}$ at which local tangent approximations need to be constructed for computing Q_l and Z_l and for updating the location of the query point. The updates are performed by a sequence of Move iterations.

Each move iteration uses information about the last 'Moved' location p and the tangent lines $\{l_i\}$ at the points $\{p_i\}$ that are closest to p on $\{C_i\}$. Thus, instead of computing

approximate scalar fields that are required in computing (approximations to) the zero-set of Z and the valley-set for Q, the Snap process computes them exactly for a field that is constructed using local (at a given query point p) affine approximations to the input geometry.

6.4.1 Compatibility conditions for a set of input curves

When are the approximations made by Snap valid? For a set of input curves, let us first consider an incompatible configuration. The *wart* is an example of configurations where, for two curves, the ball-center-curve would have a bifurcation. Fig 6.8 shows how the closest projection map may jump (be discontinuous) when moving along the gap in this case.

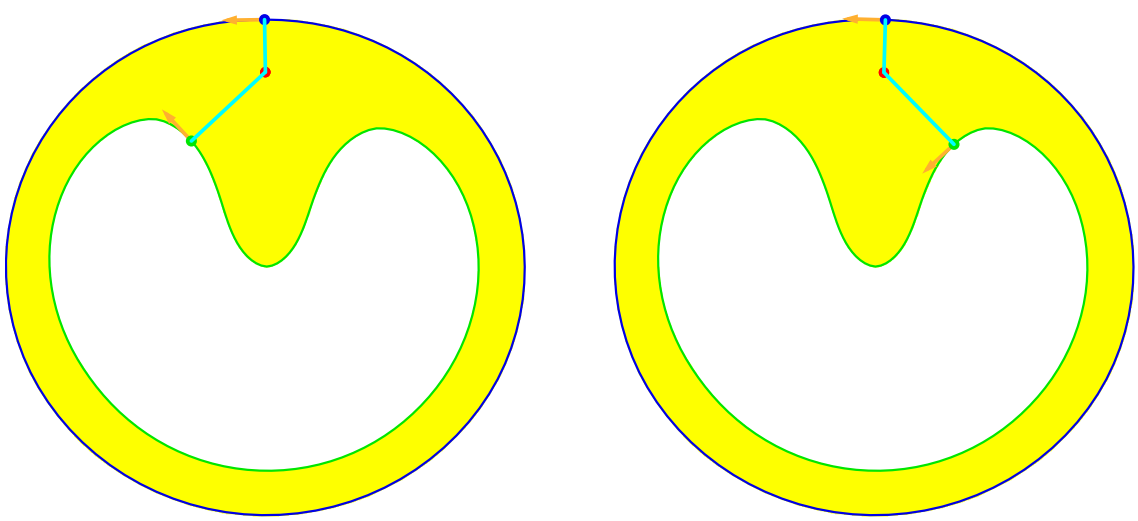

Figure 6.8: Incompatible configurations (wart): the closest projection on the green curve jumps while moving along the gap.

One could construct some form of an "average curve" ignoring such jumps. However, the variability is defined as the square-root of the average squared distance to the curves. As such a computed average curve will not establish homeomorphic closest-projection correspondences, the average will use the same point on at-least one curve at-least twice for

computing the variability field. Thus, the average will not capture a point-wise variability field for the input curves. We declare such configurations of inputs as invalid.

A set of input Jordan curves are compatible if their average is a simple curve that is not contractible in the gap such that the closest projection map from the average curve to every other is monotonic (i.e., each input curve is homeomorphic to the average curve, and, the closest projection map is the explicit homeomorphism).

The above condition for compatibility, however, is existential. Can we derive an a-priori condition that may be evaluated for a particular set of input curves? If we require that, for a set of compatible input curves, the closest projection map from each curve to every other be a homeomorphism, then, any point in the gap has a unique closest projection on the seed curve (for any choice of the seed curve from amongst the input curves).

A sufficient, but, not necessary condition for the closest projection map from curve C_1 to curve C_2 (and vice-versa) to be a homeomorphism is presented in [86]. The condition is phrased in terms of two measures:

- The symmetric Hausdorff distance between the two shapes ($d_H(S_i, S_j)$).
- The minimum of their respective minimal feature sizes, mlfs(S_i, S_j).

The condition is:

$$d_H(S_i, S_j) < (2 - \sqrt{2})\,\text{mlfs}(S_i, S_j) \tag{6.1}$$

For two compact sets X and Y, the directed Hausdorff distance from X to Y is the maximum over the distances from each point $x \in X$ to Y. The symmetric Hausdorff distance is the maximum over the directed Hausdorff distances from X to Y, and, the directed Hausdorff distance from Y to X.

The mlfs [66] of a shape is the infimum of the local feature size (lfs) [87] over every point of the shape. The local feature size at a point on the shape is the distance between the point and the medial axis (Sec. 6.2) of the shape.

The Hausdorff distance between two convex polygons with n vertices each can be computed trivially in $O(n^2)$ and a linear algorithm for computing the symmetric Hausdorff distance between two convex polygons is presented in [88]. [89] presents an $n\log(n)$ solution for computing the Hausdorff distance between two simple polygons. In [90], the authors propose a randomized $O(n^3+\epsilon), \forall\epsilon > 0$ algorithm for computing the Hausdorff distance between two triangle meshes with n triangles. A practical implementation of an $O(n^4 \log n)$ procedure for computing the Hausdorff distance between two triangle meshes with n simplices is presented in [91]. For the more general case of free-form shapes, [92] presents an algorithm for computing the Hausdorff distance between two planar parametric curves that have a rational parameterization and a sampling approach for computing the approximate Hausdorff distance between two surfaces is presented in [93].

The mlfs of a shape may be computed by computing its medial axis and finding the closest point between it and its medial axis. The medial axis may be computed exactly for certain classes of shapes, including polyhedra [94] and natural quadrics (planes, spheres, cylinders and cones: shapes that are the natural result of milling and turning) [95]. [96] presents an algorithm for computing the medial axis of planar free-form shapes. Sampling based approaches for computing an approximate medial axis often compute ([78, 97, 98]) the voronoi diagram of a set of point-samples on the boundary of the input shape and leverage the results that in the limit of sampling density: a) in 2D, the voronoi diagram's vertices converge to the medial axis of the shape b) in 3D, a subset of voronoi diagram's vertices converge to the medial axis [87]. Sampling based approaches typically compute the delaunay triangulation of the set of point-samples from which the voronoi diagram of the point-samples (which is the dual of the delaunay triangulation) can be computed in linear time [99]. This is possible in $O(n \log n)$ (2D [100]) or $O(n^2)$ (3D) [99].

Thus, both quantities (Hausdorff distance and mlfs) appearing in the condition can be computed a-priori for all possible pairing of the input curves to have a guarantee that, the closest projection map between each pair of curves is well behaved.

Additionally, when the sufficient condition holds for every pair of input curves, then, as every point on the average is contained in the gap, the symmetric Hausdorff distance between the average and a particular curve S_i is less than $\max_{j\in\{1,\ldots n\}} d_H(S_i, S_j)$. Also, in light of the properties of the average, we advance the following conjecture: the mlfs between the average and S_i is greater than $\min_{j\in\{1,\ldots n\}} \text{mlfs}(S_i, S_j)$.

Under the mlfs conjecture, using the fact that the Hausdorff distance and the mlfs between each input and the average can be bounded by measurements of these quantities for every pair of inputs, we can obtain an a-priori check for validity of inputs as:

The condition is:

$$\max_{(i,j)\in\{1,\ldots n\}\times\{1,\ldots n\}\wedge i\neq j} d_H(S_i, S_j) < \min(2-\sqrt{2})\,\text{mlfs}(S_i, S_j)$$

Due to the difficulties in implementing non-approximate methods for computing the Hausdorff distance and mlfs, and, as the above a-priori condition is sufficient but not necessary, we propose (Sec. 6.6) computing a result using Snap and then checking its usefulness as the average.

Additionally, in Sec. 7.3, we present NormalSnap, an approach to snapping that, rather than projecting a candidate point p onto the average line, 'moves' it to the average line along the normal n_i to the seed curve C_i at the closest point p_i of p (i.e., updates p as $p = p + sn_i$). NormalSnap fixes the closest projection onto the seed curve as long as the line segment bounded by the points p and $p + sn_i$ does not intersect the medial axis of the region bounded by the seed curve. Thus, if we check for projection compatibility [86]) between each input curve and the seed curve, we obtain a guarantee that: (a) the closest projection map from the resulting average to the seed curve is a homeomorphism and (b) the average is homeomorphic is each input curve.

Finally, we note that, the conditions expressed in the above discussion are well defined for surfaces in 3D as well as curves in 3D, and, the above mentioned conditions are the

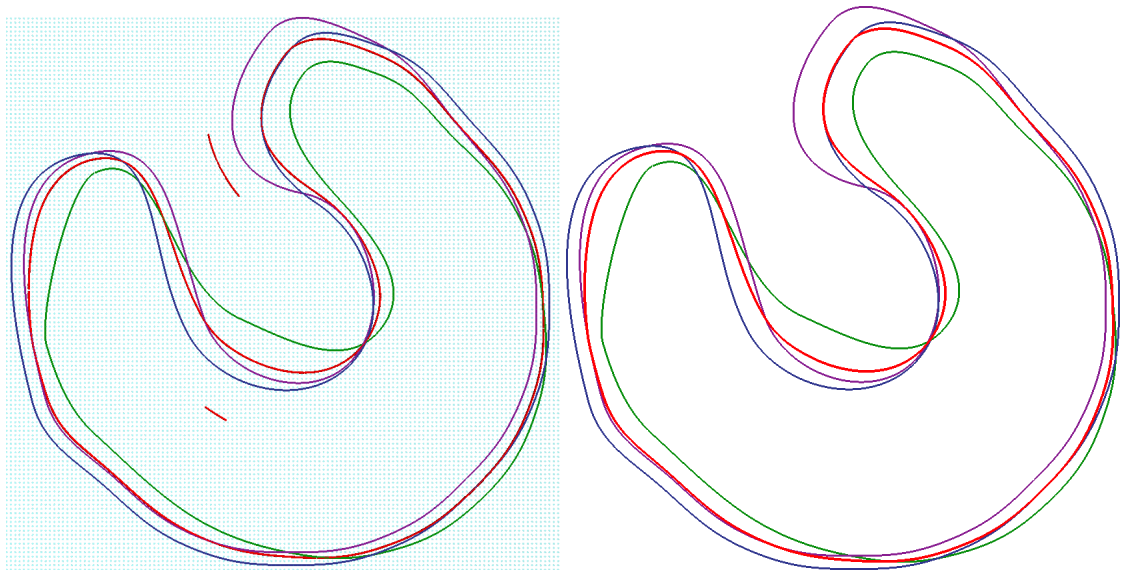

Figure 6.9: Left: Snapping a grid (cyan) of points over the plane yields samples on the valley-set (red) of the summed squared distance function to the set of curves. There are components of the valley that lie outside the gap. Right: The result (red) of snapping samples generated on an input curve.

conditions for a-priori compatibility of curves. Under these conditions on the inputs, the closest-projection map from one curve to another establishes correct correspondences.

6.4.2 Implicit clipping of the valley-set by Snap

For a set of compatible curves, the summed squared distance field is defined over the entire plane. Valleys of this field exist outside the gap too, and, thus, the valley condition is satisfied by points both inside and outside the gap.

Indeed snapping a grid of points scattered over the plane yields points on the valley-set that are outside the gap too (Fig. 6.9, left). Note that the zero-set of the signed-distance function for the configuration of Fig. 6.9) is inside the gap in this case (for any point p outside of the gap that has closest points $\{p_i\}$ and corresponding normals $\{n_i\}$ to $\{C_i\}$, the components $p_i p \cdot n_i$ all have the same sign). Thus, the ensuing discussion is focused on the valley-set of the summed squared distance function.

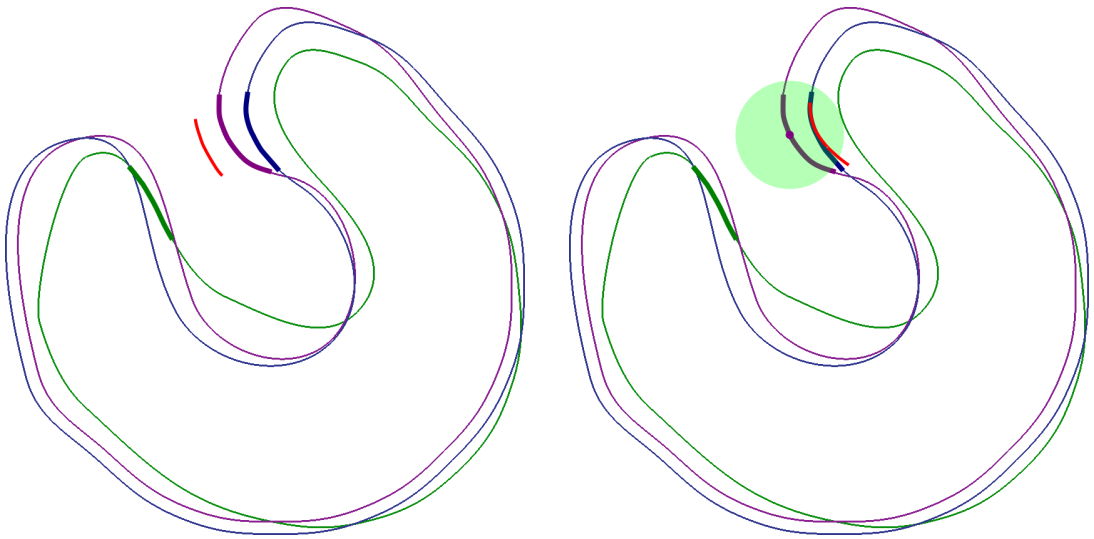

Figure 6.10: Left: The valley-set outside the gap is generated by considering the summed squared distance function induced by highlighted regions of the inputs. Note that this valley-set is generated by 'incorrect' correspondences on the green curve. Right: For a point on the highlighted region of the violet curve, the closest-projection correspondence on the green curve (its intersection point with green ball) are correct due to compatibility of the input curves. That is, the field required to obtain the valley-set outside the gap is not sampled while snapping from a seed curve.

A property of the Snap algorithm is that, for compatible inputs, it returns the sub-set of the valley-set that is of interest as the representative i.e., it performs implicit clipping. Fig. 6.9 (right) shows the result of Snapping from grid-samples that lie in the gap.

How come does Snap compute the correct component of the valley set? This is due to the closest projection step. For a set of compatible curves, the regions of the input curves that induce the field for generating the valley component outside the gap are highlighted in Fig. 6.10 (left, thickened curves). However, as the inputs are compatible, closest projection queries from the input curves, and, from points in the gap do not yield correspondences between these regions of the curve (Fig. 6.10, right). As snapping from an input seed curve does not generate correspondences of Fig. 6.10 (left), Snap doesn't generate the extraneous valley-set.

Table 6.1: For different configuration of two circles with an annular gap, for a set of samples $\{p\}$ on a seed circle, the value $\max(\Big| \|Move^n(p), p_1\| - \|Move^n(p), p_2\| \Big|)$ – the maximum absolute value of the difference in distances between the 'moved' to point $Move^{n-1}(p)$ and its closest points p_1 and p_2 on the two circles at the start of the nth Move iteration.

S. No.	Move 1	Move 2	Move 3	Move 4	Move 5	Move 6
1	6.518	0.01512	2.834×10^{-7}	9.77×10^{-15}	7.994×10^{-15}	7.252×10^{-15}
2	1.846	0.00007233	2.398×10^{-13}	7.105×10^{-15}	7.105×10^{-15}	7.105×10^{-15}
3	7.117	0.05633	0.00001609	1.414×10^{-12}	7.994×10^{-15}	6.217×10^{-15}
4	0.6153	4.866×10^{-6}	9.77×10^{-15}	7.105×10^{-15}	7.105×10^{-15}	7.105×10^{-15}
5	4.981	0.01491	4.844×10^{-7}	5.329×10^{-15}	5.329×10^{-15}	6.217×10^{-15}

6.4.3 Convergence of Snap

For discussing convergence of Snap we note that we have a definition of the global set that we wish to converge to – the zero-set of the summed signed- distance function Z or the valley-set of the summed squared distance function Q.

There are three important questions with regards to convergence using Snap iterations: stability (whether each Move reduces the distance to the average), invariance (whether Snap does not move points already on the average) and attraction (whether samples on a particular seed are attracted to the average).

For the case of a family of lines $\{l_i\}$, a single Move places p on the average line l. A subsequent Move yields the same average line. p is the projection onto it and thus remains fixed.

Next, we discuss first the case of configurations of circles for which analytic expressions for the distance function exist.

Two circle configurations: For two circles, Snap should yield points that are equidistant from both. Also, for the case when the two circles do not intersect and the gap is an annulus, we also know that the medial axis should be an ellipse. We consider a sequence of configurations of two circles. For each configuration, we generate samples on one circle and snap them. We report how each Move iteration of Snap makes progress towards the set of points equidistant from either circle in Table 6.1.

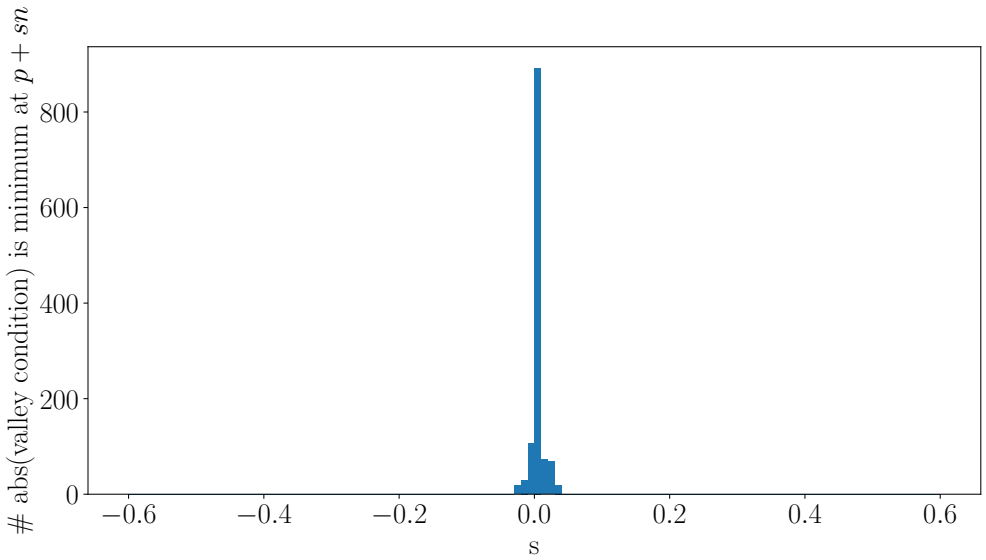

Figure 6.11: For multiple configurations of 2 circles, for each configuration, a set of seed points are snapped. At the snapped location p, the vAL is computed and its normal n is determined. Across the set of input configurations, For each s value, the number of times when the absolute value of the valley condition evaluated at $p + sn$ is the minimum compared to other s values is computed and displayed using a histogram.

We also study the degree of conformance of the Snapped samples to the valley condition of requiring the vanishing of the gradient along an eigenvector of the Hessian for configuration of pairs of circles

For a configuration of two circles, we Snap a set of seed points – we simply iterate Move 3 times. Each snapped point has a valley average line $l = (p, n)$ associated with it. We evaluate the valley condition at locations $p + sn$, for $s \in \{-0.6, 0.59, 0.58, \ldots, 0.6\}$. We evaluate the expressions for the gradient, the Hessian, and a unit larger eigenvector e_1 of the function Q at the above locations and compute the scalar $\nabla Q \cdot e_1$ (which should be zero for the valley).

We report the histogram of s values where the absolute value of the evaluation of the valley-condition is minimum across a number of such circle configurations in Fig. 6.11.

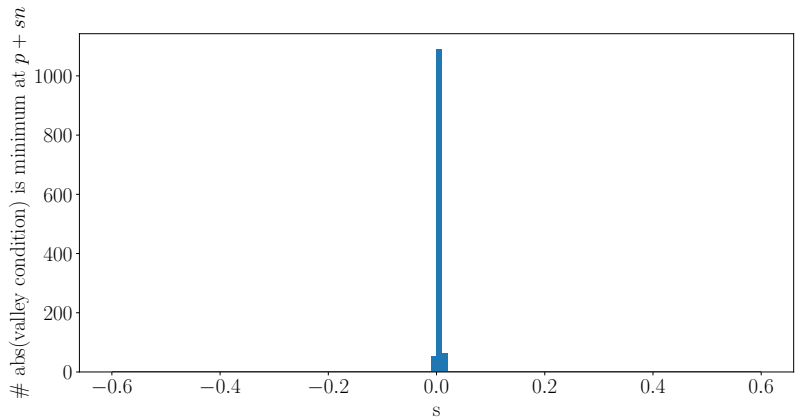

Figure 6.12: For configurations of 3 and 4 circles, for each configuration, a set of seed points are snapped. At the snapped location p, the vAL is computed and its normal n is determined. Across the set of input configurations, for each s value, the number of times when the absolute value of the valley condition evaluated at $p + sn$ is the minimum compared to other s values is computed and displayed using a histogram.

Table 6.2: For different configuration of 3 and 4 circles with an annular gap, the maximum (over all seed samples being Snapped) of the length of the Move vector during the nth Move iteration.

S. No.	Move 1	Move 2	Move 3	Move 4	Move 5
1	2.65582	0.00746656	0.0000252748	1.65739×10^{-7}	1.09267×10^{-9}
2	1.786	0.002749	2.185×10^{-6}	3.164×10^{-9}	6.071×10^{-12}
3	2.706	0.03446	0.0001953	1.271×10^{-6}	8.542×10^{-9}
4	2.842	0.02988	0.00004097	1.624×10^{-7}	6.438×10^{-10}

Many circle configurations: Similar to the case for two circles, we report the conformance of the Snapped samples to the valley condition for configurations of 3 and 4 circles in Fig. 6.12.

Also, for the set of samples under Snap, we report how the maximum distance moved (over the set of samples) changes with successive Move iterations in Table. 6.2.

The case for general curves: For general curves, computing the zero-sets and the valley-sets accurately is difficult as the distance function to a shape does not have a closed form expression. However, for a set of compatible input shapes, each 'Move' iteration of Snap places constraints on the regions of each input shape that may contribute to the construction of the average.

Let us restrict attention to one curve C_i. During a Move, assume that the point under snap is 'moved' from p to q (Fig. 6.13, and, that the closest projection from p on C_i is p_i. Then, C_i does not contain any point in the disc $B(p, r_p)$ where $r_p = \|p, p_i\|$. If a point on C_i influences the calculation of the average line at q, as we assume that C_i is compatible, then, q_i must be in the disc $B(q, r_q)$ where $r_q = \|q, p_i\|$. The region in which the point must lie in is the crescent region formed by the intersection of the two balls in Fig. 6.13.

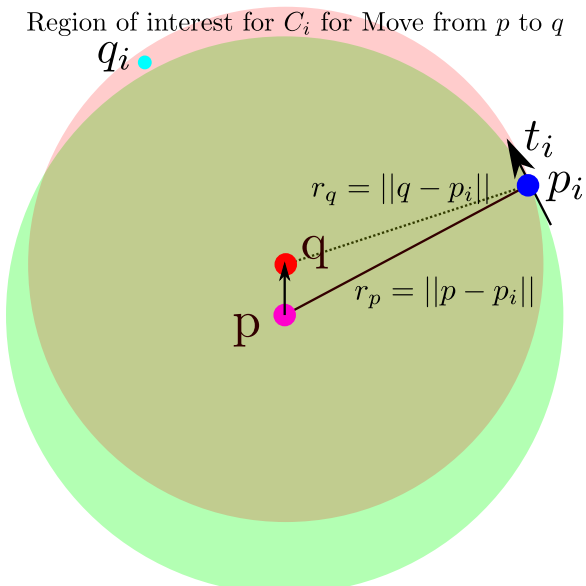

Figure 6.13: During a particular Move, the point under snap is 'moved' from p to q. It is known that the closest projection to curve C_i is the point p_i. Thus, C_i does not contain any point in a region of distance $r_p = \|p - p_i\|$ around p. Additionally, if a new point on C_i has to contribute to the calculation of the average line at q, it must be at a maximum distance $r_q = \|q - p_i\|$ away. Thus, the region in which the point must lie in is the crescent region formed by the intersection of the two balls in this image.

Thus, for a set of compatible input curves, if the length of the Move vector v such that $q = p + v$ is small, then, the crescent region in which regions of the input curve that may contribute to the computation of the average at p_{new} lie is small in volume.

As a result, we propose measuring the distance moved at every Move iteration and using it to decide when to stop iterating Snap. Aside from the results on smooth circles in Table 6.2, we present experimental results on piecewise linear representations using this methodology (Table 7.2).

6.4.4 Symmetry properties of the average

A different question from convergence is: is the result of snap independent of the selection of the seed shape? For piecewise linear shape representations, this statement is false – in fact, these representations are not compatible as they have a local feature size of 0 at vertices.

Our Snap algorithm does not preclude such an independence as it composes constructions that are symmetric (independent of the order of inputs). For the case of curves, in each Move step, we project onto a line that is extracted using a 'symmetric' scalar field (Z or Q).

For the case of two curves, Snap yields samples that are almost at equal distance from either curve (Table 6.1) – i.e., samples that are on their ball-center-curve. A sufficient condition on the inputs that will ensure that the ball-center-curve is without bifurcations is presented in [66].

The condition is:

$$d_H(S_i, S) < \text{mlfs}(S_i, S) \tag{6.2}$$

Thus, Snap yields a symmetric average if the above condition holds on the inputs for the case of 2 curves.

For the case of n curves, while we do not have a condition on the inputs which, if satisfied, will yield a result that is symmetric, we have seen some experimental evidence (Fig. 6.12) that Snap's result approaches the valley-set of Q which is a 'symmetric' scalar fields and thus whose zero-set and valley-set are independent of the ordering of the inputs.

In a similar vein, in Ch. 7, we present our experiments on a family of triangle meshes where we find that Snap called on different seed surfaces yields sets that (although clearly different, since they are produced by snapping triangulations with different sampling and connectivity) are close in the Hausdorff sense (See Ch. 7, Table 7.1).

6.4.5 Incremental averaging

Another question is: can the average be computed incrementally? Given a list of n curves $\{C_i\}$, let C denote their average. Let the average of the first $n-1$ curves of the list be denoted by C^{n-1}. The incremental averaging question is: can we compute the average curve C using C^{n-1} and C_n.

We do not foresee a relatively simple extension to Snap that enables incremental computation of the average. Indeed, one recipe for computing the average curve incrementally is: compute as the 'incremental' average, the medial axis of the gap for C_n and C^{n-1}. For a list of concentric circles of increasing radii r_1, r_2, r_3, this approach will yield a circle which is concentric with the inputs and has radius $\frac{\frac{r_1+r_2}{2}+r_3}{2}$. This approach does not yield a symmetric result as the radius is not a symmetric expression in r_1, r_2 and r_3. Indeed, first averaging the circles corresponding to r_2 and r_3 and then incrementally averaging the result and the circle with radius r_1 yields a circle with radius $\frac{\frac{r_2+r_3}{2}+r1}{2}$. Thus, this approach and does not yield C.

It is also possible to compute a weighted average of C^{n-1} and C_n (see Sec. 5.5). Such an approach too may need not necessarily yield C.

6.5 Beyond compatible configurations

In this section, we leverage our understanding of the action of Snap to propose modifications that increase the set of configurations for which Snap yields a useful result on.

6.5.1 Gap relative projection (GRP)

Defining p_i as the closest point on C_i to p is adequate for compatible configurations. However, Fig. 6.14 shows a configuration where both vAC and zAC yield inadequate results.

To understand why a segment of the left-most portion of the average in the gap is missing in the vAC and lacking the expected curvature in the zAC, consider a point p on

that missing segment. Its closest projection p_1 onto the outer-most (green) curve C_1 is not on the left-most part of C_1, but on a different portion of C_1.

One may wish to use a curve's orientation for selecting the corresponding point p_i on it. Thus, we may correspond p to the point p_i where $p_i p \cdot t_i = 0$ and $p_i p \cdot n_i > 0$, selecting the first point where the normal is facing outwards.

However, in the input configuration of Fig. 6.14 even this check too is not sufficient. Rather, we define p_i as the gap relative projection (GRP) of p.

GRP: The *Gap Relative Projection* (GRP) defines p_i as a point of C_i such that the line segment (p_i, p) lies in the gap and is orthogonal to C_i at p_i.

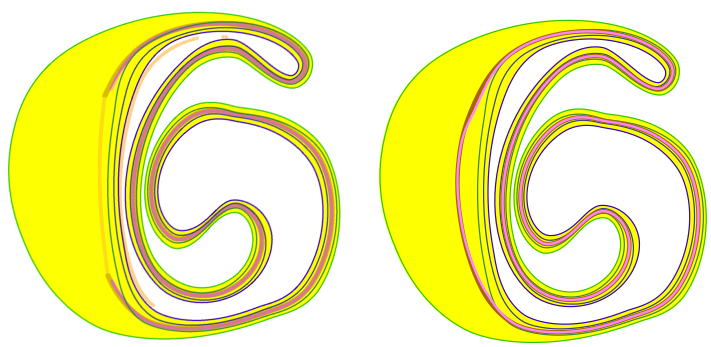

Figure 6.14: Configuration of four input curves, three of which are similar. Top: the valley of Q contains components inside (pink) and outside (orange) of the gap and is discontinuous (misses its left-most section) inside the gap. Overlaid over this is the zero set of D (thick brown) that lies within the gap, but contains an erroneous region (orange), which results from using projections onto segments of the left-most green input curve that are not visible form within the gap. Bottom: Correct result (pink vAC over thick, brown zAC) obtained by using the modified (GRP) definition of p_i.

6.5.2 Tracing: implicit GRP

Trace (Sec. 5.3) uses local connectivity of each input curve, and selects locally closest projections. Thus, Trace implicitly performs a Gap Relative Projection, given that it has been initialized with a good starting seed point. Fig. 6.15 shows an example configuration where Trace correctly selects the relevant correspondences even in cases when Snap fails

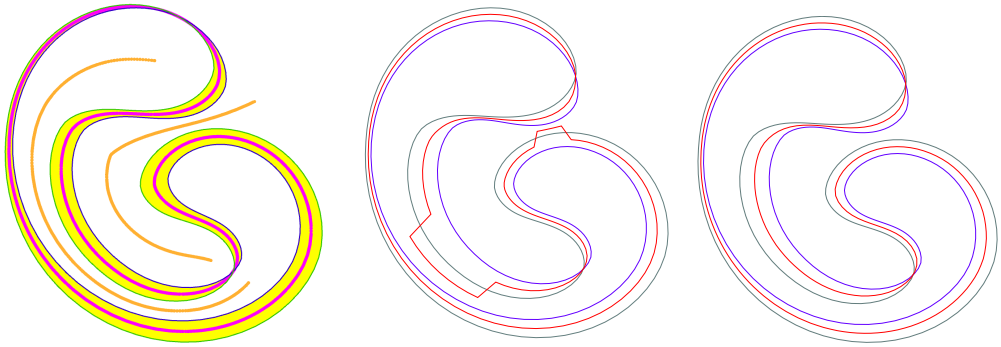

Figure 6.15: Left: The vAC of two curves (green and blue) has 3 components: one in the gap (pink) and two outside (orange). Center: Snap select an incorrect valley in some regions. Right: Trace selects the correct valley.

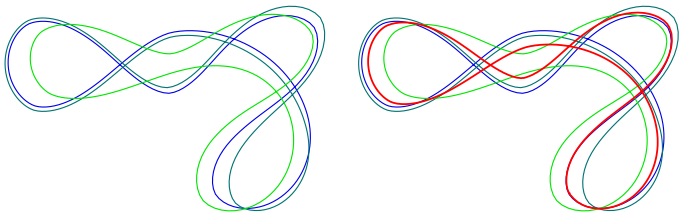

Figure 6.16: Self-crossing input set (left) and the vAC representative (right, red).

In fact, since Trace performs a local analysis which uses the connectivity and orientation of the input curves, it also yields a useful average for configurations that are locally compatible, but where the curves may self-cross once or more (Fig. 6.16).

However, note that Trace is not guaranteed to produce a useful average result for self-intersecting curves. The *GRP confusion* (Fig. 6.17) is an example of configurations where the gap is not an annulus. Because our *Trace* algorithm works on a local section of the gap at a time, it may create a valid average curve for some of such configurations, but, may also fail to start properly or be confused when these overlaps are local.

6.5.3 Constructing a set of compatible curves from the input-set

Gap Relative Projection and Trace increase the set of input configurations for which Snap yields a valid average by selecting the correct local minimum point of the distance function for establishing correspondences to a query point under Snap. Given a set of input curves

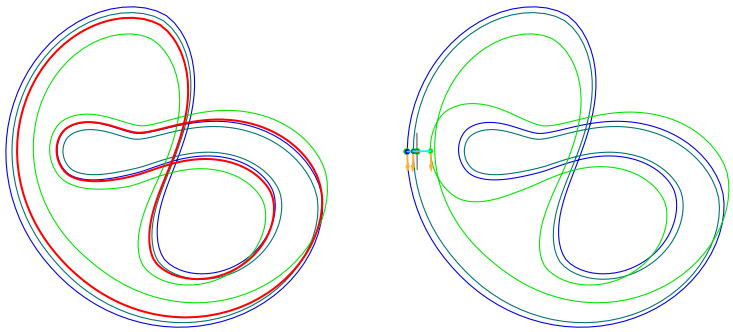

Figure 6.17: GRP confusion: Left: A set of inputs for which we can Trace the average curve correctly. Right: Our particular choice of the seed for computing the first sample on the average (our implementation of Trace picks the left-most vertex of the input curves) leads to a wrong gap-relative projection onto the green curve (right).

$\{C_i\}$ that are incompatible, another possibility is to: (a) compute a set of compatible curves $\{C'_i\}$, (b) visualize or evaluate a quantitative measure of discrepancy between $\{C_i\}$ and $\{C'_i\}$, and (c) if the compatible set $\{C'_i\}$ is deemed a good approximation of the inputs $\{C'_i\}$, compute the average of $\{C'_i\}$ and an approximation of the average of $\{C_i\}$.

In 'Relative Blending' [101], the authors propose an approach to smooth the sharp features of a shape A by specifying the smoothing indirectly through a smooth control shape B. The smoothened resulting shape $R_B(A)$ is obtained by rolling a variable radius ball on the boundary of B in a manner such that the boundary of $R_B(A)$ is almost ball-compatible [66] with the boundary of B.

Along similar lines, given two curves C_1 and C_2, we can compute $C'_1 = N_{C_2}(C1)$ such that the closest-projection map from C_2 to C_1 is almost a homeomorphism (the word almost is used here in a similar vein as [101] – i.e., C'_1 is a limit curve of a family of curves for which the closest-projection map from C_2 is a homeomorphism). C'_1 is constructed by growing balls centered at each point of C_2 till they intersect C_1. C'_1 is given by the boundary of the union of these balls.

To obtain a set of compatible curves $\{C'_i\}$ from $\{C_i\}$, we first initialize the set with $\{C_i\}$. Then for each pair of curves $C'_j, C'_k \in \{C'_i\}$ we repeatedly and simultaneously replace C'_j by $N_{C'_k}(C'_j)$ and C'_k by $N_{C'_j}(C'_k)$.

6.6 Checking validity of the average

While there are a-priori conditions which, if satisfied, yield a valid average, these conditions are not necessary conditions. Thus, practically, we propose checking the validity of the computed average instead of checking the compatibility of the inputs.

We propose here three non-equal criteria that, depending on the application, may be required of a valid average. In some use-cases, it may be enough to require that the average establishes pairwise homeomorphisms between the set of inputs. One may, on the other hand, require the average to be homeomorphic through the closest point correspondences established from the average to each of the input surfaces. Finally, one may also require the average to be independent of the choice of the seed shape.

For the purpose of our thesis, we declare that the average is valid if the closest-projection map from it to each input is a homeomorphism, and, vice-versa.

Thus, the check for curves is: the closest projection map from the computed representative to each input curve should lead to moving 'forward' on C_i. Similarly, the closest projection map from C_i to the computed curve should result in moving 'forward' on the average. One can sample the inputs densely, and perform these checks on a sampling.

Incremental validity check: The above proposed validity check can be performed incrementally for Trace (Alg. 5). Assume that u is the last sample appended to the list of samples, and that v is the new vertex produced by *Snap*. To perform a local compatibility check, for each curve C_i, we verify that the closest projections from the line segment (u, v) to the section of C_i between the closest projection u_i of u and the closest projection v_i of v is a homeomorphism and vice-versa.

CHAPTER 7

AVERAGING A SET OF COMPATIBLE SURFACES

This chapter discusses the use of the Snap algorithm to compute a representative surface for a set of surfaces. As surfaces in 3D, similar to curves in 2D are manifolds of dimension $n-1$ embedded in an n dimensional space, two variants of the Snap algorithm are presented. For a set of compatible input surfaces $\{B_i\}$, we snap samples to the average surface using a) the zero-set formulation or b) the valley-set formulation.

The chapter is organized as follows: first, we present the Snap method for snapping samples of a seed surface onto the average surface. Subsequently, we present details of the various sub-methods invoked by Snap. Following this, we propose a version of the Snap algorithm that directly yields correspondences for the seed surface's samples in Sec. 7.3. We conclude by presenting experimental results obtained using Snap.

7.1 Snapping for surfaces

The Snap method, specialized for surfaces, is listed in Alg. 6.

Algorithm 6 Snap method for surfaces

```
procedure SNAP(p*_k, 𝔽 = {B_i})
    p ← p*_k
    while not converged do                                  ▷ Move iteration
        for i ∈ {1, ... |𝔽|} do
            p_i ← ClosestProjection(p, B_i)
            n_i ← OutwardSurfaceNormal(p_i, B_i)
        n ← AverageNormal({n_i})
        v ← DisplacementVector(p, n, {p_i}, {n_i})
        p ← p + v                                           ▷ Move
    return p
```

Snap is provided with a sample of the seed surface p_k^*. p is initialized to p_k^*. Then, Snap iterates the 'Move' step. Each Move computes a Move-vector v and moves p by v.

For computing v, the normals $\{n_i\}$ to the input surfaces at the closest projections $\{p_i\}$ of p are computed and, v is computed so that $p + v$ lies on the average plane to the set of planes $P_i = (p_i, n_i)$.

Note how, in comparison to Alg. 1 that was presented for curves, the 'Project' step that computes the closest projection onto the average has been replaced by a 'Move' step that computes a projection (not necessarily closest) onto the average tangent plane of $\{B_i\}$ at $\{p_i\}$.

To fully specify Snap, we only need to discuss how to compute v. In the next section, we discuss the extension of the idea presented for planar curves: at each Move iteration, compute a v that is directed along the average surface normal n. Subsequently, we present a version of Snap which makes a different choice of the move vector v and discuss how it may yield more useful correspondences.

Materials discussed in the following preceding sections are useful for the following discussion:

- Techniques for computing an average vector for a set of unit vectors (Sec. 3.1).
- The zAP and vAP formulations for computing the average plane (Sec. 3.4).

7.2 SymmetricSnap: Snapping by moving parallel to the local average normal

The first variant of the Snap algorithm for surfaces, makes a symmetric decision in computing the direction of the move vector v. Thus, we term it SymmetricSnap.

Given a point p, SymmetricSnap computes the estimated average plane and projects p onto it. The average plane is computed in the following manner: ClosestProjection(p, B_i) computes the closest projection p_i of p on B_i. OutwardSurfaceNormal(p_i, B_i) computes the outward facing normal n_i of B_i at p_i. This gives us a set of planes $\{P_i\} = \{(p_i, n_i)\}$ for averaging.

For SymmetricSnap, AverageNormal$(\{n_i\})$ computes the unit normal to the average plane of $\{P_i\}$.

For zAP, AverageNormal returns

$$n = Normalize(\sum_i n_i) \tag{7.1}$$

For vAP, AverageNormal returns

$$n = e_1 \tag{7.2}$$

where e_1 is the unit eigenvector corresponding to the largest eigenvalue of the Hessian of the summed squared distance to the planes $\{P_i\} = \{(p_i, n_i)\}$ (see Sec 3.4).

Then, SymmetricSnap computes (DisplacementVector) the vector v that will be used to update p. For SymmetricSnap, v is parallel to n.

Similar to the case for lines, while one could compute a point-plus-normal representation of the average plane P, this computation becomes numerically unstable when the input planes become parallel. Hence, we compute instead the distance s along n that brings p to point $p + sn$ on P. For both vAP and zAP, we derive an expression for s using a constraint that all points on P must satisfy and that may be written in terms of the summed squared distance field Q or the signed distance field Z.

The zAP displacement vector may be computed as:

$$v = sN \text{ with } s = (\sum_i \frac{(p_i - p) \cdot n_i}{N \cdot N}) \tag{7.3}$$

where $N = \sum_i n_i$ (note that the computation of the zAP displacement vector does not require $N = \sum_i n_i$ to be normalized).

The above condition is derived from the constraint that $Z(p + sn) = 0$, which implies

$$\sum_i (p_i - p - s\frac{\sum_i n_i}{||\sum_i n_i||}) \cdot n_i = 0 \tag{7.4}$$

The vAP displacement vector is derived from the constraint $\nabla Q(p + sn) \cdot n = 0$. Sec. 3.4 lists expressions for Q.

$$v = sn \text{ with } s = \frac{-(Ap + b) \cdot n}{(An) \cdot n} n \tag{7.5}$$

Finally, Move updates the current position: $p = p + v$.

Theoretically Snap terminates (Boolean converged becomes true) when $Snap(p) = p$. In practice, we stop when the distance between $Snap(p)$ and p falls below a threshold. We discuss the validity of this heuristic in Section 7.5.

When SymmetricSnap terminates, the closest projection p_i of point p onto B_i yields the established correspondences. p corresponds to each p_i, and, every pair of points in the set $\{p_i\}$ are in correspondence with each other. The closest projection correspondence map is denoted by $\Pi_i : B \to B_i : \Pi_i(p) = p_i$.

A valid Snap surface induces homeomorphisms between each ordered pair of input surfaces, given by $P_{i \to j} = \Pi_j \circ \Pi_i^{-1}$ ($\circ$ denotes composition). Thus, theoretically, upon termination of SymmetricSnap, attributes on B_k defined by $f_k : B_k \to \mathbb{R}^n$ can be pulled back onto any B_j as: $f_j : B_j \to \mathbb{R}^n = f_k \circ P_{j \to k}$.

7.3 NormalSnap: Snapping by moving along a seed surface's normal

Aside from the selection of seed surface for initialization, SymmetricSnap's constructions are independent of the order of $\{B_i\}$. An unfortunate side-effect of this symmetry however, is that, in general, the closest projection correspondence for the point $p = Snap(p_k^*)$, where point p_k^* is a point on the seed surface B_k is not equal to p_k^*

Thus, if we desire to find the corresponding points $\{p_i\}$ on $\{B_i\}$ for a query point p_k on B_k, we need to first find the point p_k^* on B_k such that $\Pi_k(Snap(p_k^*)) = p_k$. Then p_k will be

contained in the set of correspondences computed for $p = Snap(p_k^*)$. However, computing p_k^* may require several guess-update iterations.

So, we propose a modified version of Snap – 'NormalSnap' that enables efficiently answering queries of the form: what are the corresponding points on $\{B_i\}$ to a point p_k on the surface B_k. The key idea of NormalSnap is to fix the closest projection of p onto B_k during Move to the seed sample p_k^*. This is ensured by computing the displacement vector v as being parallel to the normal n_k to the seed surface B_k at the point being under Snap p_k^*.

NormalSnap differs from SymmetricSnap only in the computation of DisplacementVector:

$$v = \frac{s}{n \cdot n_k} n_k$$

where s is the value computed during SymmetricSnap (see Equations 7.5 and 7.3).

Let $p = Snap(p_k^*)$. The total distance moved during Snap is $\|p_k^* - p\|$. As the direction of each move is along n_k, upon termination of NormalSnap, we can represent the average surface B implicitly as a variable distance offset surface of B_k. Given a point p_k^* on B_k, we obtain a point p on B as $p = p_k^* + \|p_k^* - p\| \, n_k$

The idea of constraining the Move direction is equally applicable for snapping to the average curve.

7.4 Implementation details

Snap can be implemented for surface representations that support closest point and normal queries. In our implementation and reported results, the input surfaces B_i's are high resolution triangle meshes approximating smooth surfaces.

Closest point queries to each B_i are accelerated using an axis aligned bounding box tree. As a closest point query does not generally yield a unique result for triangle meshes, we use a random closest point from the set of closest points as the query result. A possible

alternative would be to use the centroid of the set of closest points as the result of the closest projection result, but, we have not experimented with such a choice and rely on the use of highly refined triangle meshes.

We take as input a seed triangle mesh T_k^* that is either equal to, or, is an acceptable approximation of B_k. We snap each vertex of T_k^* to obtain the vertex set of the average mesh T. T has the same connectivity as T_k^*.

An alternative approach to accelerating computation is to use the GPU. One can: a) accelerate the evaluation of the distance field at sample locations, and, b) accelerate nearest point on surface queries.

Distance field evaluation: [102] (Ch. 34) presents an approach to sampling the signed distance field to an input triangle mesh upto a maximum distance d away from the mesh using the GPU. A uniform grid is constructed with the user desired resolution. For each triangle of the mesh, an oriented bounding box is constructed and tetrahedralized. For each z-slice of the uniform grid, the set of constructed tetrahedra are intersected with the z-slice and the intersection cross section is rendered using a GPU fragment program that computes the signed distance to the input triangle mesh for pixel in the rasterization of the intersection cross section (see also [103] for details of the fragment program).

Accelerating closest point queries: [104] presents an approach for approximately computing the closest point to a trimmed NURBS surface. For each patch of the NURBS surface, a set of axis aligned bounding boxes $\{b_i\}$ are constructed that contain the patch. For a query point p, the minimum and maximum distances to $\{b_i\}$ are first computed (in parallel for all the bounding boxes) on the GPU using a fragment program. The bounding box b_k with the smallest minimum distance to p is then computed using a parallel reduction on the GPU in $\log n$ passes where n is the number of bounding boxes. The maximum distance from p to b_k is used to parallely compute the subset B of $\{b_i\}$ that may potentially contain a region of the NURBS patch that is at a closer distance to p than the patch region

that is contained in b_k. The search for the closest point is subsequently restricted to the trimmed set of bounding boxes.

7.4.1 Tracing for surfaces

We now discuss an approach to extend Trace(Sec. 5.3) to surfaces. The Trace process accepts as input a set of surfaces and produces a set of samples on the average along with a triangulation of the samples. Trace for surfaces uses some ideas described in SwingWrapper [105], which proposes a procedure for re-tiling ([106]) a triangle mesh. Ideas discussed in SwingWrapper relevant to the Trace approach proposed here are: (a) the process of incrementally creating the re-tiling by adding at each step, a single triangle, and, potentially, a new vertex and (b) constructing a triangle that is incident on an existing border vertex of the re-tiling constructed thus far if the potential new vertex location is near enough to a border vertex.

We first sample three points on a seed surface using the approach used for constructing an initial triangle in SwingWrapper ([105], Fig. 3). The points are snapped to get a triangle whose vertices lie on the average. Post this initialization process, Trace incrementally adds a triangle to the average triangulation at each step. For each boundary edge e of the average triangulation constructed thus far, a new vertex p is predicted using a parallelogram predictor. A sample on the average is obtained as $Snap(p)$. Similar to SwingWrapper, if the snapped location is close to a boundary vertex v of the average triangulation, the triangle incident on v and containing e is added to the triangulation. Else, the average triangulation is updated by adding to it the triangle incident on $Snap(p)$ and containing e.

While Trace is sequential, post initialization, it generates candidate samples for Snap, rather than requiring a set of seed samples. This property is useful when the inputs are surfaces with boundary, such as a collection of registered range scans [107]. For surfaces with boundary, snapping samples on one particular seed may not suffice to well sample all regions of space where the average exists. Thus, we anticipate that, with appropriate

Table 7.1: The variation, and its rate of change, between evolving surfaces resulting from applying Snap to different seed meshes in a family. For a set of input triangle mesh families, the second column reports the maximum of the mean Hausdorff distance (sampled) over all pairs of meshes in each family. Subsequent columns report the maximum mean Hausdorff distance over all resulting triangle-meshes for a particular Move iteration, for increasing Move iteration count, first for SymmetricSnap's Move iterations, and, then, for NormalSnaps' Move iterations. Each Hausdorff distance is normalized by the diagonal of the bounding box of the pair of meshes under consideration.

Averaging Method	**Mesh Family $\mathbb{F}$**	**Normalized Max Mean Hausdorff (MMH) $\mathbb{F}$**	**MMH Symmetric Moves**				**MMH Normal Moves**			
			Iter 1	Iter 2	Iter 3	Iter 4	Iter 1	Iter 2	Iter 3	Iter 4
vAS	Implicit 1	0.01397	0.00079	0.00079	0.00079	0.00079	0.00081	0.00078	0.00078	0.00078
	Implicit 2	0.02622	0.00043	0.00029	0.00029	0.00029	0.00053	0.00035	0.00035	0.00035
	Concentric Spheres	0.86518	0.00059	0.00025	0.00025	0.00025	0.00062	0.00037	0.00037	0.00037
zAS	Implicit 1	0.01397	0.00078	0.00078	0.00078	0.00078	0.00078	0.00078	0.00078	0.00078
	Implicit2	0.02622	0.00042	0.00029	0.00028	0.00028	0.00042	0.00029	0.00028	0.00028
	Concentric Spheres	0.86518	0.00062	0.00024	0.00024	0.00024	0.00062	0.00024	0.00024	0.00024

Table 7.2: The Snap procedure is performed for each vertex of a triangle mesh in the family, and the average and max distances moved by the image of the vertex under successive Move iterations is noted. This is done with each triangle mesh in the family serving as the seed, and, the summary statistics reported above are the maximum of the average and mean distances moved. The distances are normalized with the average max mean Hausdorff distance reported for the family as reported in Table 7.1.

Averaging Method	**Mesh Family $\mathbb{F}$**	**Symmetric Move distances**								**Normal Move distances**							
		Iter 1		Iter 2		Iter 3		Iter 4		Iter 1		Iter 2		Iter 3		Iter 4	
		Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max
vAS	Implicit 1	1.4865	4.6409	0.0004	0.6796	0.0000	0.0272	0.0000	0.0272	1.4868	4.6409	0.0006	0.8325	0.0001	0.0257	0.0000	0.0187
	Implicit 2	4.7306	11.2438	0.0637	2.4293	0.0104	1.6599	0.0063	1.7926	4.7749	12.4449	0.0964	18.8834	0.0126	10.5771	0.0086	5.7605
	Concentric Spheres	1.7330	1.7343	0.0054	0.0079	0.0001	0.0029	0.0001	0.0022	1.7333	1.7370	0.0055	0.0079	0.0005	0.0029	0.0001	0.0022
zAS	Implicit 1	1.4865	4.6408	0.0003	0.2457	0.0000	0.0272	0.0000	0.0271	1.4865	4.6408	0.0003	0.2457	0.0000	0.0272	0.0000	0.0271
	Implicit 2	4.7216	18.7917	0.0596	11.3477	0.0096	5.4460	0.0050	1.3284	4.7216	18.7917	0.0596	11.3477	0.0096	5.4460	0.0050	1.3284
	Concentric Spheres	1.7327	1.7337	0.0058	0.0079	0.0001	0.0020	0.0000	0.0008	1.7327	1.7337	0.0058	0.0079	0.0001	0.0020	0.0000	0.0008

modifications, Trace can also be extended to surfaces with borders. Some modifications required are presented in Ch. 9, where we discuss a variant of Snap for open curves.

7.5 Experimental results

Snap works for a varied set of input configurations, as illustrated in Fig. 7.2. Note that the Hausdorff distance between the torii is large compared to their overall size, showing that the key requirement for compatibility is that the surfaces be roughly parallel and not spatial proximity. Snap also displays good convergence properties (Table 7.2).

The quantitative results presented in the remainder of this section are also described in our publication [108].

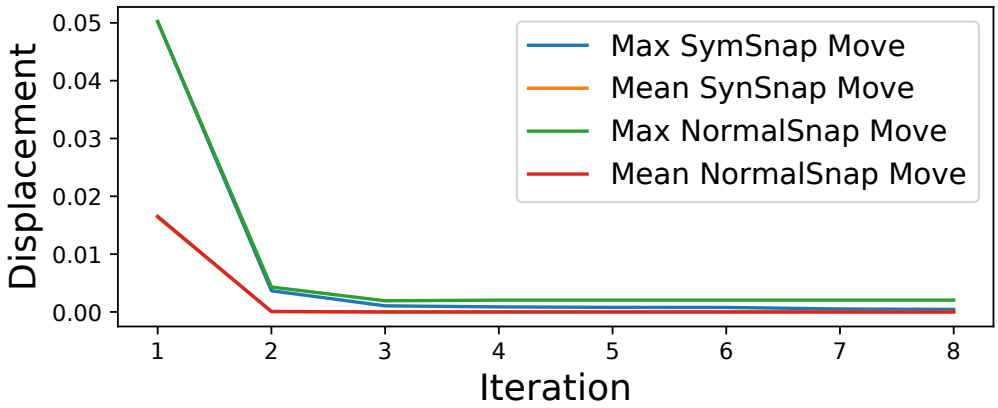

Figure 7.1: The max and mean move displacements for each Move iteration of vAS SymetricSnap and NormalSnap are plotted for a set of shape families that each consist of subdivision surfaces.

7.5.1 Validity checking

Similar to the case for checking the validity of the average for curve (Sec. 6.6), rather than the more difficult task of establishing necessary and sufficient a-priori conditions that will guarantee that the computed average is useful, we propose a compute and decide paradigm: computing the snap set – the result of snapping points on a seed, and, then answer questions about its validity as the average. We propose the following check for usefulness of the average:

- Approximate B_k by a triangle mesh T_k^*.
- Compute the vertex set $\{v\} = Snap(T_k^*)$, and the Snap correspondence sets $\{v_k\} = \Pi_k(\{v\})$, where Π_k returns any closest projection in the case of multiple such projections.
- Obtain the average triangle mesh T as the triangle mesh that has $\{v\}$ as its vertex set, and that has the same connectivity as T_k^*.
- Obtain the corresponding triangle mesh set $\{T_j\}$, where T_j has $\{v_j\}$ as its vertex set, and inherits the connectivity of T.

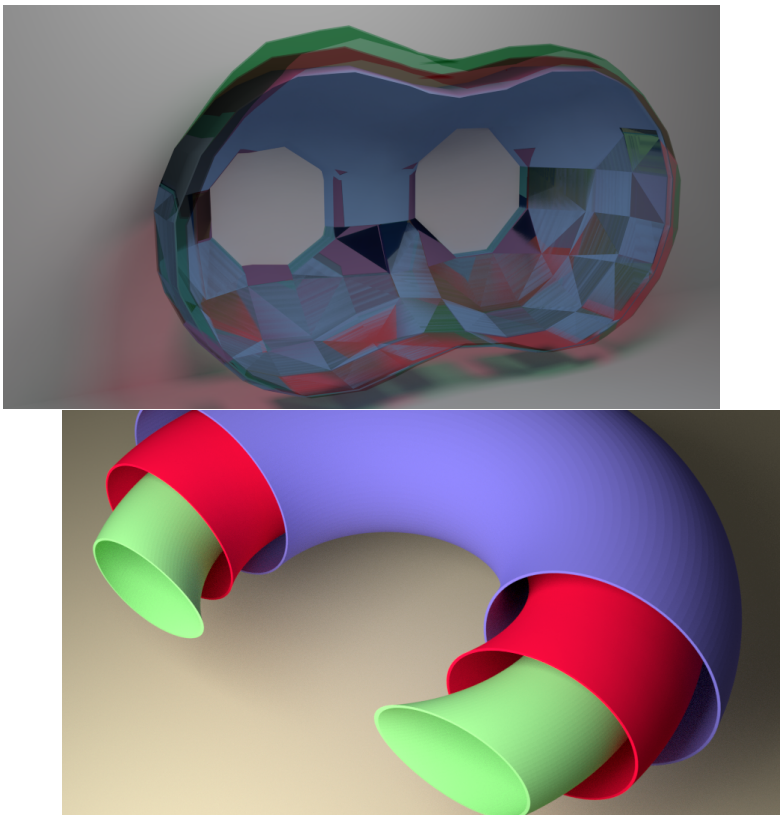

Figure 7.2: Top: A cut-section of the vAS SymmetricSnap average (red) of two surfaces (green, blue). In regions where the surfaces differ, the average surface is symmetrically located, while in regions that they overlap, the average surface is coincident, as evidenced by the z-fighting. To prominently showcase the z-fighting, the inputs surfaces in this example are approximated by low resolution triangle meshes. Bottom: A graded, section view of the average (red) for two tori that differ in Hausdorff distance. The tori are approximated by high resolution triangle meshes.

- Check T and each T_j for self-intersections. Declare invalid if there are any self-intersections. For this purpose, we employ the algorithm in CGAL [109], but, also refer the reader to [110].
- Compute the dot product of the normal of each triangular face of T, and the corresponding face of each T_j. Declare the result invalid if any of the dot products is not positive.

Additionally, the user may also want to check how well the resulting triangulations T_k approximate the input surfaces B_k using prior art such as [111, 87, 78].

7.5.2 Symmetry: Invariance to seed mesh

To study the variation of the Snap result with the choice of seed mesh, we ran Snap on a set of families of triangle meshes obtained as zero level-sets of algebraic expressions, extracted and processed to remove duplicate vertices using Meshlab [112]. For each family, from each triangle mesh, we compute a sequence of evolving surfaces that are the result of successive Move iterations of the SymmetricSnap and NormalSnap. Thus, for a particular Move iteration i, and a particular seed triangle mesh S_j, the intermediate triangle mesh S_j^i is the result of invoking Move i times on the vertices of S_j. In Table 7.1, we report the maximum over all ordered pairs of intermediate triangle meshes (S_j^i, S_k^i) of the mean directed Hausdorff distance for each Move iteration. The change in these numbers gives an estimate of the rate of convergence of Snap. The Hausdorff distances are approximated using the implementation of [93].

7.5.3 Convergence properties

Results on the convergence of Snap from a particular seed surface are reported in Table 7.2. Here, the Snap procedure is performed with each triangle mesh in the family serving as the seed (its vertices are snapped). Over this set of Snaps, the maximum and average distance moved by the points under Snap in a particular Move iteration is reported.

To showcase that Snap is stable, and, that it is thus justified to stop Snap after 3-4 Move iterations, we plot the magnitude of the Move iteration displacement vectors over a number of iterations (Fig. 7.1). We see that the bulk of the displacement happens in the first few iterations, and, that, subsequently, there are no sudden, large Move displacements.

CHAPTER 8

AVERAGING CLOSED, COMPATIBLE NON-PLANAR CURVES

Prior chapters demonstrate the applicability of the proposed ideas for averaging curves in 2D and surfaces in 3D. The generalizability of a core theoretical idea – the definition of the average set as the valley of a symmetric field, and the core algorithmic insights of the Snap iteration – symmetrized projections onto locally queried valley sets are highlighted in this chapter where we present an approach to compute an average curve for a set of compatible 3D space-curves.

We present a Snap method for 3D curves that also yields all-pair correspondences. The correspondences yielded by Snap have the property that each set of corresponding points $\{p_i\}$ are the closest projections of a Snapped point p on the corresponding curves $\{C_i\}$. Thus, for the case of 2 curves, p plays a similar role to one of a point on, a 3D extension of the medial axis (Fig. 8.1).

In this chapter, we first present an overview of the Snap method for closed-loop space-curves. Then, we present a detailed description of the Snap algorithm. Finally, we discuss a number of potential applications including geometric design and data-assimilation.

8.1 Snapping for curves in 3D

The Snap algorithm for computing an average curve for a set of 3D curves requires modifications to extract a valley-set of relevant dimension (a 1-dimensional valley-set), and is outlined in Alg. 7. The steps during a single Move iteration are depicted in Fig. 8.2.

In the discussion in this chapter, we assume that the input curves are the set $\{C_i\}$, with $i \in \{1, 2, \ldots, n\}$. The seed curve is one of the input curves. We alias the seed curve as C_0 and, when we wish to refer to a point on the seed curve, we refer to it as p_0. This numbering convention makes references to quantities derived from the seed curve explicit.

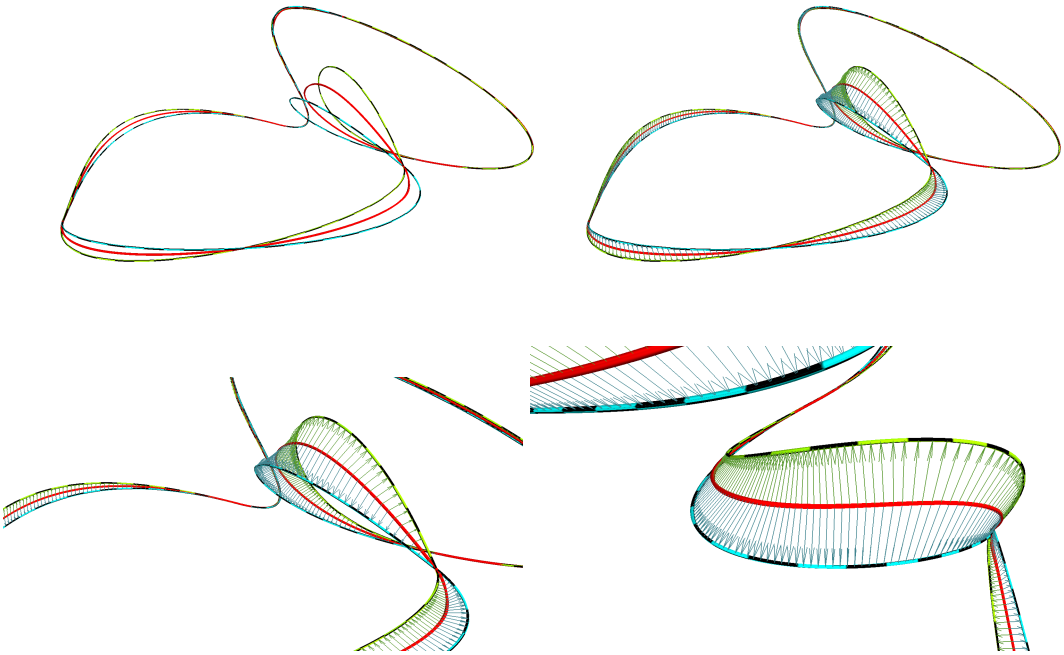

Figure 8.1: Top-left: Two non-planar input curves and their average (red). Top-right: The correspondences that are established (green arrows). Bottom: Different zooms (cropped) of the top-right arrangement.

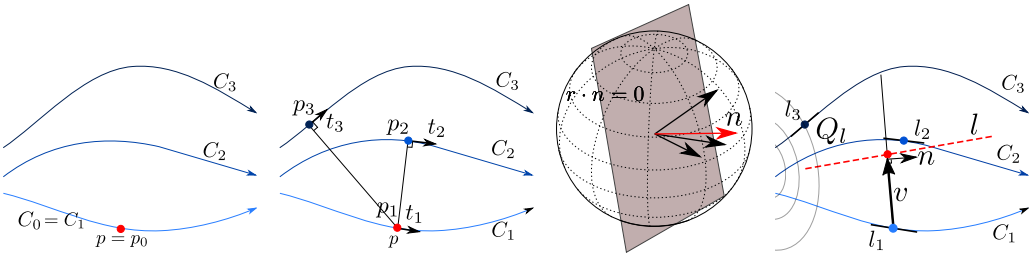

Figure 8.2: The Snap algorithm for space-curves takes as input a set of curves $\{C_i\}$, and a point p_0 on any one input curve, in this case C_1, aliased to C_0 (left). Just as in the case for 2D curves, Snap performs a sequence of Moves. Each Move updates p to $p = p + v$. Computing the new position involves the following steps (center-left to right): a) compute the closest projections $\{p_i\}$ of p onto each C_i and also the tangent t_i to C_i at each p_i. b) compute a 'move' plane P_l through p_0 that has normal n which is computed as (generally) the average of $\{t_i\}$, and, c) compute displacement vector v orthogonal to n such that the valley condition is satisfied at $p + v$. Geometrically, v is computed so that $p + v$ is the intersection of the plane P_l and the average line l.

Algorithm 7 Snap method for curve in 3D

```
1:  procedure SNAP(p_0, ℂ = {C_i})
2:      p = p_0
3:      repeat                                          ▷ Sequence of Moves
4:          for i ∈ {1, ... |ℂ|} do
5:              p_i ← ClosestProjection(p, C_i)
6:              l_i ← (p_i, t_i)                        ▷ Line tangent to C_i at p_i
7:          n ← MovePlaneNormal({t_i})                  ▷ Normal to plane P_l
8:          l ← AverageLine({l_i})                      ▷ Average line to {l_i}
9:          v ← Displacement(n, l)                      ▷ Vector parallel to P_l
10:         p ← p + v                                   ▷ Move
11:     until Converged(p, {C_i})
12:     return p
```

In comparison with the Snap algorithms presented in the earlier chapters, the chief modification is in the computation of the displacement vector. The displacement vector is computed so that $p + v$ lies on the average line to $\{l_i\}$ (for the average of a set of lines in 3D, see Sec. 3.5). The details of how v is computed are presented in Sec. 8.2.

Snap operates on any representation of the input curve set $\{C_i\}$ that supports closest projection from a given spatial location queries and tangent vector at point on curve queries. Snap initializes the desired point p to p_0 (Line 2). Then, it iterates (Line 3) a small number of times (typically four) a step which we call a 'Move'. Each Move (Lines 4 through 9) computes a normal n (Line 7), uses it to compute a Move-vector v (Line 8) in a move-plane P_l that contains p and is normal to n, and finally moves p by v (Line 9).

To compute the normal vector n, we need (Line 7) the tangents $\{t_i\}$ to the input curves at the closest projections of p. So, we iterate (Line 4) over all input curves. For each C_i, we compute the closest projection p_i of p onto C_i (Line 5), and, at the same time (Line 6), the oriented tangent t_i to C_i at p_i.

The missing ingredients in the above description are: how is the move plane normal n computed and how is the displacement vector v computed. n is computed as the average vector for the set $\{t_i\}$ (discussed in Sec. 3.1, but, similar to the case for surfaces, we also discuss another choice in the following section – moving normal to the seed tangent t_0.

v is computed so that $p + v$ lies on the average l of $\{l_i\}$. The method for computing the average line for a set of lines in 3D has been discussed in Sec. 3.5. Recall that the definition requires us to define the field Q_l that at every point in space evaluates to the summed squared distance to the lines $\{l_i\}$ and extract the valley of this field (which we have shown is a line). Additionally, we restrict v to be parallel to the P_l. Thus, geometrically, we compute v so that $p + v$ is the intersection of P_l and l (see Fig. 8.2 (right)).

Now, we discuss the selection of v – the normal to the Move plane P_l.

8.2 Moving to the local average line

The snap algorithm moves onto the currently estimated average line l at each move iteration. While the minima point(s) of the summed squared distance function Q_l to the lines $\{l_i\}$ do lie on l, we have discussed the insuitability of computing the minimum point from a numerical stability viewpoint in Chs. 5 and 7.

Another, perhaps more important reason for not moving to the minima point of Q_l in every Move iteration is due to convergence issues. Consider the dynamic updates to p over a sequence of moves. At a particular iteration, a Move to the minima of Q may significantly alter the points of closest projections on $\{C_i\}$ computed during the next iteration. Thus, the average line, and, thus the new p could be significantly different from the previous iteration. Furthermore, subsequent move iterations could continue to do the same.

We note that, for compatible input curves, given p, a move in a 'transverse' direction of $\{C_i\}$ reduces the variation of the closest projections. Thus, we wish to move to the average line by moving in a transverse direction.

Moving to the average line by moving orthogonal to the average tangent direction: A good transverse direction is one that is orthogonal to the average direction t of $\{t_i\}$. Thus, we find v that is parallel to the plane $P_l(p, t)$ and such that $p + v$ is on l.

We use the average line's valley-set characterization for finding v. Given t, we construct an orthogonal pair of vectors v_1, v_2 that are parallel to P_l. Then, any point on P_l is written as $p + \alpha v_1 + \beta v_2$. We find values of α and β such that the valley condition (Eq. 3.10) holds.

We briefly recall the condition: we require the gradient of Q_l to vanish along two principal directions e_1 and e_2 of the Hessian of Q_l. Thus, we wish that:

$$\begin{aligned} \nabla Q_l(p + \alpha v_1 + \beta v_2) \cdot e_1 = 0 \\ \nabla Q_l(p + \alpha v_1 + \beta v_2) \cdot e_2 = 0 \end{aligned} \tag{8.1}$$

Note that the valley point $p \in P_l$ is not given by the vanishing of the gradient at p. Rather, we require that the gradient at p vanishes in two special directions e_1 and e_2 that are derived not from P_l but from the field Q_l.

Expressing the points and vectors in any particular frame of reference, the above is a pair of linear equations in α, β, whose solution yields a point on the intersection of P_l and l.

Additionally, as we have constructed the normal n to P_l to be along the average t, and thus orthogonal to the eigenvectors e_1 and e_2, there is a special case: we can take v_1 and v_2 to be equal to e_1 and e_2. Then, the above equations can be solved for α and β individually. We use the fact that e_1 and e_2 are orthogonal eigenvectors of H, and that, thus, $\langle He_j, e_k \rangle = \lambda_j \langle e_j, e_k \rangle = \lambda_j \delta_{jk}$. Substituting the expressions of Eq. 3.10 in the above equations, we have the following two equations, each with only one variable:

$$\begin{aligned} \langle Hp + \alpha He_1 + \beta He_2, e_j \rangle &= \langle \Big(\sum_i A_i x_i \Big), e_j \rangle \\ \Rightarrow \langle Hp, e_j \rangle + \alpha \lambda_1 \delta_{1j} + \beta \lambda_2 \delta_{2j} &= \langle \Big(\sum_i A_i x_i \Big), e_j \rangle \end{aligned}$$

Moving to the average line by moving orthogonal to t_0: In the above we espouse the use of a symmetric normal n for the move plane P_l – the average t of $\{t_i\}$. In Sec. 7.3, we discussed a NormalSnap variant of Snap that fixes the direction of move and stated how

it is useful as, upon termination of NormalSnap, we obtain more useful correspondence queries that directly answer the question:'Given a particular point on the seed shape, what are the corresponding points on the other input shapes?'

Thus, we also propose using $n = t_0$ (t_i of the seed curve C_i). How correct is the choice of $n = t_0$? As discussed previously, we want to construct P_l so that the intersection of l with P_l yields a move vector transverse to the average of $\{t_i\}$. For roughly parallel curves, even t_0 is well aligned with the average direction, and, thus, is a good candidate for n.

However, n is computed only using a particular seed curve. Thus, does it not violate the symmetry that we desired? Note that the line l determined as the valley-set of the set of lines $\{l_i\}$ is a property of the set of lines. Thus, even if n and its associated P_l are constructed asymmetrically, as during move, we compute a point on P_l that is also on the average line l (i.e., geometrically, we compute the intersection of P_l with l), we obtain results that are independent of the ordering of the input curves. Hence, the only condition that we require on the choice of n to ensure symmetry is that it yields a plane that intersects with l.

Note, however, that for any move plane P_l that is not orthogonal to the squared dot average of the vectors supporting $\{l_i\}$, we must use Eq. 8.1 and cannot simplify to a diagonal system of equations as e_1 and e_2 do not span P_l.

8.3 Implementation details

Our implementation uses piecewise-linear representations of the input curves and applies the Snap method to a set of ordered samples on any one of the inputs. Implementation details specifying the operations of Alg. 7 for piecewise linear inputs are:

ClosestProjection and ClosestProjectionTangent: We compute the closest projection of p to each segment of C_i. In case p has multiple closest projections on a particular curve C_i, we select any one as p_i. We compute the line(s) supporting the edge(s) of C_i on which p_i lies, and, return an arbitrary line l_i from this set, also noting the direction t_i of l_i.

MovePlaneNormal: We have two implementations for constructing n for P_l: one that uses the squared dot average of vectors presented in Sec. 3.1 and another that uses the inherited normal $n = t_0$. While theoretically, the average point-set doesn't depend on the precise choice of n, practically, as we Snap a set of samples lying on C_0, the choice of n influences the sampling of the average point-set we obtain.

AverageLine: As mentioned in Sec. 8.2, we do not explicitly construct the average line. We do compute the eigen-vectors of the Hessian of Q_l for the set of lines $\{l_i\}$ for use in the computation of the displacement v.

Displacement: The displacement vector v is computed so that $p + v$ lies on the average line l for the set of tangent lines $\{l_i\}$ using Eq. 8.1.

8.3.1 Computational optimizations: RestrictedSnap

The above description presents a simple implementation. Now, we discuss computational optimizations. For piecewise linear representations of the inputs, the most expensive step during each iteration of Snap is the closest projection computation step. Naïvely, given the current Snap position p, and a curve C_i this step requires the computation of closest points between p and each segment of C_i. Faster queries can be performed using a space partitioning data-structure. For example, the 3D Fast Intersection and Distance Computation package [113] of CGAL [114] constructs a tree of axis-aligned bounding boxes, the leaves of which contain just one edge-segment, and, uses this data-structure along with an auxiliary, guessed, non-less than actual estimate of the distance to accelerate closest point computations. We use CGAL's AABB for computing initial closest projection correspondences.

Snapping a point p yields a sequence of points $\{p, p^1, p^2, \ldots, p^n\}$ where p^i is the result of the ith Move. Certain computational optimizations become possible if we assume that the segments (p^i, p^{i+1}) do not cross the medial axis surface of each input.

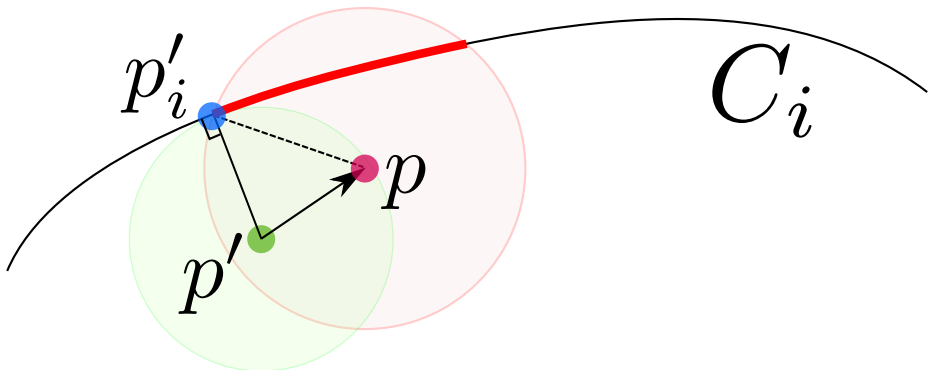

Figure 8.3: Smaller search region for next closest projection. Assume that p' has closest projection p'_i on C_i and that the previous move (not Snap) step moved to $p = p' + v$. We restrict the search of the closest projections of p onto input curve C_i to a ball having center p and radius $\|pp'_i\|$.

Under this assumption, we can utilize the knowledge that the closest projection correspondences do not jump to compute the closest projection for p^i on curve C_j as being given a local minimum of the distance function – an observation that is also made for the Trace algorithm (Alg. 5).

Unlike the Trace algorithm which sequentially traces the average by snapping each seed point in order, the algorithm proposed here – RestrictedSnap – operates on each point under Snap independently. For a point p under snap, RestrictedSnap maintains an upper bound r on the closest distance for each sample being snapped, and only computes closest distances to the connected subset of C_i clipped by $B(p, r)$ that contains p.

The connectivity aware variant of the Snap algorithm uses, at each iteration post the initial iteration, both the current and previous move locations p and $p'(= p - v)$, the correspondences established at these locations to each curve C_i – p_i and p'_i – and, the connectivity of each input curve to restrict closest projection queries to the intersection of C_i and a ball $B_p(\|p - p'_i\|)$ of squared radius $\|p - p'_i\|^2$ centered at p (Fig. 8.3)

For our implementation using polylines, we keep track of the following information: closest projection of p' on C_i, p'_i, as well as an edge of C_i that is incident on p'_i, $e(p'_i)$. We initialize an active set of edges as $A = \{e(p'_i)\}$, the active set boundary as $B = \{e(p'_i)\}$

and, mark the $e(p'_i)$ as visited. We initialize also the minimum squared distance d^2_{min} as $\|p - p'_i\|^2$, and the edge with closest distance $e_{i_{min}}$ as $e(p'_i)$.

d^2_{min} and $e_{i_{min}}$ are updated as we grow the active-set of edges. At each grow step, we add an edge that neighbors B and is not in A to the active set, and add the neighbor of the edge that is not in A to B, until B contains edges that do not intersect $B_p\|p - p'_i\|$, or, until, all the edges of C_i are in A. d^2_{min} and $e_{i_{min}}$ are updated each time an edge that is added to A in a particular grow step has a point that has smaller squared distance to p than d^2_{min}.

Even the initial closest projection correspondence step can be sped up (at the loss of parallelism) in the following manner: we randomly compute closest projections for a fraction of seed vertices on the seed curve C_k. We select the point p' from these samples that leads to minimum average squared distance to all the curves and note its correspondences p'_i and $e(p'_i)$. We select a seed vertex p on C_k that neighbors p'. The connectivity aware algorithm is used to compute the closest projections of p on each C_i by clipping and local querying using $B_p\|p - p'_i\|$ and $e(p'_i)$. This process is then repeated for the neighbor of p that is not p'.

8.4 Experimental results

For a sets of 3D curves, the maximum (over all seed samples being Snapped) of the length of the Move vector during the nth Move iteration are reported in Table 8.1.

Computationally, local closest projection queries are approximately twice as fast as CGAL's AABB acceleration closest projection queries (Fig. 8.4).

Some examples of the computed average and visualized variability tubes are shown in Fig. 8.5.

An example where spatial curve datasets arise naturally is when one wishes to average curve for distinguished curves on free-form surfaces. For example, for certain manufactured parts such as blades for turbines and propellers, one is interested in analyzing differences in blade profiles. These are often 3D curves, which can be extracted manually or as

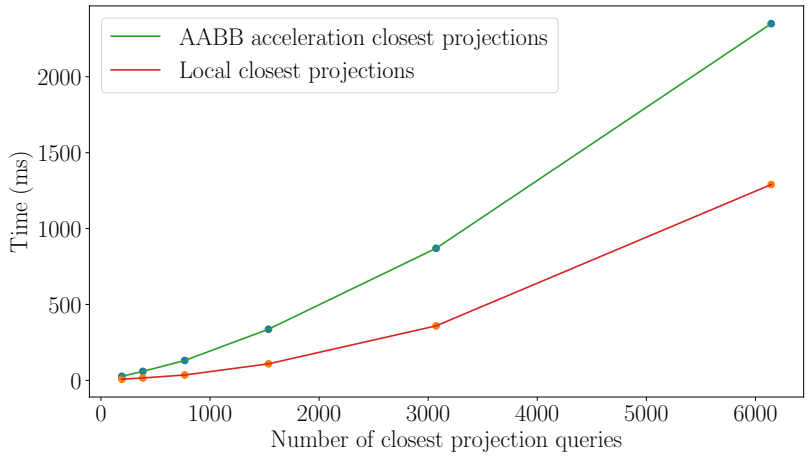

Figure 8.4: Comparison of run times for closest projection queries using a spatial acceleration structure vs localized closest projections described in RestrictedSnap (Sec. 8.3).

Table 8.1: For different sets of 3D curves, the maximum (over all seed samples being Snapped) of the length of the Move vector during the nth Move iteration.

S. No.	Move 1	Move 2	Move 3	Move 4
1.	0.132	0.00059	2.02×10^{-05}	2.99×10^{-06}
2.	0.0830	7.98×10^{-06}	4.20×10^{-13}	6.76×10^{-16}
3.	0.141	0.002	0.00043	8.31×10^{-12}
4.	0.2003	0.00045	4.79×10^{-05}	4.98×10^{-13}
5.	0.274	0.00087	0.00016	3.66×10^{-12}
6.	0.15	0.00049	1.89×10^{-11}	4.48×10^{-16}

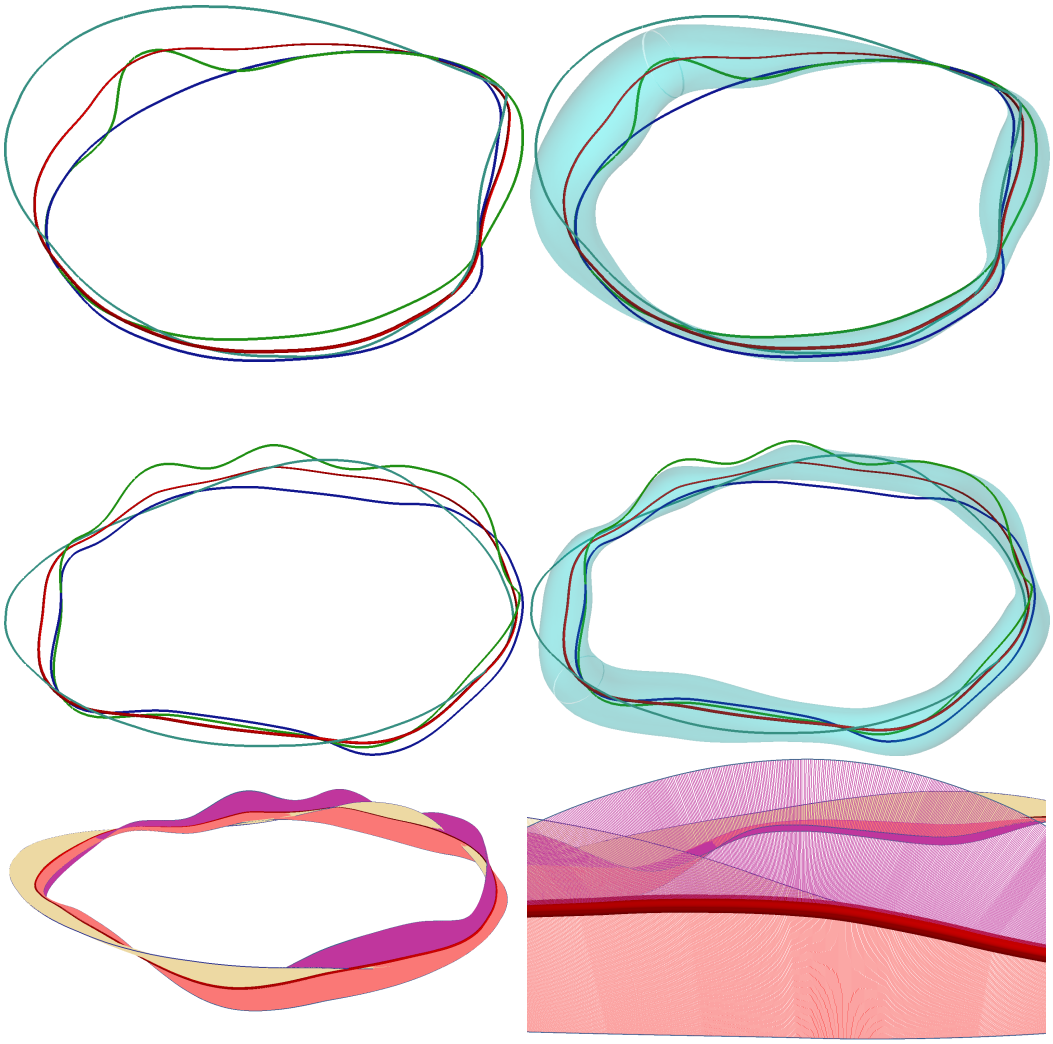

Figure 8.5: Top: Set of input curves their computed average curve (red) and variance tubes (cyan). Center, Bottom: Set of input curves, their average (red), variance tube (cyan) and correspondences established via the average.

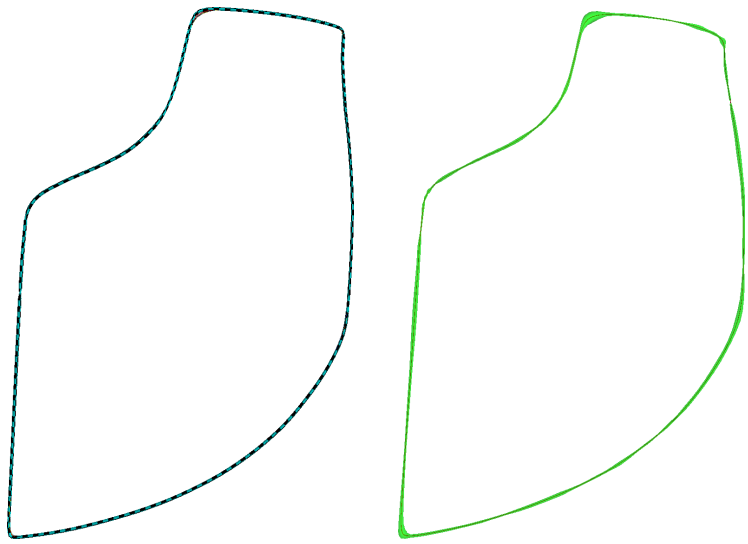

Figure 8.6: Left: A set of 3D curves (one is thickened, while the others are shown with a thin radii) that are the profiles of a free-form blade. Right: The variability tube shown here, with the radius linearly magnified, highlights areas of deviation between the inputs.

the set of points that meet certain (differential) geometric criteria. In Fig. 8.6, we extract profile curves from a collection of such a 3D models, and, visualize the average profile curve and the variability field as a tube around it.

8.5 Applications

Data-assimilation

Given a field F_i (scalar/vector/tensor) defined on each curve, we can construct an assimilated field $F = \frac{1}{n}\sum_i F_i$ defined over the average.

The above technique also allows us to assimilate fields defined in an ϵ region around the input curves, where ϵ is the minimum of the local feature sizes of the inputs. For each point p on the average, we construct the local field for a plane orthogonal to the tangent in the following manner:

- Use the established correspondences $\{p_i\}$ and the tangents at these correspondences $\{t_i\}$ to construct an affine frame $\{p_i, [t_i, R_i e_1, R_i e_2]\}$, where R_i denotes a rotation in

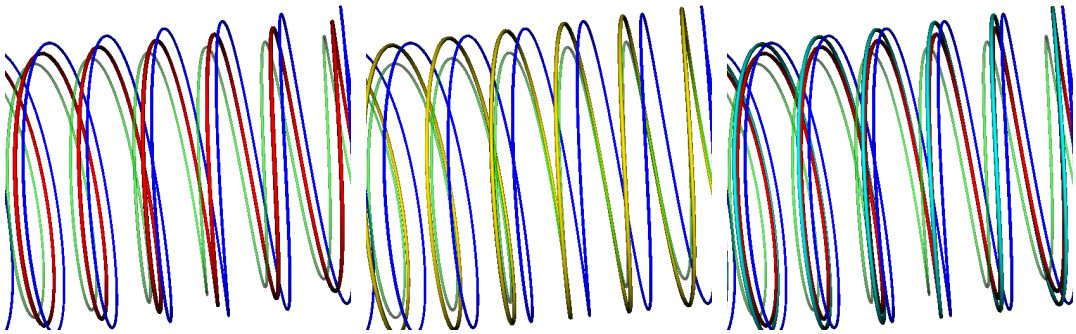

Figure 8.7: Left: Two inputs helixes of varying and radii, and their average (red). Center: A dense sampling of the blue curve is registered using ICP to a dense sampling of the green curve. The registered result is shown in yellow. Right: The blue curve's samples are registered to samples on the average, with the registration result shown in cyan.

a plane through the origin orthogonal to $t \times t_i$ that maps t to t_i, while e_1 and e_2 are the two eigenvectors of the hessian H of the summed squared distance function at p.

- Use these set of affine frames to establish correspondences between a point of the average plane $P(p, t)$ and that of each corresponding planes $P_i(p_i, t_i)$ – if a point on P and one on P_i have the same coordinates in their respective affine frames, then they correspond.
- Average the field contributions at the corresponding points:

$$F(p + \alpha e_1 + \beta e_2) = \frac{1}{n} \sum_i F_i(p_i + \alpha R_i e_1 + \beta R_i e_2)$$

The described approach can be used to propagate patterns defined on one curve to all others.

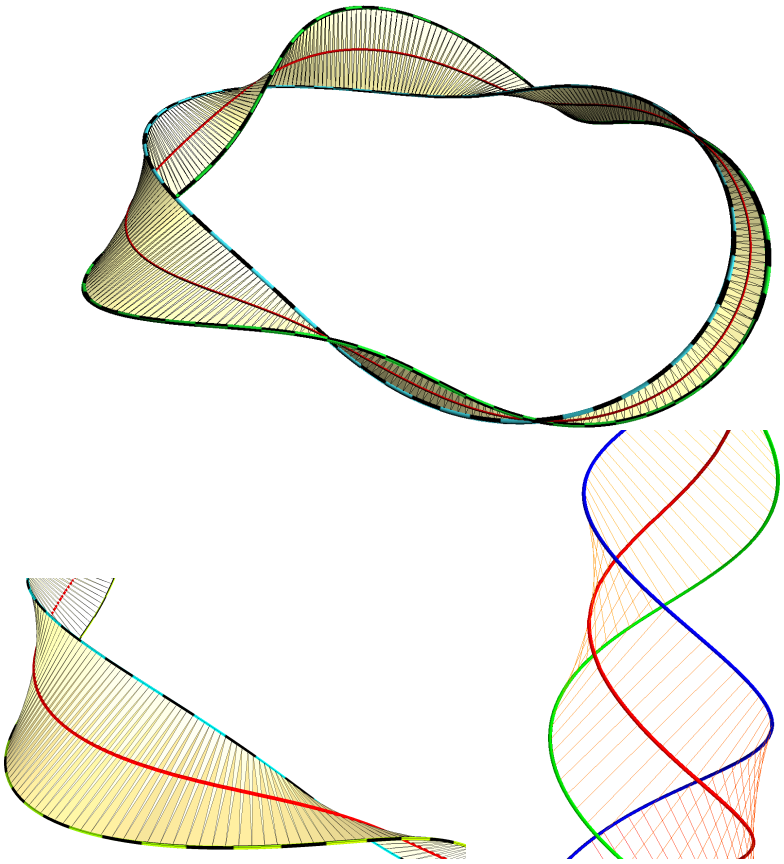

Figure 8.8: Top: Average curve (red) overlayed triangle meshes (yellow) obtained by connecting corresponding points on the two input curves. Bottom left: A zoomed in view of top. Bottom right: Ruling created by connecting corresponding points on two input curves.

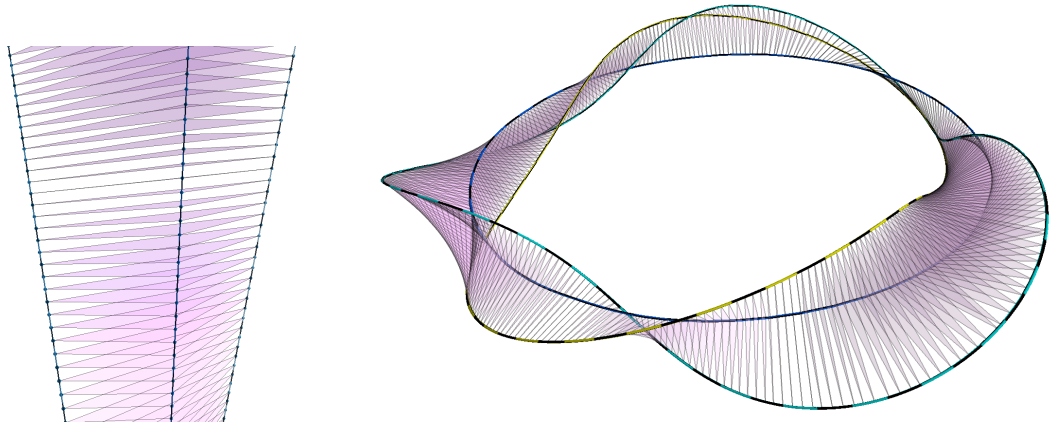

Figure 8.9: Left: Three input lines and a resulting stack of triangles interpolating sets of corresponding samples. Right: The interpolating triangle stack for three input curves.

Rigid registration and tight tolerancing

The iterative closest point algorithm computes a rigid transformation that aligns a source point cloud to a target point cloud. Given an input set, none of whose elements is distinguished, we propose that our presented algorithm be employed in a two step rigid registration process using iterative closest point: a) compute the average b) register each input shape to the average. Fig. 8.7 demonstrates an example result. One way of quantifying the difference between the two registrations is to compute the summed squared distances between each point of the point cloud and its correspondence on the other point cloud. As the average shape's samples are situated 'in between' the input point clouds, the summed variance reported with the average point cloud post ICP are individually about 0.25 times that reported post ICP registration of one point cloud with another for the example above. The use of the average thus, serves as a means to compute tighter tolerance specification for inputs when none of them is a justifiable reference.

Adaptive snapping for collective uniform re-sampling

Thus far, we have described how to snap samples on an arbitrary seed curve to obtain, samples on the average and sampled correspondences on every other input curve. The sampled correspondences thus, depend on the seed samples that are snapped.

One option for generating seed samples for snapping is to uniformly sample the seed input curve. However, this may yield correspondences that are far from uniformly sampled on the other curves. In this section, we discuss two alternate approaches for generating samples on the seed curve.

First, we present a parallel sub-division approach that generates a set of seed samples that result in an average spacing between snapped correspondences that is below a user specified thresh-hold. That is, if $Snap(p)$ and $Snap(q)$ are consecutive ordered average samples computed by snapping p and q, then, the average arc-length $\frac{1}{n}\sum_i \text{ArcLen}(Snap(p)_i, Snap(q)_i)$ between the set of consecutive ordered correspondences $\{Snap(p)_i\}$ and $\{Snap(q)_i\}$ is below a user specified thresh-hold.

Our proposed algorithm starts with a coarse set of ordered samples on the seed curve, and then selectively refines this sampling. Let u, v be the result of snapping two consecutively ordered points on the seed curve's (C_0's) samples. Let the arc length between their corresponding points u_i, v_i on C_i be s_{u_i,v_i}. The user to specifies a desired arc length s which we compare with the average of $\{s_{u_i,v_i}\}$. If the average arc length is less than s, we sample C_0 between u and v, generating a new point for snapping.

For the case of polylines, we first obtain an ordered set of sample points on C_0, consecutive points of which are approximately $3s$ arc-length distance apart. We snap these points to get an ordered set P^1, and a set of corresponding points P_i^1 on each C_i. We compute the arc-lengths on C_i between consecutive elements of each P_i^1, and use these values to compute an ordered set s^1 of average arc-lengths 'travelled' along the input curves between consecutive samples of P^1.

For two consecutive seed curve points $u, v \in P^1$, if the average arc-length between $\{(u_i, v_i)\}$ on the input curves is below s, then, the section of the seed curve C_0 bounded by u and v is removed from further consideration. Else, the point w that is mid-way between u and v on C_0 is constructed as a point to be snapped in the next iteration. For each such w, we keep track of the fact that the neighbors of w are u and v. We also keep track of the average arc length s_{uv} between u and v, and the correspondences u_i of u on the inputs.

During the next iteration, when w is snapped yielding correspondences w_i, we compute the average arc length s_{uw} between $\{(u_i, w_i)\}$, and also the average arc length $s_{wv} = s_{uv} - s_{uw}$. This information is used to create a maximum of two possible points to be snapped in the next iteration, along with the creation of relevant book-keeping information for each created point as described in the paragraph above. This adaptive subdivision algorithm is run for user specified maximum number of iteration. Fig 8.10 shows a result generated using this algorithm.

A sequential variant of the above proposal allows us to generate a fair sampling takes all the curves into account and meet the user constraints on average arc-length advanced more accurately. The algorithm is as follows: Starting from an initial seed curve sample, search for the next sample location on the seed curve that will ensure that the total distance moved along all the curves is a fixed user defined value using binary search. Fig. 8.11 compares the sampling obtained using fair Snap with an approach that snaps uniformly sampled points on the seed curve.

Geometric modeling

Both the computed average, and the computed correspondences can be used for a number of geometric modeling tasks. Some representative examples include.

- *Ruled surfaces through a pair of input curves:* For a pair of input curves, the correspondences established can be used to compute ruled surfaces (in theory – triangle meshes for piecewise linear applications in practice) (Fig. 8.8).

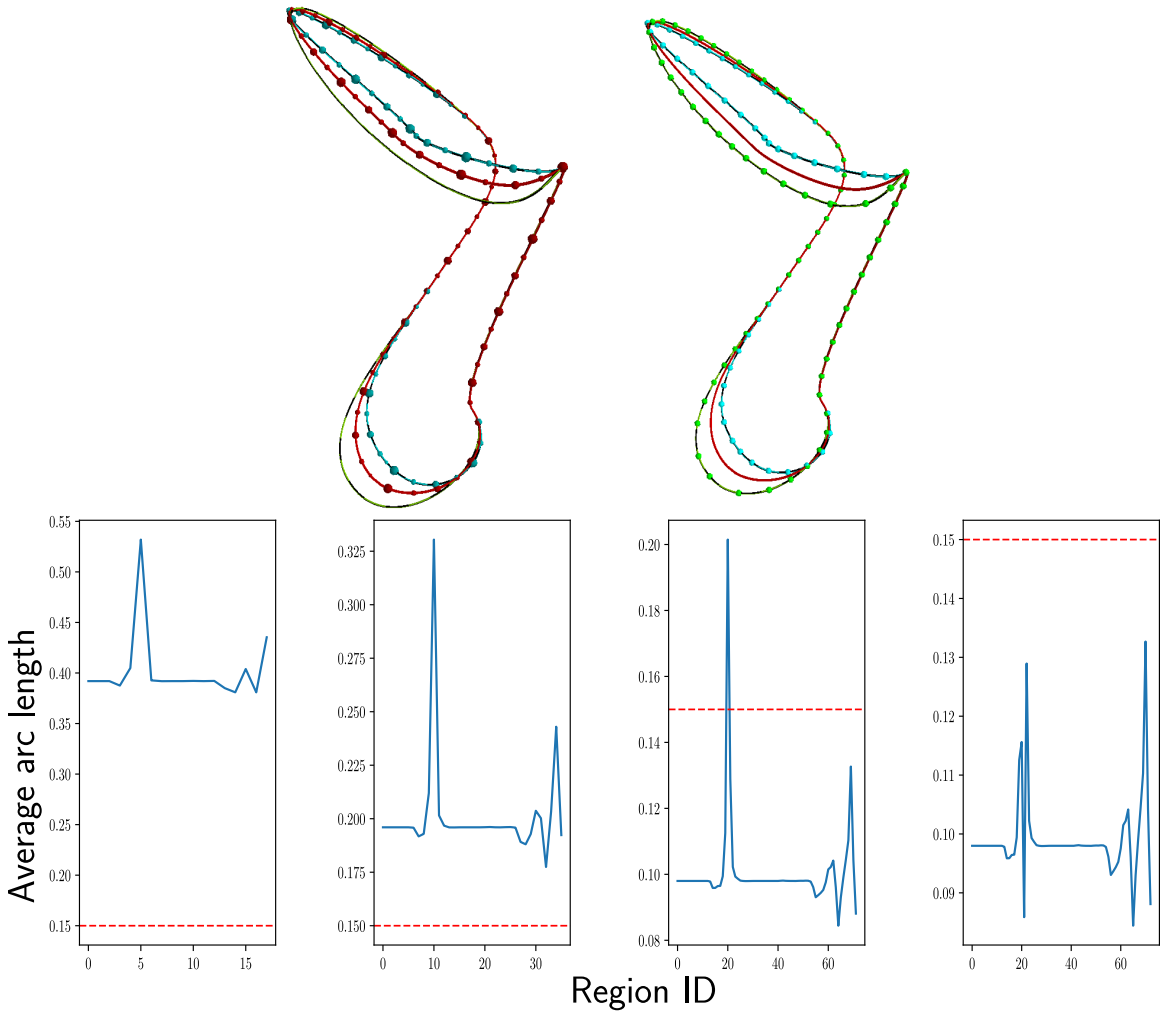

Figure 8.10: Adaptive sampling to ensure that the average summed arc length advanced along the input curves between snapped locations (left, red balls) of ordered points of C_0 is less than the user specified threshold of 0.15 units. Top Left: The samples on C_0 (cyan) and their snapped locations are shown in balls of radius that decreases with the level of refinement at which the samples on C_0 are created and snapped. Top Right: The resulting sampling on the inputs via correspondences established by the snapped result. Bottom: Graphs showing the average arc lengths for consecutive samples at the end of each adaptive re-sampling iteration.

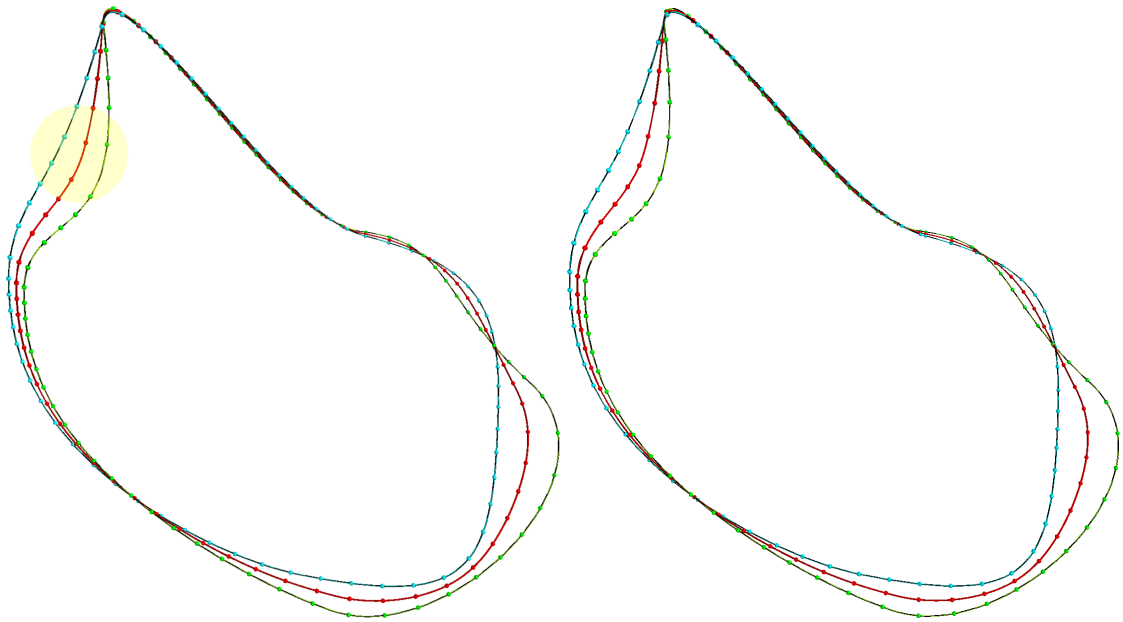

Figure 8.11: Left: A result obtain by snapping a uniform sampling on the cyan input curve. Right: Result from a fair sampling scheme that inserts samples on the average so that consecutive points p, q on the average induce correspondences $\{p_i\}, \{q_i\}$ such that the summed arc lengths $\sum_i p_i q_i$ is a constant value (within numerical tolerance).

- *Triangular tubes through three input curves:* A user can specify a set of three curves and the computed correspondences can be used to compute, for each sample on the average a plane from the induced correspondences (Fig. 8.9).

Spatial synchronization of point motions

Given a set of curves in 3D, we can Snap to their average. Doing so establishes correspondences. Here, we discuss an application where, rather than just input curves (paths), we are given several motions $M_i(t)$ of a point. Then, one way of defining the average motion M is, $M(t) =$ the average of $\{M_i(t)\}$. Suppose however that the input motions $M_i(t)$ are not synchronized, beyond their simultaneous starting and ending constraints. Can we use an extension of the proposed averaging to synchronize or re-parametrize them?

To simplify the discussion, assume that all motions start at time $t = 0$ at point u_0 and end at time $t = 1$ at point u_1. One way to re-parametrize all the motions is to: (a) simply consider the paths C_i (ignoring the timing of each motion M_i), (b) compute the average

curve C in 3D, (c) construct a parameterization of the average curve, and, (d) use the closest-point projection correspondences between C and $\{C_i\}$ to synchronize the motions.

The parameterization or timing information for the average C can be, for example, constructed as the arc-length parameterization or possibly adjusted for ease-in and ease-out. If one wishes instead to take into account the initial timings of the input motions and average them at the same time as one is averaging the positions, then, one can: (a) for each point p on C, compute the corresponding points $\{p_i\}$ on C_i, (b) compute the parameter values for individual motions $M_i^{-1}(p_i)$ and, compute the parametrization of the average curve by assigning to every point p of the average curve C, the 'average' parameter value given by $t(p) = \frac{1}{n}\sum_{i=1}^{n} M_i^{-1}(p_i)$.

Note that one may also take into account the initial timings of the input motions by averaging a set of input curves in the four-dimensional (4D) space-time. Doing so poses two challenges:

- Select the relative scale of time versus the spatial dimensions, as this choice will affect the definition of distance in 4D and hence in the result.
- Provide expressions for snapping in 4D. This includes computing closest-point projections onto a curve in 4D and defining and computing the average of lines in 4D.

CHAPTER 9

EXPERIMENTS ON AVERAGING CURVE SEGMENTS

9.1 Background and motivation

For a set of open curves, in general configuration, there may exist pairs of curves for which, the closest projection map of one onto the other will not be a homeomorphism. However, an often seen configuration in 2D is that of disjointed but locally nearly parallel curves, akin to what a user might draw while sketching (Fig. 9.2, left). This configuration is different topologically from the inputs we have considered thus far, but, locally, geometrically similar to the configurations of curves Snap has been shown to yield useful results for (Chapter 5).

In this chapter, we propose extensions to the Snap method for computing an average for a set of open input curves and present our initial experimental results. The chapter is organized as follows: first, we present an overview of the OpenSnap method – an extension of the Snap method for configurations of nearly-parallel, open curves. In Sec. 9.3, we describe the OpenSnap method in detail. In Sec. 9.4, we present how an average curve may be computed using OpenSnap. In Sec. 9.5, we propose the DirectionalOpenSnap algorithm which yields a representative (i.e., an average) curve-network for input sets that comprise non-parallel curve-segments. The ideas presented in this chapter allow a user to specify a sketch by overdrawing (Fig. 9.1) and to consolidate a drawn sketch by additionally also providing a (set of) guide curve(s) (Fig. 9.2)).

9.2 OpenSnap: Snapping for roughly parallel curve-segments

First, let us consider why Snap (Alg. 1) does not work for a set of locally nearly parallel open curves. The red curve in the center-right image of Fig. 9.2, is obtained by snapping samples on the magenta guide curve. Perceptually, however, the input curves appear to

Figure 9.1: Sketch specification by overdrawing: The user draws a crude sketch (green) and then, overdraws additional strokes to specify the sketch. Top left: the user draws the blue stroke to specify the red average. Top right: The computed average in the previous interaction is now the specified sketch (magenta) and the user additionally specifies a overdraw (cyan). Bottom left: Finally, the user augments the current average with the black overdraw. Bottom right: The initial sketch and the final specified sketch are shown together.

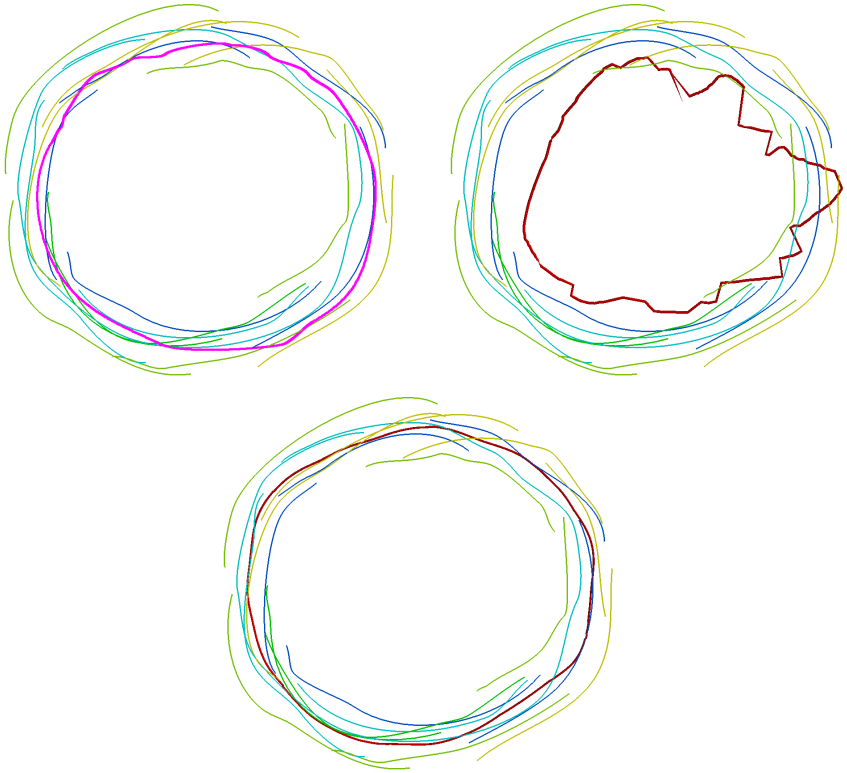

Figure 9.2: Top-left: A set of input curves with the user specified sampling curve (magenta) overlaid over the inputs. Top-right: The result of snapping samples on the sampling curve without clipping. Bottom: The average curve (red) computed by snapping samples on the sampling curve using OpenSnap.

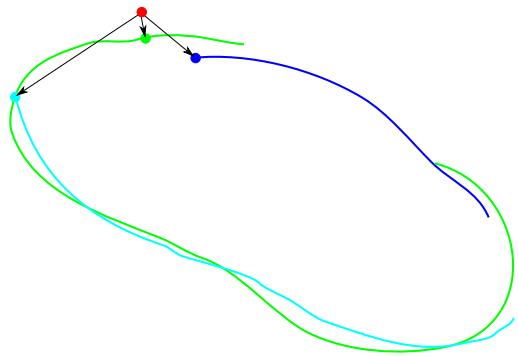

Figure 9.3: As open curves are incompatible, an unclipped closest projection query at a query point (red) places distant samples on the inputs (end point of arrows) in correspondence.

be clipped portions of hand-drawn circles, and, the red curve of Fig. 9.2 (right) better represents the input-set. Thus, rather than placing samples on distant curves (Fig. 9.3) in correspondence, we wish to modify the closest projection computation step of Snap. This modification is detailed in Sec. 9.3.

A subtler issue is demonstrated in the configuration of Fig. 9.4 (center). Here, Snap yields a discontinuous result. To obtain a smooth result, we propose algorithmic changes that effect the designed join curve-segment. Summarily, we modify both the geometry of each input curve, as well as its contribution to the symmetric scalar field. The result is a smooth join curve (Fig 9.4 (right)).

The OpenSnap method for nearly-parallel open curve-segments is listed in Alg. 8. As discussed in the previous paragraphs, essential novel additions to Alg. 1 (Snap for Jordan curves) are:

- *Geometry modification:* The input in pre-processed via the ModifyCurves method, which modifies the boundary geometry of each input curve. In our experiments, the geometry modification step extends the boundary geometry of each input using a linear extrapolation.
- *Active input-set selection:* For each (extended) input curve C_i, the closest projection point p_i for each curve C_i is checked for containment in a clipping shape and is discarded if it is outside.

Figure 9.4: Left: A simple configuration of a line and a half-line, which demonstrates the need for designing the join curve. Center: Symmetry dictates that the red average curve should be situated in between the two objects in some regions, and, should be co-incident with the line in other regions. Right: Aesthetics considerations warrant a smooth transition between these two portions of the average.

- *Field modification:* v is computed so that $p + v$ lies on either the *weighted* valley average or the *weighted* zero-set average of the lines $\{l_i\}$. The process of computing the weights is described in Sec. 9.3.2.

Algorithm 8 Snap method for open curves

```
procedure OpenSnap(p_0, ℂ = {C_i})
    p = p_0
    repeat                                          ▷ Sequence of Moves
        r ← MaxClippingBallRadius
        {C_i} ← ModifyCurves({C_i}, r)
        for i ∈ {1, ... |ℂ|} do
            p_i ← ClippedClosestProjection(p, C_i)
            l_i ← (p_i, t_i)                        ▷ Line tangent to C_i at p_i
        n ← AverageOfVectors({t_i})                 ▷ Normal to move line L_l
        v ← Displacement(n, {p_i})                  ▷ Vector parallel to L_l
        p ← p + v                                   ▷ Move
    until Converged(p, {C_i})
    return p
```

Below, we describe each of these differences in greater detail. Then, in Sec. 9.4, we describe how we obtain an average curve using OpenSnap.

9.3 OpenSnap details: Clipped correspondences, geometry and field modifications

Snap iterates the following process:

- Compute corresponding points.
- Compute locally valid, symmetric scalar fields from the correspondences.
- Project onto subspaces derived from this scalar field.

Another evocative description is that there are scalar fields f_i associated with each input curve C_i that depend on both the point p_i, and the spatial location p that is querying the field $f_i(p, p_i)$. The form of these fields is determined during Snap's correspondence finding step when both p and p_i are known. When correspondences are determined, we 'pick up' fields associated with the corresponding points and sum them up to compute $f(p) = \sum_i f_i(p, p_i)$.

OpenSnap modifies both these steps. In fact, we have seen modifications to either effect already: correspondences are modified in Trace and when using a gap relative projection (Sec. 6.5). The field 'picked up' is modified (for an entire curve) to a weighted squared distance field when morphing (Sec. 5.5). For OpenSnap, we modify the correspondences by a clipping step, and, modify the field locally along the curve at its boundary. Additionally, we also modify the curve's boundary.

The rationale for each step of OpenSnap is: a) the clipped projection seeks to determine 'active' curves and corresponding points on them that locally contribute in the computation of the average, b) the geometry modification allows the correspondence finding step to 'pick up' fields in regions of interest to us and c) the field modification seeks to influence the shape of the average curve.

9.3.1 ClippedClosestProjection: Selecting active curves

At every iteration of OpenSnap, a clipping ball is instantiated centered at p. The clipping ball is used to clip away non-intersecting curves. Note (Alg. 8) that we clip the modified curves against the clipping object to get the active set.

The radius of the clipping ball is a parameter we have control over. In our work, we only experimented with a constant user-specified radius. At every point being snapped, we use the fixed radius clipping ball to decide active curves.

9.3.2 Join curve specification using field and geometry modifications

Fig. 9.4 highlights the need for algorithmic modifications for obtaining a smooth average. Snap's result has discontinuities near the boundary of each input curve. We design the 'join curve' using justifiable but subjective, aesthetic decisions.

We propose the use of geometry and field modifications to control the result of OpenSnap and realize the designed join curve. Our proposed approach is inspired by the use of 'influence' shapes [115, 21]. We locally modify the geometry of each input curve C_i and associate a scalar 'weighing' field w_i to each C_i. w_i is 1 in the interior of C_i and smoothly decays to 0 at the boundary of the modified C_i. For a given query point p, during Move, we compute clipped closest projections $\{p_i\}$ to the modified geometry and Move to the valley-set (in our implementation) of the weighted summed squared distance field $Q(p) = \sum_i w_i(p_i)Q_i(p_i)$.

We wish to design our modification objects so that snapping a 'nicely sampled' set of points in the 'gap' (this term is used here in a gestalt manner, as the gap is no longer defined set-theoretically), results in a 'nicely sampled' set of points in 'a nicely shaped' valley, for a useful set of input configurations. Fig. 9.4 shows OpenSnap's result for the line, half-line configuration obtained from an implementation where we use a simple linear extension as the geometry modification, and a more involved decay field as the field modification. We now describe each modification in detail.

Geometry modification

The geometry modification step extends the boundary of each C_i by an arc-length of r using linear extrapolation. We wish to extend each curve by an amount that is commensurate with the radius of the clipping ball used for detecting active curves. The geometry modifications depend on the approach used for selecting the radius of the clipping ball. In our work, we extend each curve by the clipping ball radius.

Field modification

The field modification is performed by associating scalar weighing fields w_i to each (modified) C_i, with each w_i decaying to 0 on the boundary of C_i. During Move, for each clipped closest projection result p_i, we also obtain the weight $w_i(p_i)$ and move to the weighted average of lines $\{l_i\}$ as described in Sec. 3.6 using the weights w_i. For C_i, w_i is not 1 only on the modified geometry. Thus, w_i is always 1 if C_i does not have a boundary.

For curves with a boundary, we must use different decay strategies based on how the modified geometry is created, and, hence, based on how the clipping ball radius is computed.

For the case of a constant clipping ball radius, we use a quintic decay function for the weights. Assuming that C_i is parameterized using arc length, a representation for w_i is as a univariate scalar function from arc length to the weight value. $w_i : \mathbb{R} \rightarrow \mathbb{R}$, with, $w_i(s) = a_0 + a_1 s + a_2 s^2 + a_3 s^3 + a_4 s^5 + a_5 s^5$. For aesthetic join curves, we require: $w_i(0) = 0, w_i(r) = 1, w_i'(0) = 0, w_i''(0) = 0, w_i'(r) = 0, w_i''(r) = 0$ where the derivatives are wrt. arc-length s. These conditions can be solved for coefficients of the quintic polynomial. Similar conditions are expressed for the other boundary of each input C_i.

9.4 Average curve for parallel curve-segments using OpenSnap

OpenSnap snaps a candidate point to the average. The next question is: what set of points to snap to obtain a useful average curve? Note that, for the case of compatible inputs, samples on any input could be snapped. However, as seen in Fig. 9.4(center), the red curve obtained by snapping samples on the cyan half-line only is not sufficient. The answer to the question of the set of points to snap depends on if the user wishes to specify a sketch by overdrawing, or, perform sketch consolidation.

When the user wishes to specify a sketch by overdrawing, she starts with an initial curve and sketches regions where the initial shape has been incorrectly specified. This process

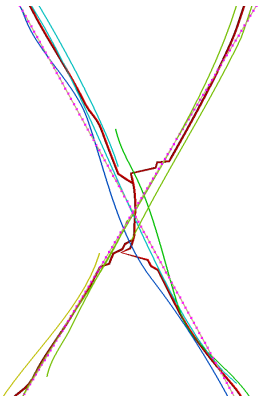

Figure 9.5: The representative curve (red) computed by snapping a set of sample points (magenta) using OpenSnap for a set of input curves that contain curve-segments that are not parallel locally.

can be performed repeatedly (Fig. 9.1). For sketch specification, samples on the initial shape are snapped.

For a sketch consolidation task, the user wishes to average a given sketch rather than refine an initially specified stroke. In this mode, we require the user to also specify a 'sampling' curve. Samples on the sampling curve are Snapped using OpenSnap. Fig. 9.2 shows a result obtained by using OpenSnap and snapping samples on a user specified sampling curve. A set of disconnected average curves (an average-curve network) is obtained if the user specifies a set of sampling curves.

9.5 DirectionalOpenSnap: Average curve-network from non-parallel inputs

OpenSnap, like Snap, yields useful results when the input consists of curve-segments that are locally parallel. When this is not true, OpenSnap fails to yield useful results (Fig. 9.5). To support such configurations, we propose DirectionalOpenSnap that also consider directional information when deciding active curves for a point under snap.

DirectionalOpenSnap accepts a candidate point and a candidate tangent. It only differs from OpenSnap in requiring as input the candidate tangent, and, in implementing ClippedClosestProjection differently. The input to DirectionalOpenSnap is a point and a tangent sampled from a user drawn curve network. Apart from clipping closest projection queries

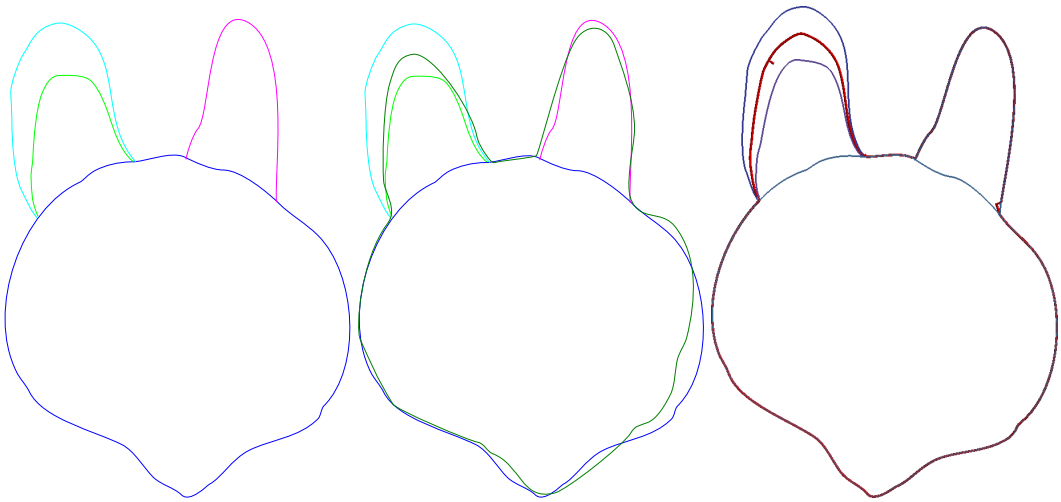

Figure 9.6: Sketch specification by overdrawing. Left: An input sketch of a bunny with one droopy ear that the artist tries to correct by overdrawing. Center: A guide curve network (dark green) over the inputs. Right: The user's specified sketch (red) is computed as the result of snapping the seed curve network.

using a ball, DirectionalOpenSnap also removes closest projections on active curves where the tangent is not within 30 degrees of the input.

Fig. 9.6 is a result of using DirectionalOpenSnap to compute representative curve-networks by Snapping samples (point, tangent at point) from a user-specified sampling curve-network.

9.6 Conclusion

This chapter presented our modifications to Snap to enable it to operate on parallel, open curves. We presented initial experimental results and discussed and proposed solutions to key issues that arise in extending Snap to open curves.

While in this thesis, we have proposed a method to augment a sketch with a guide curve network that provides seed samples for Snapping, for a overdrawing based sketch-specification application, it may be desirable to not require the specification of guide curves. Using the ideas presented in this chapter, given a set of input strokes (open curves), such a system can work by a) pre-processing the input strokes b) snapping samples $\{p_i\}$ on each

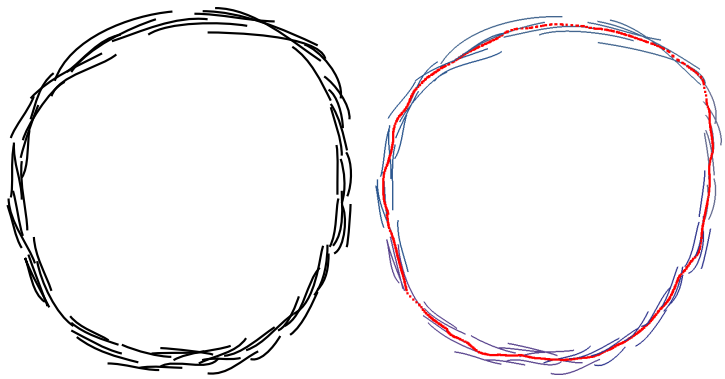

Figure 9.7: Sketch consolidation without using a guide curve by snapping all inputs. Right: A set of short strokes to be consolidated. Left: Samples (red) obtained by snapping samples on each of the input strokes overlaid over the inputs. A post processing step for extracting a curve from the samples is required.

input curve C_i to get a snapped point-set $\cup Snap(\{p_i\})$ c) creating topology for the snapped point-set (determine neighborhood information for each point of the snapped point-set) using trivial nearest neighbor algorithms to get snapped curve(s) d) re-sampling the snapped curve(s). As a preliminary result, for a set of short drafting strokes, the resulting point-set obtained by snapping samples on all inputs is shown in Fig 9.7. Note that despite no connectivity information, the snapped points are suggestive of a curve, and, for example, one way creating a curve from these samples is to extract the crust [78].

We hope that our proposed technique will provide for online sketch-consolidation / sketch-specification through overdrawing by the judicious use of spatial acceleration structures, point-wise (parallelizable) projection operations, and serve as an alternative to more elaborate and computationally involved sketch-consolidation systems such as [116]. However, a number of challenges will need to be addressed for the successful use of the techniques discussed in this chapter in an end to end system for sketch consolidation. One example of such a challenge is: the ends of strokes are often drawn and captured (by a digital drawing tablet) with less precision and may also possess abrupt changes in direction. For the sketch consolidation task, the boundary of each stroke might need to be clipped during a pre-processing step before OpenSnap, or, a different technique for geometry modification rather than linear extrapolation may be required.

CHAPTER 10

AVERAGING TOOL-PATHS FOR ANALYZING SYSTEMIC ERROR AND NOISE IN A CNC MACHINING PROCESS

10.1 Background and motivation

In this chapter, we present a use-case of the average curve and of the variability field computed using the Snap algorithm: analysis of a computer aided manufacturing process that uses CAD software for digitally specifying the shape to be manufactured, and CNC machining for realizing the designed shape.

In computer aided manufacturing, the "As Designed" shape designed using a CAD system (we term this the nominal shape) occupies a central role in informing manufacturing process decisions. For example, the nominal representation is used to compute and provide control signals for driving the manufacturing of the physical artifact and to compute manufacturing tolerances from. The nominal shape can also inform process improvements. Comparing manufactured artifacts to the nominal shape, an operator can, using her experience, make an informed decision on if a different cutting tool would yield better results, or, if the feed-rate needs to be reduced. The usefulness of digital models for aiding decision making is a key reason for the increasing trend in manufacturing to use an explicit "digital twin" [117] – a digital representation of the entire manufacturing pipeline rather than just the nominal shape.

Additionally, in modern day manufacturing, instrumented manufacturing systems which provide online updates as to their status are increasingly prevalent. These systems leverage our improving ability to store and process large datasets and to interconnect complex systems.

In fact, we have demonstrated that using the architecture described in [118], it is conceivable to obtain in near-realtime, samples that estimate the "As-Manufactured" shape. Given systems capable of high-frequency data feedback (such as [118]), are there new possibilities in how the acquired data can aid decision making? In this chapter, we present one such possibility that highlights the role of the average shape.

Consider first the following setup that provides the operator with comparative measurements between the artifact manufactured by a production run and the nominal shape:

- The operator designs a nominal digital D^* representation using CAD software.
- D^* is used by a computer aided manufacturing (CAM) software to compute a motion representation.
- The motion representation is parsed by a machine control software which uses it to generate control signals for driving the machine tool.
- During a production run, the control signals are applied to the machine tool for realizing the designed shape.
- For the ith production run, data read back from instrumenting sensors is used to construct a digital estimate D_i of the manufactured artifact.
- D_i is compared with D^* (Fig. 10.2).

In such a setup, D_i differs from D^* due to errors in the manufacturing and digital capture processes. Note that a systematic error such as one caused by a wrongly calibrated machine or a wrongly calibrated sensor would affect each D_i. The key observation is that a setup which furnishes the operator with comparative data referenced off the nominal shape inherently conflates two types of errors – consistent / systematic errors that affect each machined artifact similarly, and, random errors that affect each manufactured shape independent of their affect on any other.

However, if an average D of $\{D_i\}$ is computed, then, a comparison of D and D^* will showcase the "systemic" consistent errors (Fig. 10.1). Thus, the average shape informs the

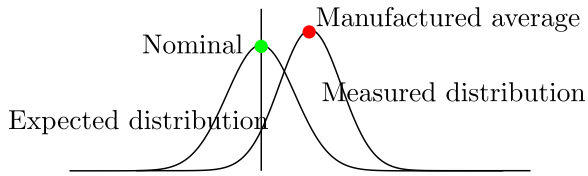

Figure 10.1: A conceptual depiction of how averaging multiple digital representations of the artifacts produced by a manufacturing process allows for disentangling consistent and inconsistent / random errors in the manufacturing process. A comparison of the manufactured average with the nominal shape highlights consistent errors. A measure of variability around the average quantifies the extent of random errors in the manufacturing process.

operator of the nature of the errors in the manufacturing process and allows her to separately consider process improvements for remedying consistent and inconsistent errors.

In the following sections, we demonstrate the usefulness of the average curve and the variability field for informing the motion of a ball-ended cutting tool for a CNC machine. We first obtain time-series data for the positions of the center of the ball of the cutting tool for a set of production runs each of which machines the same geometry (a curve). Then, we compute the average curve of cutting tool center positions and the local variability field and visualize these results. The average curve is also compared with the nominal curve. We demonstrate how to use the computed average and variability information to aid in re-planning the CNC cutting tool's motion.

We have developed a subtractive manufacturing system that allows for:

- Controlling the cutting tool motion for a particular CNC machine.
- Designing the geometry of the path to be traversed by the cutting tool.
- Designing and/or specifying the cutting tool's motion.
- Estimating the realized motion.

We use this system for our demonstration. Thus, first, we describe our developed system. Our system employs SculptPrint [119], an existing voxel based CAM system for designing tool path curves: curves along which a ball-ended cutting tool's center moves for realizing the desired shape. We augment SculptPrint with the capability to construct a time-parameterization of the tool path curves (i.e., to compute a tool path motion) that is

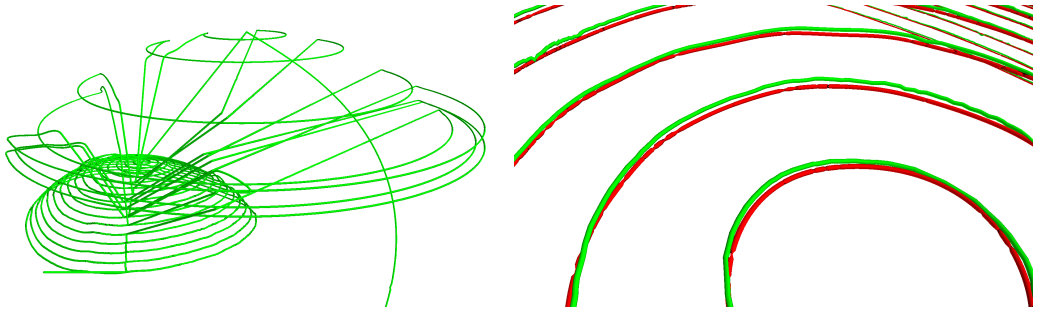

Figure 10.2: Left: A time varying position vector curve of the center of a ball-ended CNC cutting tool for a machining pass (green). Right: The data captured from the instrumented machine (red) overlaid over the desired path (green).

realizable by a given CNC machine. We use the PocketNC CNC machine for our experiment. PocketNC is a commercially available, desktop-sized CNC machine. The PocketNC machine is instrumented to sample the realized motion of the CNC machine's cutting tool using axis coupled rotatory encoders. We control PocketNC using a modified version of Machinekit, an open-source CNC machine control software.

10.2 Description of the tool-path data acquisition system

A schematic overview of the system is provided in Fig 10.3. The different components comprising the system are:

10.2.1 PocketNC and MachineKit

Machinekit [120] is a Linux-based open-source CNC control platform. It is a community effort that builds upon the Enhanced Machine Controller project [121] which lead to the development of a fully-featured software controller capable of controlling multi-axis machines.

The machine used in our system is the PocketNC, a commercially available, 5-axis desktop-sized machine. PocketNC is controlled by Machinekit running on a small single board computer: the Beaglebone Black (BBB). PocketNC's BBB runs Linux that is patched with the Xenomai real-time scheduling micro-kernel. Xenomai allows for the creation of

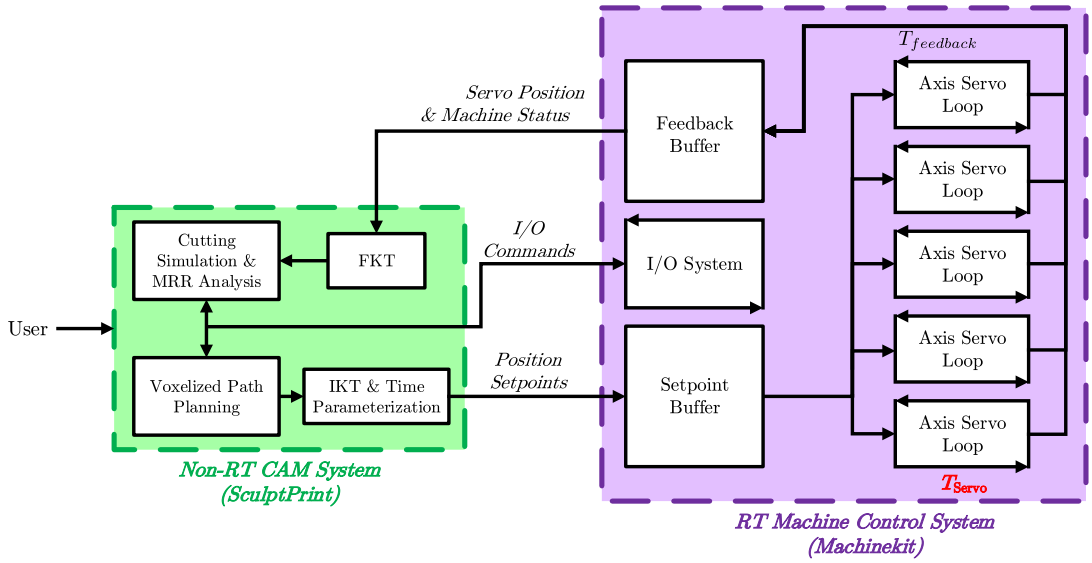

Figure 10.3: A block diagram of the developed system: the CAM system (SculptPrint) allows the user to design the geometric path to be following by the cutting tool. The designed path is fed to a motion generation algorithm whose output is used to drive the CNC machine using a modified version of the open-source control software MachineKit. Rotary encoders attached to each axis of the machine are read at a particular sampling rate to obtain samples on the realized tool path (image originally appeared in our publication [118]).

real-time threads and also allows real-time threads to preempt non real-time threads thus providing low worst-case scheduling latencies vital for control of the CNC machine.

10.2.2 SculptPrint

Computer-aided manufacturing software generates motion commands for a CNC machine tool using stock and input part geometry (the nominal shape). The motion commands affect the motion of the cutting tool along tool-paths curves. The material removed by the cutting tool as it moves along the tool-path curve transforms the stock shape to the desired nominal shape.

Both the nominal shape and the tool paths are often designed manually. While the designed geometry is often described using parametric curves and surfaces, our system utilizes a CAD/CAM software, SculptPrint [119], that represents the geometry using voxels. A voxel representation of a parametric shape is computed as follows: space is subdivided

into a grid of volume elements (voxels). The position of a particular voxel is implicit, and, a boolean value is stored for each voxel to denote if the voxel intersects with the shape's boundary. SculptPrint uses an adaptive, hierarchical, voxel representation, using small voxels only near the solid's boundary. The use of such a representation and the implementation of efficient algorithms for common solid modeling tasks on such a representation (detailed in [122]) ensures that high resolution approximations of the tool-path and of quantities such as the material removed during machining can be computed at quick (though not interactive) rates. The novel representation and efficient, embarrassingly parallel processing algorithms allow SculptPrint to provide numerous novel design primitives for designing the geometry of the tool-paths for the cutting tool.

10.2.3 Trajectory generation

Given the above setup for tool-path geometry generation and CNC control, one requires additionally, a trajectory planning algorithm, which takes as input a high-resolution discrete approximation to the user designed tool-paths and outputs a cutting tool motion that affects the path.

Conventionally, motion commands are specified using a human-readable file that contains a sequence of 'G-Code' instructions. A G-Code instruction specifies simple geometric primitives (lines, arcs, and splines) and control information such as feed-rates and spindle speed. The machine tool software controller parses G-Code and uses information about the feed-rate, the machine's capabilities, and the geometric primitive specified to generate reference motion that is to be tracked by the machine. For a machine that uses stepper motions (as PocketNC does), the controller also generates a pulse-train to realize the reference motion.

In contrast, our system allows for free-form machining using a direct control architecture [118]. Rather than specifying a CNC cutting tool's motion by means of G-code, we modify Machinekit's trajectory planner to allow us to directly feed required control data to

the CNC machine's motor controllers at their rate of operation, $\frac{1}{T_{servo}}$. This modification of the trajectory planner of Machinekit allows us to directly control the machine's motion at the machine's native temporal and spatial resolution.

Armed with this modification, we directly fit and optimize the parameterization of a spline that interpolates axis position samples provided by SculptPrint using an approach similar to [123]. Our developed trajectory planning algorithm involves three steps: a) the fitting of a curve in 5D CNC controller configuration space to discrete samples obtained from the CAM system b) the optimization of the time parametrization of the fit curve while adhering to posed constraints and c) the uniform time sampling of poses from the optimized parametrization.

The trajectory planning component of our system is flexible enough to allow us to specify kinematic constraints that are to be enforced at any point on the configuration space path.

10.2.4 Encoder feedback

While the trajectory planner does generate a cutting tool motion, it is not necessarily the motion that results when the CNC controller drives the machine's axis using set-points computed from the planned motion.

The resulting motion is estimated using samples read from a set of rotatory encoders, each of which is rigidly coupled to an axis of the CNC machine. This yields configuration space samples for the machine. Using the machine specific forward kinematic transformations, we obtain samples on the machined shape.

Due to the physical nature of the data-acquisition system, the sampled encoder data has noise in both the spatial and temporal domains. Thus, the data we obtain as estimate of the realized tool-path differs from the nominal tool-path as a result of errors during both the CNC machining process, and the data-acquisition process.

10.3 Experimental results

10.3.1 Experiment description

In our demonstration, we consider two simple 2D nominal tool-path curves: a filleted square (Fig. 10.4) and a circle. For each curve, applying machine specific inverse kinematic transformations, we obtain the configuration space curve along which the machine's axis must move. We use manufacturer prescribed kinematic constraints for the PocketNC to plan a realizable motion using the trajectory planner. The acceleration and velocity profiles for the machine axes for the planned motion are depicted in Fig. 10.4 (one machine axis is stationary).

The motion is sampled at servo rate as per our control strategy and provided to the controller for driving the axis motors. Note that, we *do not* cut material, rather, present data captured when cutting air. Values read from the encoders yield samples of the positions of the different axes of the CNC machine. Applying the machine specific forward kinematic transformation, we obtain sample positions on the realized tool path. As expected, the captured samples closely track the nominal tool-path.

10.3.2 Data analysis

There are two types of analysis enabled by the proposed curve averaging solution. Firstly, we can compute regions on the input curve where the distance between the average encoder curve and the nominal curve is higher than a particular threshhold – 'non-conforming' regions. We can bring these regions of high deviation to the attention of the CNC operator. These two use cases are illustrated in the below discussion.

Use of the average: In Fig. 10.5, the average curve constructed from encoder data from multiple runs and the nominal curve are overlaid. We zoom in on a particular region to show the difference in the two curves.

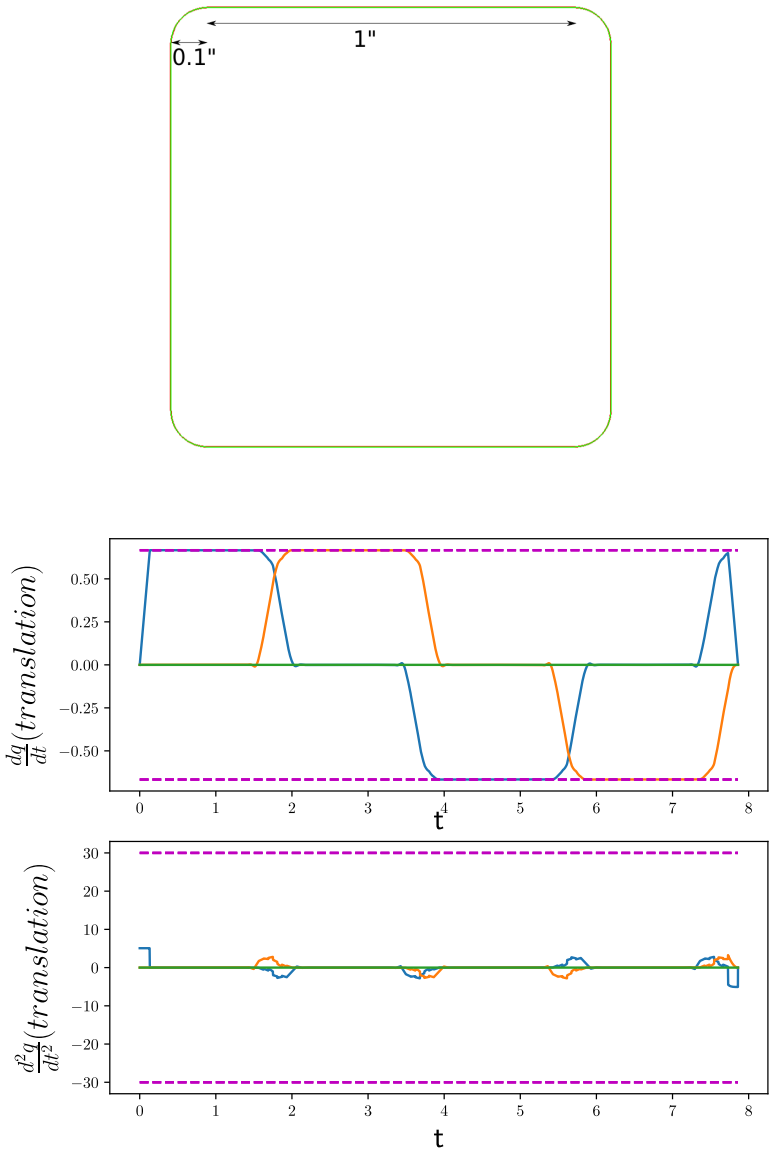

Figure 10.4: Top: The path used for our experiments (green). Center and Bottom: Velocity and acceleration (bottom) profiles for the translational axis for the motion generated by the trajectory planner.

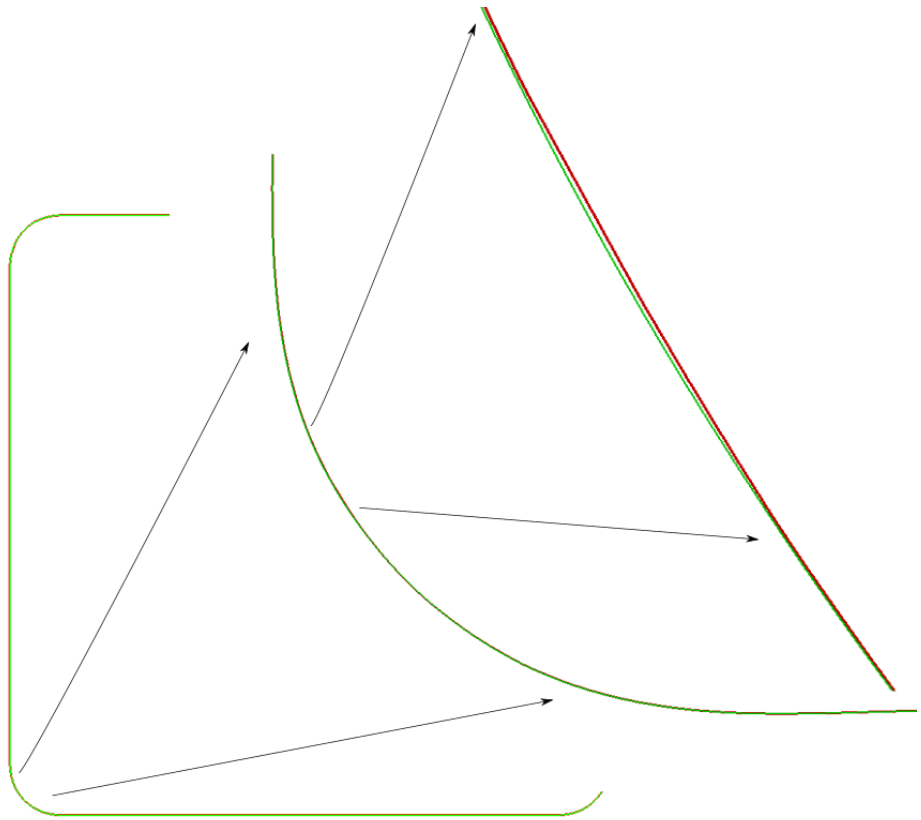

Figure 10.5: The average curve (red) and the nominal curve (green) shown at successive zoom levels.

Figure 10.6: A zoom into the top edge of the filleted square. Here, the input curves coincide with the average (red) and thus, are not visible. However, the nominal shape (green) and the average are not coincident.

Fig. 10.6 is a zoom in on a region where the nominal shape has no curvature. We observe that there is a consistent deviation from the nominal shape for each manufacturing run. This suggests a consistent, systematic error. Two possible causes of this are:

- Incorrect machine calibration: The CNC machine is calibration once manufactured. The calibration process involves the one-time specification of certain fixed parameters for the manufactured machine. Errors in the calibration process can lead to systematic errors as well. For example, there could be an inaccuracy in the specification of the machine's 'home' position and, the spatial location realized when the machine is homed could, in reality, differ from that determined during calibration. Then, as the encoders are coupled with the machine's axis, the encoders will report measurements that are incorrectly referenced.
- Skipped motor steps: The CNC machine in our experiments uses micro-stepped stepper motors and, for a translational axes, a single step corresponds to 0.000012". A motor may skip a step due to load. This will lead to a difference in the reading recorded by the encoder coupled with the axis of the motor, and the expected encoder reading in the absence of a skip. As the CNC machine in our experiment runs open loop, an error introduced due to a skipped step will persist.

Incorrect machine model: We note another source of systematic error is that of an incorrect machine model being used by a CAM system. To enable a CAM system to transform spatial coordinates to a machine's joint (or configuration) space, a machine model is used. The machine model specifies the geometry of the machine and is used to determine the machine specific inverse kinematic (IK) transformation. An incorrectness in the machine's geometry specification will result in a (consistently) incorrect commanded joint position and thus a difference in the as-designed path and the average followed path. However, such an error will not be detected by our current system which transforms the encoder positions with the forward kinematic (FK) transformation obtained from the machine model.

In our setup, while both the FK and the IK are incorrect, they are still inverses of each other and thus, while the machined geometry may be incorrect as a result of an incorrect machine model, the geometry reconstructed from the encoder samples will not be. Thus, errors caused due to an incorrect machine model are not detected by our system, but, they can be detected if the geometry of the as manufactured shape is acquired directly using, for example, a coordinate measuring machine.

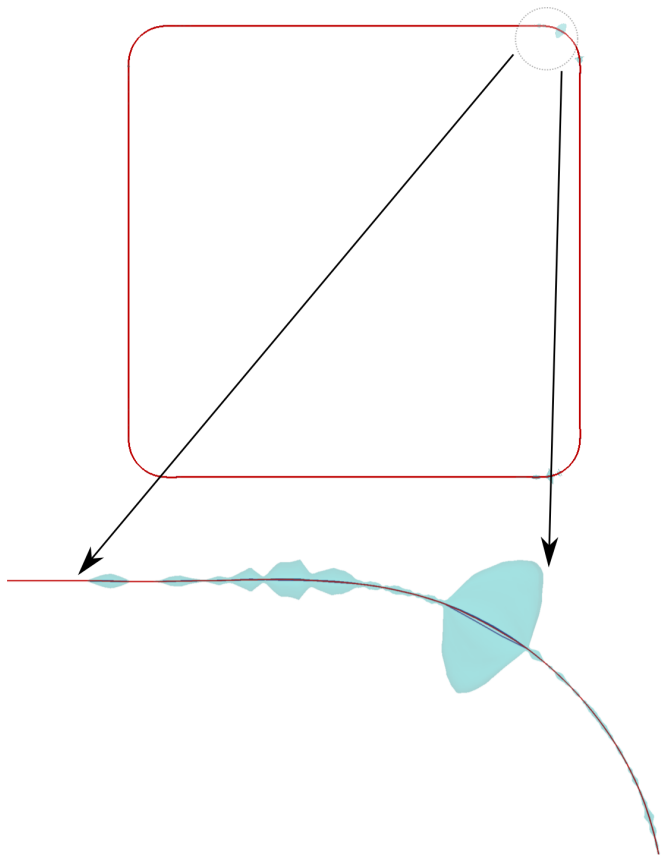

Figure 10.7: Top: The variability field visualized as a translucent tube whose radius is 40 times the computed variance, to magnify errors. Bottom: A high-variance region is shown zoomed-in.

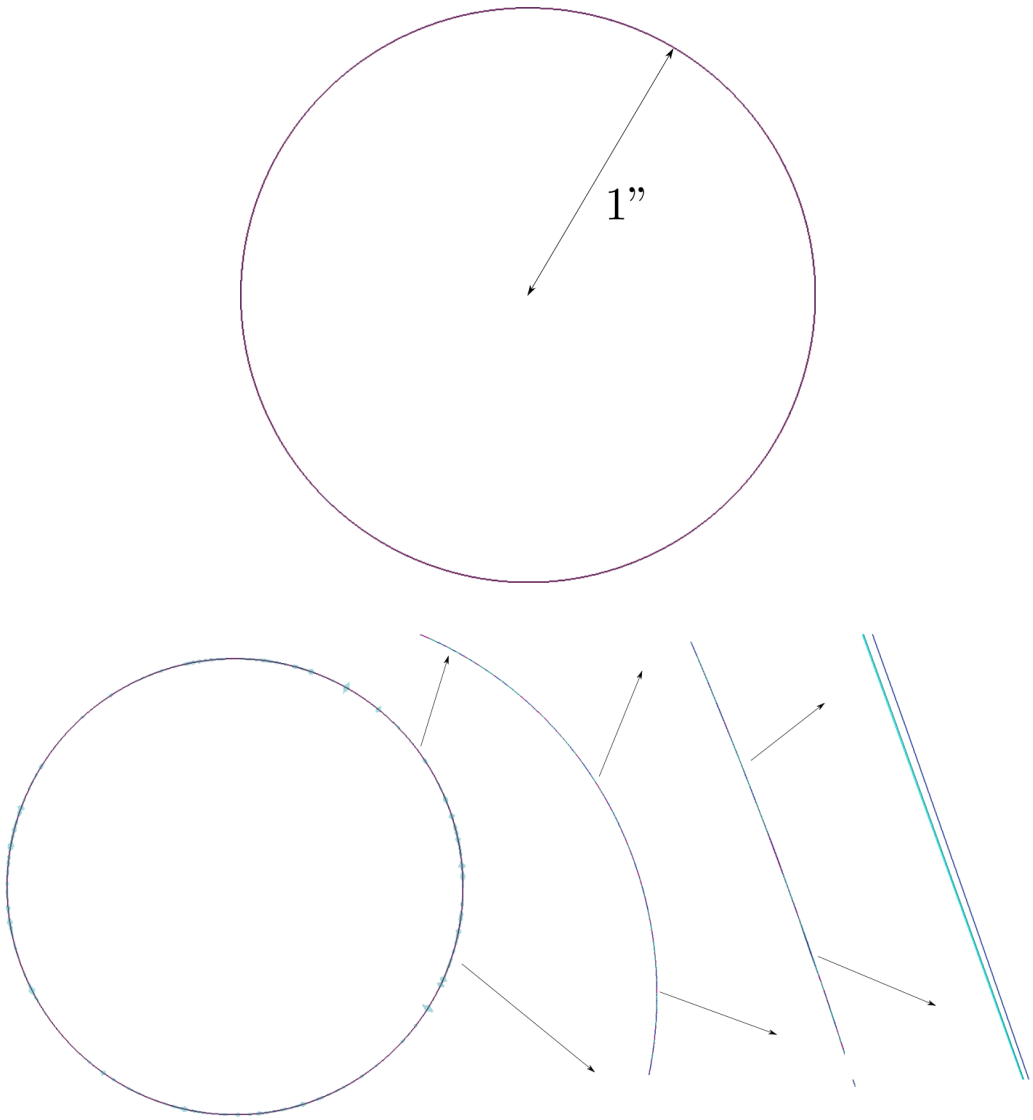

Figure 10.8: Top: A set of 3 encoder captured circular motions of the tool-path overlaid on each other. The tool-paths appear to be co-incident. Bottom: The variability tube scaled by a factor of 100 is visualized and used to reveal and guide exploration of regions of high variance in the captured paths.

We explore the use of the algorithms described in this thesis for computing the average path of the cutting tool. For the case when the cutting tool follows a path that is described by a small number of parameters and one desires the average path to belong to the same parametric family, we propose first computing the average and then computing the param-

eter values that yield the parameteric curve that best-fits the average. This approach is in contrast with two alternatives:

- *Averaging parameters:* For a set of tool paths, we: a) compute best-fit parameter values for each tool path, b) compute the average value of each parameter and c) compute the average tool path as the tool path corresponding to the average parameter value.

  For example, for a circular tool path, for each realization of the tool-path, a center c_i and radius r_i are approximated [124, 125], and the average circle is computed as the circle centered at the centroid of $\{c_i\}$ that has radius equal to the average of $\{r_i\}$.

  However, the drawback of averaging parameters is that the choice of the parameter space influences the result.
- *Direct fitting:* Instead of computing the average and then computing best-fit parameter values for the average, we directly compute parameter value functions for the best-fit parametric curve.

Use of the variability tube: We have samples of the variance field, and thus can focus on regions where there is a high variance. In Fig. 10.7, we display the variability field as a tube whose radius is 40 times the computed variance value. This highlights regions where there is a wider variation amongst the manufactured shapes.

In our experimental set-up, the chief reason for the variance of Fig 10.7 is that our acquisition process has low temporal resolution (we can sample from the encoders at a slow rate).

To partially remedy this issue, we changed the control algorithm to plan a trajectories at 20% of the prescribed kinematic limits. Executing a slow motion, we obtain a dense set of servo samples from our system. Using the lower limits, we plan a motion to move the cutting tool around the circle of Fig 10.8 and visualize the received encoder data. Fig. 10.8 (top) shows that it is difficult to make an informed guess on which region of the circle to view closely for viewing differences in the tool-paths. In Fig. 10.8 (bottom), we visualize

the variability tube scaled by a factor of 100. This reveals regions where the tool-paths have slight variations. We note that in the zoomed in region, one tool-path is a small distance away from the other two.

Both the average and the variability tube thus facilitate analysis of a tool-path. Being informed by the information represented by the average and variability tube, the CNC operator can, for example: a) attempt to root-cause consistent error sources b) modify tool-path geometry for improving conformance or, c) re-plan the motion for improving conformance.

CONCLUSION

In our thesis, we have proposed a characterization of the average of a set of shapes. We have provided the analytic solutions yielded by our characterization for the case of lines and planes. These solutions have been used in our developed algorithms for computing averages of a set of compatible, embedded shapes. The algorithms we have proposed are local and motivated by geometric intuition.

We have proposed an algorithmic approach 'Snap' that is grid-free, parallelizable (note that Trace is not parallelizable), simple and efficient. Snapping a set of seed samples yields a set of average samples, and, also yields sampled correspondences on each input shape and thus samples of the variability field.

We have proposed and studied a number of algorithmic modifications to core 'Snap' idea that extends its applicability. These modifications either modify the correspondence computation step of Snap, or the field evaluation step, or, both.

Additionally, we have presented a number of practical applications of our algorithms, including, sketch specification by overdrawing, manufacturing process analysis, geometric design, and collective re-sampling.

REFERENCES

[1] T. Kolsek, A. Jurca, and T. Vidic, "Recommendation system for sizing of children's footwear," *1st Int. Conf. on 3D Body Scanning Technologies, Proceedings*, pp. 126–131, 2010.

[2] "Shoe sizes - Mondopoint system of sizing and marking," Tech. Rep. International Standard ISO 9407, 1991.

[3] D. Omrcen and A. Jurca, "Shoe size recommendation system based on shoe inner dimension measurement," *2nd International Conference on 3D Body Scanning Technologies, Lugano, Switzerland*, pp. 158–163, 2011.

[4] J. Williams and J. Rossignac, "Tightening: Curvature-limiting morphological simplification," in *Proceedings of the 2005 ACM Symposium on Solid and Physical Modeling*, ser. SPM '05, Cambridge, Massachusetts, 2005, pp. 107–112, ISBN: 1-59593-015-9.

[5] M. B. Cohen, Y. T. Lee, G. Miller, J. Pachocki, and A. Sidford, "Geometric median in nearly linear time," in *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, ACM, 2016, pp. 9–21.

[6] S. R. Buss and J. P. Fillmore, "Spherical averages and applications to spherical splines and interpolation," *ACM Transactions on Graphics (TOG)*, vol. 20, no. 2, pp. 95–126, 2001.

[7] D. Eberly, R. Gardner, B. Morse, S. Pizer, and C. Scharlach, "Ridges for image analysis," *Journal of Mathematical Imaging and Vision*, vol. 4, no. 4, pp. 353–373, 1994.

[8] H. Karcher, "Riemannian center of mass and mollifier smoothing," *Communications on pure and applied mathematics*, vol. 30, no. 5, pp. 509–541, 1977.

[9] C. Icking and R. Klein, "Searching for the kernel of a polygon – a competitive strategy," in *Proceedings of the eleventh annual symposium on Computational geometry*, ACM, 1995, pp. 258–266.

[10] J. Kent, R. E. Parent, and W. E. Carlson, "Establishing correspondences by topological merging: A new approach to 3-D shape transformation," in *Proceedings of Graphics Interface*, vol. 91, 1991, pp. 271–278.

[11] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," California Univ. San Diego, La Jolla, Dept of Computer Science and Engineering, Tech. Rep., 2002.

[12] C. Xu, J. Liu, and X. Tang, "2D shape matching by contour flexibility," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 1, pp. 180–186, 2009.

[13] G. Mori, S. Belongie, and J. Malik, "Efficient shape matching using shape contexts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 11, pp. 1832–1837, 2005.

[14] D. Ghosh, A. Sharf, and N. Amenta, "Feature-driven deformation for dense correspondence," in *Medical Imaging 2009: Visualization, Image-Guided Procedures, and Modeling*, International Society for Optics and Photonics, vol. 7261, 2009, p. 726 136.

[15] M. O. Irfanoglu, B Gokberk, and L. Akarun, "3D shape-based face recognition using automatically registered facial surfaces," in *Proceedings of the 17th International Conference on Pattern Recognition (ICPR), 2004*, IEEE, vol. 4, 2004, pp. 183–186.

[16] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Doklady Akademii Nauk*, Russian Academy of Sciences, vol. 163, 1965, pp. 845–848.

[17] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *Journal of the ACM (JACM)*, vol. 21, no. 1, pp. 168–173, 1974.

[18] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series.," in *KDD workshop*, Seattle, WA, vol. 10, 1994, pp. 359–370.

[19] M. Maes, "On a cyclic string-to-string correction problem," *Information processing letters*, vol. 35, no. 2, pp. 73–78, 1990.

[20] V. Palazón-González and A. Marzal, "On the dynamic time warping of cyclic sequences for shape retrieval," *Image and Vision Computing*, vol. 30, no. 12, pp. 978–990, 2012.

[21] A. Kaul and J. Rossignac, "Solid-interpolating deformations: Construction and animation of PIPs.," *Computers & Graphics*, pp. 107–115, 1992.

[22] J. Rossignac and A. Kaul, "AGRELs and BIPs: Metamorphosis as a Bézier Curve in the space of polyhedra," *Computer Graphics Forum*, vol. 13, no. 3, pp. 179–184, 1994.

[23] S. Cohen, G. Elber, and R. Bar-Yehuda, "Matching of freeform curves," *Computer-Aided Design*, vol. 29, no. 5, pp. 369–378, 1997.

[24] J. Rossignac, "Ball-based shape processing," in *Discrete Geometry for Computer Imagery*, Springer, 2011, pp. 13–34.

[25] F. Chazal, A. Lieutier, and J. R. Rossignac, "Orthomap: Homeomorphism-guaranteeing normal-projection map between surfaces," Georgia Institute of Technology, Tech. Rep., 2004.

[26] E. A. Bier and K. R. Sloan, "Two-part texture mappings," *IEEE Computer Graphics and Applications*, vol. 6, no. 9, pp. 40–53, 1986.

[27] A. Asirvatham, E. Praun, and H. Hoppe, "Consistent spherical parameterization," in *International Conference on Computational Science*, Springer, 2005, pp. 265–272.

[28] D. Luebke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson, and R. Huebner, *Level of detail for 3D graphics*. Morgan Kaufmann, 2003.

[29] J. Schreiner, A. Asirvatham, E. Praun, and H. Hoppe, "Inter-surface mapping," in *ACM Transactions on Graphics (TOG)*, ACM, vol. 23, 2004, pp. 870–877.

[30] E. Praun, W. Sweldens, and P. Schröder, "Consistent mesh parameterizations," in *SIGGRAPH '01 Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, 2001, pp. 179–184.

[31] T. W. Sederberg and E. Greenwood, "A physically based approach to 2-D shape blending," in *ACM SIGGRAPH Computer Graphics*, ACM, vol. 26, 1992, pp. 25–34.

[32] David Eberly, *Distance from a point to an ellipse, an ellipsoid, or a hyperellipsoid*, Available at `https://www.geometrictools.com/Documentation/DistancePointEllipseEllipsoid.pdf`.

[33] R. Nürnberg, *Distance from a point to an ellipse*, Available at `http://wwwf.imperial.ac.uk/~rn/distance2ellipse.pdf`.

[34] T. Akenine-Möllser, "Fast 3D triangle-box overlap testing," *Journal of graphics tools*, vol. 6, no. 1, pp. 29–33, 2001.

[35] A. Rosenfeld and J. L. Pfaltz, "Sequential operations in digital picture processing," *Journal of the ACM (JACM)*, vol. 13, no. 4, pp. 471–494, 1966.

[36] ——, “Distance functions on digital pictures,” *Pattern recognition*, vol. 1, no. 1, pp. 33–61, 1968.

[37] G. Borgefors, “Distance transformations in digital images,” *Computer vision, graphics, and image processing*, vol. 34, no. 3, pp. 344–371, 1986.

[38] J. A. Sethian, “Fast marching methods,” *SIAM review*, vol. 41, no. 2, pp. 199–235, 1999.

[39] W. E. Lorensen and H. E. Cline, “Marching cubes: A high resolution 3d surface construction algorithm,” in *ACM siggraph computer graphics*, ACM, vol. 21, 1987, pp. 163–169.

[40] T. S. Newman and H. Yi, “A survey of the marching cubes algorithm,” *Computers & Graphics*, vol. 30, no. 5, pp. 854–879, 2006.

[41] T. Ju, F. Losasso, S. Schaefer, and J. Warren, “Dual contouring of hermite data,” in *ACM transactions on graphics (TOG)*, ACM, vol. 21, 2002, pp. 339–346.

[42] A. Cayley, “On contour and slope lines,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 18, no. 120, pp. 264–268, 1859.

[43] J. C. Maxwell, “On hills and dales,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 40, no. 269, pp. 421–427, 1870.

[44] J. W. Milnor, M. Spivak, R. Wells, and R. Wells, *Morse theory*. Princeton university press, 1963.

[45] L. Vincent and P. Soille, “Watersheds in digital spaces: An efficient algorithm based on immersion simulations,” *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 6, pp. 583–598, 1991.

[46] S Beucher, “The watershed transformation applied to image segmentation,” *Scanning Microscopy International*, pp. 299–314, 1992.

[47] C. Steger, “Subpixel-precise extraction of watersheds,” in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, IEEE, vol. 2, 1999, pp. 884–890.

[48] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

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

[50] R. Peikert and M. Roth, "The "parallel vectors" operator: A vector field visualization primitive," in *Proceedings of the conference on Visualization'99: Celebrating ten years*, IEEE Computer Society Press, 1999, pp. 263–270.

[51] C. Werner, "Formal analysis of ridge and channel patterns in maturely eroded terrain," *Annals of the Association of American Geographers*, vol. 78, no. 2, pp. 253–270, 1988.

[52] J. J. Koenderink and A. J. van Doorn, "Local features of smooth shapes: Ridges and courses," in *Geometric methods in computer vision II*, International Society for Optics and Photonics, vol. 2031, 1993, pp. 2–14.

[53] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, and A. Santella, "Suggestive contours for conveying shape," *ACM Transactions on Graphics (TOG)*, vol. 22, no. 3, pp. 848–855, 2003.

[54] S. Musuvathy, E. Cohen, J. Damon, and J.-K. Seong, "Principal curvature ridges and geometrically salient regions of parametric B-spline surfaces," *Computer-Aided Design*, vol. 43, no. 7, pp. 756–770, 2011.

[55] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International journal of computer vision*, vol. 1, no. 4, pp. 321–331, 1988.

[56] J. A. Sethian, *Level set methods and fast marching methods: Evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*. Cambridge university press, 1999, vol. 3.

[57] M. Rumpf and B. Wirth, "A nonlinear elastic shape averaging approach," *SIAM Journal on Imaging Sciences*, vol. 2, no. 3, pp. 800–833, 2009.

[58] H. Blum, "Biological shape and visual science Part I," *Journal of theoretical Biology*, vol. 38, no. 2, pp. 205–287, 1973.

[59] K. Siddiqi and S. Pizer, *Medial representations: Mathematics, algorithms and applications*. Springer Science & Business Media, 2008, vol. 37.

[60] L. Zhang and D. Manocha, "An efficient retraction-based RRT planner," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, IEEE, 2008, pp. 3743–3750.

[61] D. Attali, J.-D. Boissonnat, and H. Edelsbrunner, "Stability and computation of medial axes: A state-of-the-art report," in *Mathematical foundations of scientific*

visualization, computer graphics, and massive data exploration, Springer, 2009, pp. 109–125.

[62] D.-T. Lee, "Medial axis transformation of a planar shape," *IEEE Transactions on pattern analysis and machine intelligence*, no. 4, pp. 363–369, 1982.

[63] A. Lieutier, "Any open bounded subset of $\mathbb{R}^n$ has the same homotopy type as its medial axis," *Computer-Aided Design*, vol. 36, no. 11, pp. 1029–1046, 2004.

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

[65] A. Sud, M. Foskey, and D. Manocha, "Homotopy-preserving medial axis simplification," in *Proceedings of the 2005 ACM symposium on Solid and physical modeling*, ACM, 2005, pp. 39–50.

[66] F. Chazal, A. Lieutier, J. R. Rossignac, and B. Whited, "Ball-map: Homeomorphism between compatible surfaces," Georgia Institute of Technology, Tech. Rep., 2006.

[67] B. Whited and J. Rossignac, "Ball-morph: Definition, implementation, and comparative evaluation," *IEEE transactions on visualization and computer graphics*, vol. 17, no. 6, pp. 757–769, 2011.

[68] J.-M. Lien, S. L. Thomas, and N. M. Amato, "A general framework for sampling on the medial axis of the free space," in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, IEEE, vol. 3, 2003, pp. 4439–4444.

[69] J. Ma, S. W. Bae, and S. Choi, "3D medial axis point approximation using nearest neighbors and the normal field," *The Visual Computer*, vol. 28, no. 1, pp. 7–19, 2012.

[70] S. Lloyd, "Least squares quantization in PCM," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[71] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[72] P. Lancaster and K. Salkauskas, "Surfaces generated by moving least squares methods," *Mathematics of computation*, vol. 37, no. 155, pp. 141–158, 1981.

[73] D. Levin, "The approximation power of moving least-squares," *Mathematics of Computation of the American Mathematical Society*, vol. 67, no. 224, pp. 1517–1531, 1998.

[74] T. Hastie and W. Stuetzle, "Principal curves," *Journal of the American Statistical Association*, vol. 84, no. 406, pp. 502–516, 1989.

[75] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Sensor Fusion IV: Control Paradigms and Data Structures*, International Society for Optics and Photonics, vol. 1611, 1992, pp. 586–607.

[76] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," *Image and vision computing*, vol. 10, no. 3, pp. 145–155, 1992.

[77] H. Pottmann and M. Hofer, "Geometry of the squared distance function to curves and surfaces," in *Visualization and mathematics III*, Springer, 2003, pp. 221–242.

[78] N. Amenta, M. Bern, and D. Eppstein, "The crust and the β-skeleton: Combinatorial curve reconstruction," *Graphical models and image processing*, vol. 60, no. 2, pp. 125–135, 1998.

[79] N. Amenta, M. Bern, and M. Kamvysselis, "A new voronoi-based surface reconstruction algorithm," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM, 1998, pp. 415–421.

[80] F. Aurenhammer, "Voronoi diagrams – a survey of a fundamental geometric data structure," *ACM Computing Surveys (CSUR)*, vol. 23, no. 3, pp. 345–405, 1991.

[81] H. Edelsbrunner and N. R. Shah, "Triangulating topological spaces," *International Journal of Computational Geometry & Applications*, vol. 7, no. 04, pp. 365–378, 1997.

[82] R. Goldman, *Pyramid algorithms: A dynamic programming approach to curves and surfaces for geometric modeling*. Morgan Kaufmann, 2002.

[83] M. Ovsjanikov, M. Ben-Chen, J. Solomon, A. Butscher, and L. Guibas, "Functional maps: A flexible representation of maps between shapes," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, p. 30, 2012.

[84] M. Rabinovich, R. Poranne, D. Panozzo, and O. Sorkine-Hornung, "Scalable locally injective mappings," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 2, p. 16, 2017.

[85] R. B. Tilove, "Set membership classification: A unified approach to geometric intersection problems," *Computers, IEEE Transactions on*, vol. 100, no. 10, pp. 874–883, 1980.

[86] F. Chazal, A. Lieutier, and J. Rossignac, "Projection-homeomorphic surfaces," in *Proceedings of the 2005 ACM symposium on Solid and Physical Modeling*, ACM, 2005, pp. 9–14.

[87] N. Amenta and M. Bern, "Surface reconstruction by voronoi filtering," *Discrete & Computational Geometry*, vol. 22, no. 4, pp. 481–504, 1999.

[88] M. J. Atallah, "A linear time algorithm for the hausdorff distance between convex polygons," *Information processing letters*, vol. 17, no. 4, pp. 207–209, 1983.

[89] H. Alt, B. Behrends, and J. Blömer, "Approximate matching of polygonal shapes," *Annals of Mathematics and Artificial Intelligence*, vol. 13, no. 3-4, pp. 251–265, 1995.

[90] H. Alt, P. Braß, M. Godau, C. Knauer, and C. Wenk, "Computing the hausdorff distance of geometric patterns and shapes," in *Discrete and computational geometry*, Springer, 2003, pp. 65–76.

[91] M. Bartoň, I. Hanniel, G. Elber, and M.-S. Kim, "Precise hausdorff distance computation between polygonal meshes," *Computer Aided Geometric Design*, vol. 27, no. 8, pp. 580–591, 2010.

[92] L. Scharf, "Computing the hausdorff distance between sets of curves," Master's thesis, Institut für Informatik, Freie Universität Berlin, 2003.

[93] P. Cignoni, C. Rocchini, and R. Scopigno, "Metro: Measuring error on simplified surfaces," in *Computer Graphics Forum*, Wiley Online Library, vol. 17, 1998, pp. 167–174.

[94] T. Culver, J. Keyser, and D. Manocha, "Exact computation of the medial axis of a polyhedron," *Computer Aided Geometric Design*, vol. 21, no. 1, pp. 65–98, 2004.

[95] C. M. Hoffmann, "How to construct the skeleton of csg objects," *The Mathematics of Surfaces. IVA. Bowyer and J. Davenport, Eds., Oxford University Press*, 1990.

[96] O. Aichholzer, W. Aigner, F. Aurenhammer, T. Hackl, B. Jüttler, and M. Rabl, "Medial axis computation for planar free–form shapes," *Computer-Aided Design*, vol. 41, no. 5, pp. 339–349, 2009.

[97] N. Amenta, S. Choi, and R. K. Kolluri, "The power crust, unions of balls, and the medial axis transform," *Computational Geometry*, vol. 19, no. 2-3, pp. 127–153, 2001.

[98] T. K. Dey and W. Zhao, "Approximate medial axis as a voronoi subcomplex," in *Proceedings of the seventh ACM symposium on Solid modeling and applications*, ACM, 2002, pp. 356–366.

[99] S. Fortune, "Voronoi diagrams and delaunay triangulations," in *Computing in Euclidean geometry*, World Scientific, 1995, pp. 225–265.

[100] L. Guibas and J. Stolfi, "Primitives for the manipulation of general subdivisions and the computation of voronoi," *ACM transactions on graphics (TOG)*, vol. 4, no. 2, pp. 74–123, 1985.

[101] B. Whited and J. Rossignac, "Relative blending," *Computer-Aided Design*, vol. 41, no. 6, pp. 456–462, 2009.

[102] H. Nguyen, *GPU Gems 3*. Addison-Wesley Professional, 2007.

[103] C. Sigg, R. Peikert, and M. Gross, "Signed distance transform using graphics hardware," in *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, IEEE Computer Society, 2003, p. 12.

[104] A. Krishnamurthy, S. McMains, and K. Haller, "GPU-accelerated minimum distance and clearance queries," *IEEE transactions on visualization and computer graphics*, vol. 17, no. 6, pp. 729–742, 2011.

[105] M. Attene, B. Falcidieno, M. Spagnuolo, and J. Rossignac, "Swingwrapper: Retiling triangle meshes for better edgebreaker compression," *ACM Transactions on Graphics (TOG)*, vol. 22, no. 4, pp. 982–996, 2003.

[106] G. Turk, "Re-tiling polygonal surfaces," in *ACM SIGGRAPH Computer Graphics*, ACM, vol. 26, 1992, pp. 55–64.

[107] G. Turk and M. Levoy, "Zippered polygon meshes from range images," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM, 1994, pp. 311–318.

[108] M. Sati and J. Rossignac, "Average and variance of a quasi-parallel family of surfaces," *Computer-Aided Design*, vol. 102, pp. 61–71, 2018.

[109] CGAL, *Computational Geometry Algorithms Library*, http://www.cgal.org.

[110] M. Campen and L. Kobbelt, "Exact and robust (self-) intersections for polygonal meshes," in *Computer Graphics Forum*, Wiley Online Library, vol. 29, 2010, pp. 397–406.

[111] H. Edelsbrunner and N. R. Shah, "Triangulating topological spaces," in *Proceedings of the tenth annual symposium on Computational geometry*, ACM, 1994, pp. 285–292.

[112] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "Meshlab: an open-source mesh processing tool," in *Eurographics Italian Chapter Conference*, V. Scarano, R. D. Chiara, and U. Erra, Eds., The Eurographics Association, 2008.

[113] P. Alliez, S. Tayeb, and C. Wormser, "3D fast intersection and distance computation (AABB Tree)," in *CGAL User and Reference Manual*, 4.3, CGAL Editorial Board, 2013.

[114] The CGAL Project, *CGAL User and Reference Manual*, 4.13. CGAL Editorial Board, 2018.

[115] G. Turk and J. F. O'Brien, "Shape transformation using variational implicit functions," in *SIGGRAPH' 99 Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, 1999, pp. 335–342.

[116] C. Liu, E. Rosales, and A. Sheffer, "Strokeaggregator: Consolidating raw sketches into artist-intended curve drawings," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, p. 97, 2018.

[117] R. Lynn, A. Chen, S. Locks, C. Nath, and T. Kurfess, "Intelligent and accessible data flow architectures for manufacturing system optimization," in *IFIP International Conference on Advances in Production Management Systems*, Springer, 2015, pp. 27–35.

[118] R. Lynn, M. Sati, T. Tucker, J. Rossignac, C. Saldana, and T. Kurfess, "Realization of the 5-axis machine tool digital twin using direct servo control from cam," *National Institute of Standards and Technology (NIST) Model-Based Enterprise Summit*, 2018.

[119] Tucker Innovations Inc, *Sculptprint - The subtractive 3D printing application*, Available at `www.sculptprint3d.com`.

[120] *Machinekit integrator manual*.

[121] J. Albus and R. Lumia, "The enhanced machine controller (emc): an open architecture controller for machine tools," *Journal of Manufacturing Review*, vol. Vol. 7, no. No. 3, pp. 278–280, 1994.

[122] M. M. Hossain, “Voxel-based offsetting at high resolution with tunable speed and precision using hybrid dynamic trees,” PhD thesis, Georgia Institute of Technology, 2016.

[123] H. Pham and Q.-C. Pham, “A new approach to time-optimal path parameterization based on reachability analysis,” *IEEE Transactions on Robotics*, 2018.

[124] D. Umbach and K. N. Jones, “A few methods for fitting circles to data,” *IEEE Transactions on instrumentation and measurement*, vol. 52, no. 6, pp. 1881–1885, 2003.

[125] H. Ma, “Geometric fitting of quadratic curves and surfaces,” PhD thesis, The University of Alabama at Birmingham, 2012.