A COMBINED SKELETON MODEL

A Thesis

by

DANIEL EVAN MILLER

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	John Keyser
Committee Members,	Yoonsuck Choe
	Scott Schaefer
	Louise Abbott
Head of Department,	Nancy Amato

December 2014

Major Subject: Computer Science and Engineering

ABSTRACT

Skeleton representations are a fundamental way of representing a variety of solid models. They are particularly important for representing certain biological models and are often key to visualizing such data. Several methods exist for extracting skeletal models from 3D data sets. Unfortunately, there is usually not a single correct definition for what makes a good skeleton, and different methods will produce different skeletal models from a given input. Furthermore, for many scanned data sets, there also is inherent noise and loss of data in the scanning process that can reduce ability to identify a skeleton.

In this document, I propose a method for combining multiple algorithms' skeleton results into a single composite skeletal model. This model leverages various aspects of the geometric and topological information contained in the different input skeletal models to form a single result that may limit the error introduced by particular inputs by means of a confidence function. Using such an uncertainty based model, one can better understand, refine, and de-noise/simplify the skeletal structure. The following pages describe methods for forming this composite model and also examples of applying it to some real-world data sets.

DEDICATION

To my newly born nephew Jayden Xavier Okubadejo:

May your days always be filled with the light of the crucified Savior, even as we your family show you but a pale reflection of his perfect love. I look forward to watching you grow and I hope I am a good uncle to you.

ACKNOWLEDGEMENTS

I would like to thank my adviser, Dr. John Keyser, and my comittee, Dr. Louise Abbott, Dr. Yoonsuck Choe, and Dr. Scott Schaefer, for their support and guidance. This thesis was funded in part by the National Science Foundation, grants #1208174 and #1256086 (PI: Yoonsuck Choe).

NOMENCLATURE

BNL	Brain Networks Lab
KESM	Knife Edge Scanning Microscope
Medial Axis	Set of all points having more than one closest point on the object's boundary
Medial Axis Skeleton	In 2D a set tree-like branching points making up the object's medial axes, while in 3D this often is composed of planes as well as line segments and curves
Endpoint	Point at end of polyline not connected with another polyline
Midpoint	Point along polyline with a valence of two
Branch Point	Point on polyline with a valence greater than two (connecting multiple polylines)
Edge	Line segment of two points connected by minimu spanning line
Segment	Sequential series of one or more edges
Skeleton	Set of segments connected at branch points (typically used to represent and manipulate a volume or mesh in simplified form)
Segment Set	Set of segments that logically correspond to a single underlying segment
Set Model	Set of endpoint connected segment sets logically corresponding to an underlying skeleton
Result Model	Set of segments connected at branch points with points containing combination data (uncertainty evaluation)

TABLE OF CONTENTS

Page

ABSTRACT ii

DEDICATION iii

ACKNOWLEDGEMENTS iv

NOMENCLATURE v

TABLE OF CONTENTS vi

LIST OF FIGURES viii

1. INTRODUCTION AND MOTIVATION 1

    1.1 Problem Formulation 2
    1.2 Ideal Model 5

2. BACKGROUND AND PRIOR WORK 7

3. THEORY AND METHOD 10

    3.1 Definitions 10
        3.1.1 Skeletal Model 10
        3.1.2 Set Model 12
        3.1.3 Result Model 13
    3.2 Method Overview 14
    3.3 Input Initialization 15
    3.4 Combination 17
    3.5 Segment Comparison 25
    3.6 Distance Mapping 28
    3.7 Composing the Set Model 29
        3.7.1 Mapping Rules 29
        3.7.2 Maintaining Order Independence (Merge and Split) 39
    3.8 Forming the Result Model 42
        3.8.1 Set Evaluation 43
        3.8.2 Confidence Value 43
        3.8.3 Connection Repair 44

4. RESULTS AND EVALUATION 46

4.1 Combining Different Volume Skeletons 46
4.2 Combining Adjacent Volume Skeletons 46
4.3 Combining Mesh Skeletons 47

5. DISCUSSION AND CONCLUSION 54

5.1 Queries 54
5.2 Discussion 55
5.3 Revisiting the Ideal Combined Model 56
5.4 Conclusion 57
5.5 Future Work 57

REFERENCES 59

LIST OF FIGURES

FIGURE Page

1.1 (Left) Medial axis representation of a simple horse outline (generated following [1]) (Right) Simplified skeleton representation of a humanoid mesh used in character animation(generated following [2]) 2

1.2 Process of taking a physical volume V to a combined skeleton S' that better accounts for what is found through the scan V' of the volume than any single traced skeleton S_i 4

1.3 (Left) Volume data showing vasculature from a mouse brain (Right) A skeleton extracted from that data using an automated trace [6] . . 5

3.1 Simple edge example, with endpoints containing data such as radius. 10

3.2 Simple segment example, with pointers to other connecting segments at head and tail. 11

3.3 Simple skeleton example, which is composed of a list of its composite segments. 12

3.4 The set model is represented by a series of individual sets (color dictates single set) that each correspond to single final segments in the result model. Within each set is a list of segments that are used to form this final segment for the region, where the region is denoted by branch points or farthest extents of the list of segments. 13

3.5 The result model is a skeleton that also contains information about the combined consensus at each of its points. 14

3.6 (Left)Sample input skeleton. (Right)Marked non-midpoints in white added to PL. 15

3.7 Walk/Mark along each unmarked path until another element in PL is located. Red is a completed segment and yellow is being traced. . 16

3.8 Mark each topological connection among the segments (contains the same endpoint at head or tail). 16

3.9 The left figure corresponds to using the summed minimun pair distance between endpoints to get a basis for orientation. The right figure is an illustration of the result of the combination of multiple line segments (dotted lines) to a segment containing all their endpoints. 18

3.10 An example of the PCA of a set of points in space, with the green arrow representing the principal axis (generated following [13]). . . . 20

3.11 Points along each segment in the set are taken at regular intervals (top left). Using these points, a best fit polynomial is constructed using least squares optimization using PCA parameterization of points(top right). Finally, the parameter of the closest point on the polynomial is used as a basis for ordering the segments in a global manner with respect to the set (bottom). 21

3.12 (Left) Parameterized endpoints based on polynomial. (Right) Monotonic parameterization of midpoints based edge length. 22

3.13 Averaged points resulting from averaging segments at each change along interval. 23

3.14 (Left) Average points composed into a segment. (Right) Segment resampled to achieve smoothness. 23

3.15 An example of the least squares polynomial fitting of a set of points in space (generated following [16]). 25

3.16 The one-sided distance is computed by averaging the distances to one segment from all of the other segment's edgepoints. The minimum average distance is the smaller of the two one-sided average distances. 27

3.17 Cases where no match is detected and a new set is created include when the comparison score too far and when the closest distance between the segments is too great. 30

3.18 Cases where combining is detected and the set can safely and logically add the combining segment: (Top) overlapping non-branching endpoint, (Right) subsumed segment contains non-branching endpoints, (Bottom) match case. 32

3.19 Cases where splitting is detected and the set must cut the combining segment (yellow) due to the valence of the result segment (green): (Top) overlapping branching endpoint, (Right) subsumed segment contains one branching endpoints, (Bottom) subsumed segment contains both branching endpoints. 33

3.20 Cases where swapping is detected and the set must switch out the set due to the valence of the combining segment (yellow) and the valence of the result segment (green): (Top) overlapping branching endpoint, (Right) subsumed segment contains one branching endpoints, (Bottom) subsumed segment contains both branching endpoints. 34

3.21 Cases where retesting is detected and the criteria for matching is increased slightly before testing again: (Top) same endpoints mapped but no segment's endpoint pair is mapped, (Right) only one non-branching endpoint mapped, (Left) no endpoint is mapped. 35

3.22 The flow of the decision tree for deciding how a segment X will be combined with the Set Model given a distance threshold m, score threshold s, and point tolerance threshold p (used for detecting regions too small). 37

3.23 The second section of the decision tree for deciding how a segment X will be combined with the Set Model. 38

3.24 On the left: If segments are combined in monotonic fashion with regards to their numbering, the case on the left results in two segment sets in the the combined skeletal model even though adding them in the order 1,3,5,4,2 results in a single segment set. By merging sets we resolve this issue by ensuring all segments that would map together are in the same set. On the right: With segments continuing to map together and also their sets undergoing merging, sometimes the range of variability for an individual set grows beyond the desired bounds (say for a radius measure). By subdividing the set into two with regards to the farthest matches we then ensure that at each level of subdivision the set is rid of the largest outlier from its current median. 41

3.25 Combining synthetic skeletons from a model. Three different skeletons are combined into a single unified model. At top left are three skeletons, each in a separate color. At top-right is the combined skeletal model, with each segment set highlighted in a different color. At bottom-left is the composite result, with each segment highlighted in a different color. At bottom-right is the composite skeleton with uncertainty values color ranging from less certain (blue) to more certain (red). 42

3.26 If the endpoints of mulltiple result segments match together topologically, geometric connectivity is ensured by setting their endpoints to be the average of the of the mapped endpoints. 44

4.1 Combining synthetic skeletons from a model. Three different skeletons are combined into a single unified model. At top left are three skeletons, each in a separate color. At top-right is the combined skeletal model, with each segment set highlighted in a different color. At bottom-left is the composite result, with each segment highlighted in a different color. At bottom-right is the composite skeleton with uncertainty values color ranging from less certain (blue) to more certain (red). 47

4.2 Combining skeletons from two different algorithms [11, 6], and several parameter settings for each. The skeletons trace the vascular structure from part of the mouse Cerebellum. At top-left are the individual skeletons, each in a different color. At top-right is the set model, with each segment set highlighted in a different color. At bottom-left is the result model with each segment a different color. At bottom-right is the result model colored based on uncertainty, ranging from less certain (blue) to more certain (red). 48

4.3 Combining 8 adjacent volumes' skeletons from different traces using the same algorithm [6] over overlapping volume regions. The image displays the combining segments with color denoting no combination (blue) or combination (red). For this case, each volume is 550^3 in space with 100 overlap, thus allowing us to piece together a skeleton for a 1000^3 volume we cannot trace normally in memory. 49

4.4 Combining 3 skeletons from different traces using the same algorithm [6] over overlapping volume regions. The skeletons trace the vascular structure from part of the mouse cerebellum, and the same algorithm (with the same parameter settings) is used for each of the three volumes. The top image displays the skeletons (denoted by individual colors), and the bottom image displays the result model with color denoting uncertainty ranging from less certain (blue) to more certain (red). In this case, the overlapping region is more certain, while the less overlapped area is less certain. 50

4.5 Combining 6 skeletons from different traces using the same algorithm [6] over overlapping volume regions. The skeletons trace the vascular structure from part of the mouse cerebellum, and the same algorithm (with the same parameter settings) is used for each of the three volumes. The top image displays the skeletons (denoted by individual colors), and the bottom image displays the result model with color denoting uncertainty ranging from less certain (blue) to more certain (red). In this case, the overlapping region is more certain, while the less overlapped area is less certain. 51

4.6 Combining 2 skeletons generated by [6] over subsets of a larger volume of the vascular structure of the mouse cerebellum. The top left image displays the input skeletons (denoted by individual colors), and the top right image displays the set model with color denoting individual sets. The bottom left displays the uncertainty ranging from less certain (blue) to more certain (red) and the bottom right displays the final result (pink) with a trace over the entire volume (orange). 52

4.7 Combining skeletons from the armadilloman mesh model. Three different skeletons are combined into a single unified model. At top left are three skeletons, each in a separate color. At top right is the combined skeleton; the colors demonstrate different segments in the combined model. At bottom is the combined skeleton overlaid on the model. 53

1. INTRODUCTION AND MOTIVATION

Skeletal models are prevalent in a number of applications, including biology and graphics. The relatively simple structure of a skeletal model allows for an intuitive way to describe the shape of the data, as well as provides a means to simplify computations for the shape. Skeletons are used to deal with polygonal models and with volumetric data, and typically provide a much more compact representation of the object while losing some fidelity. In particular, skeletal structures form an important aspect of several biological visualizations, in that they often allow for much more rapid and less memory-intensive rendering than the volumetric or similarly highly sampled "raw" data. While an the exact mathematical representation of the skeleton of an object is commonly understood to be the true medial axis, most skeletal models are not in fact faithful to this but are conceptually related and serve as a substitute for medial axes in visualization or computation. An example of this can be seen in the following Fig.1.1.

Unfortunately, there is no consensus about what a "good" skeleton of a given model or data set should be. This discrepancy of the ideal skeleton for a data set arises because the desired result depends both on the original data and the desired use or purpose the skeleton will fulfill. There are numerous skeletonization algorithms, offering different advantages and disadvantages, and producing different skeletons given the same input model. Furthermore, the same skeletonization algorithm will often produce varying results of a data set for different sets of initialization parameters for the skeletonization algorithm. As a result, even for the same data set, there can be many possible skeletons generated. To further complicate the problem of skeletonization for volumes, most volume based algorithms require the volume to be

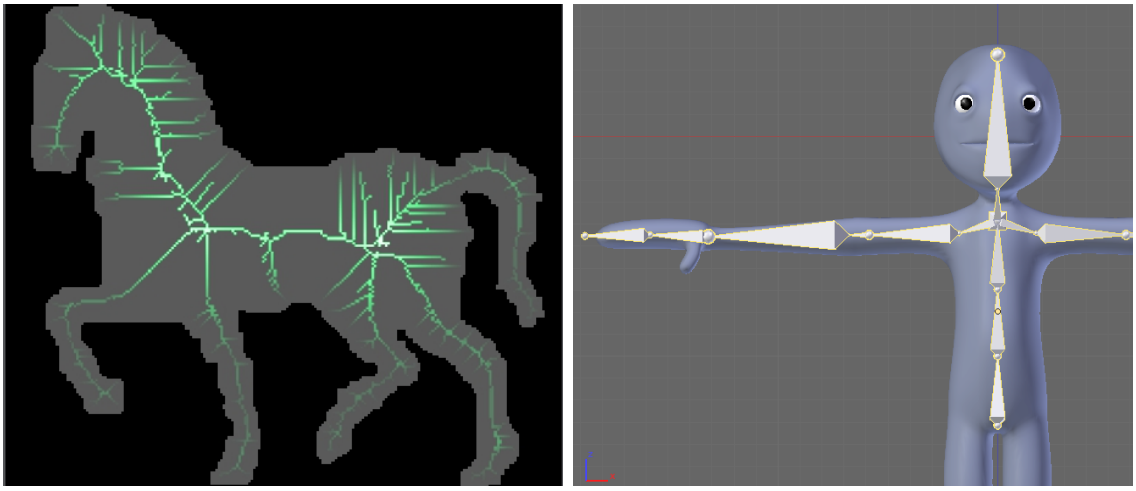

Figure 1.1: (Left) Medial axis representation of a simple horse outline (generated following [1]) (Right) Simplified skeleton representation of a humanoid mesh used in character animation(generated following [2])

small enough to fit in memory, something that is not always feasible for high resolution biological data sets. Our work here aims to deal with such situations as these by creating a unified result skeleton that incorporates the differing skeletons from several algorithms into a single robust result skeleton.

1.1 Problem Formulation

A major motivating example for us has been skeletonization of volumetric data of tube-like structures (particularly vasculature). In these settings, a 3D volume of data (512^3 voxels is typical) is "traced" by an algorithm, producing one or more disjoint skeletal structures. At the highest resolution with our equipment, this volume is effectively about 0.5^3 mm of data from a 15x9x6 mm block of embedded mouse brain (about 1-2 Tb of total storage). These volumes are generated from anatomical imaging using the Knife Edge Scanning Microscope (*KESM*), with the common flow being: specimen preparation, serial sectioning, image acquisition, image cleaning, and finally composition of the images into a volume for tracing or fitting geometric

primitives to the data. In this process there are various elements of noise and error that can accumulate and propagate to the tracing stage, such as blood clots (preventing stain), knife chatter (increasing erroneous traces), lighting artifacts (noise) or bad images (data gaps).

Specifically, when given a physical volume V, we section, image, clean, and compose the data to arrive at the digital volume V'. A variety of algorithms, often specific to the particular scanning procedure, have been developed for dealing with the various errors introduced in the transformation from V to V' and we will ignore this aspect of the process, though it should be noted that some tracing/segmentation algorithms also attempt to account for these propagated errors.

We will assume, then, that we are given a scanned volume, V', that contains data (such as vasculature or neuron structures) that is amenable to a skeletal representation. From that, we wish to extract the "best" skeleton from that volume. Many of the existing tracing methods possess a variety of parameters that can be tuned, producing a wide variety of different (but hopefully similar) skeletons. We can use some tracing algorithm T_i with parameters p_j to generate a skeleton S_k. See Fig.1.3. Given that any method T_i with p_j potentially introduces some error ϵ_k, our goal is to limit the contribution of any individual ϵ_k by combining the various S_ks to get the result skeleton S'. By using this combination of skeletons, we strive to achieve an S' that will be a better approximation of V' (and thus of V) than any individual S_k (see Fig.1.2). Our working assumption for this combination is that most of the S_ks do not have the same placement of error e_k and the majority of S_ks remain fairly reliable, or the error e_k is independent, nonuniform, and does not compose a large portion of the skeleton.

A second motivating example occurs when the underlying data is much larger than can be traced in memory at one time; our typical approach is to divide the

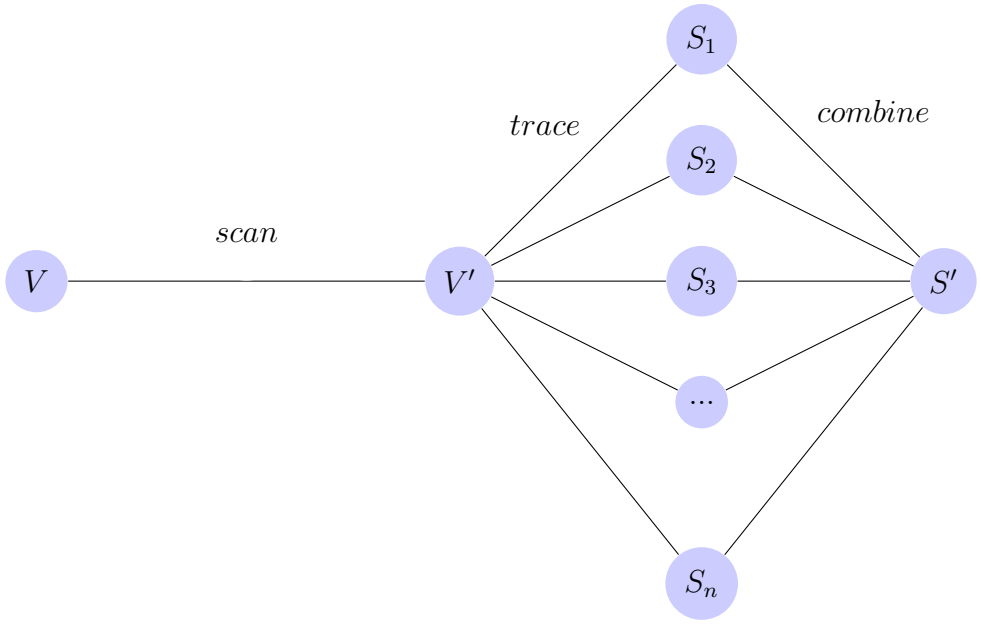

Figure 1.2: Process of taking a physical volume V to a combined skeleton S' that better accounts for what is found through the scan V' of the volume than any single traced skeleton S_i

overall data volume into overlapping blocks, trace each one, and then merge the results together. Overlapping blocks are used instead of trying to match endpoints of adjacent datasets due to the predictive nature of several local volume tracing algorithms and the relative unreliability of the algorithms near volume boundaries ([6], [11]).

To state the problem more generally, we wish to take a sequence of skeleton models representing the same structure to produce a consensus result skeleton. These skeleton models may not fully span the underlying structure individually, but their desired combination should yield model that spans as much as the input models allow. With this combination, we want a measure of the level of consensus based on aspects of similarity and correspondence between the skeleton inputs. This consensus should be based on a similarity function designed to measure geometric and topological

property proximity across the skeleton components.

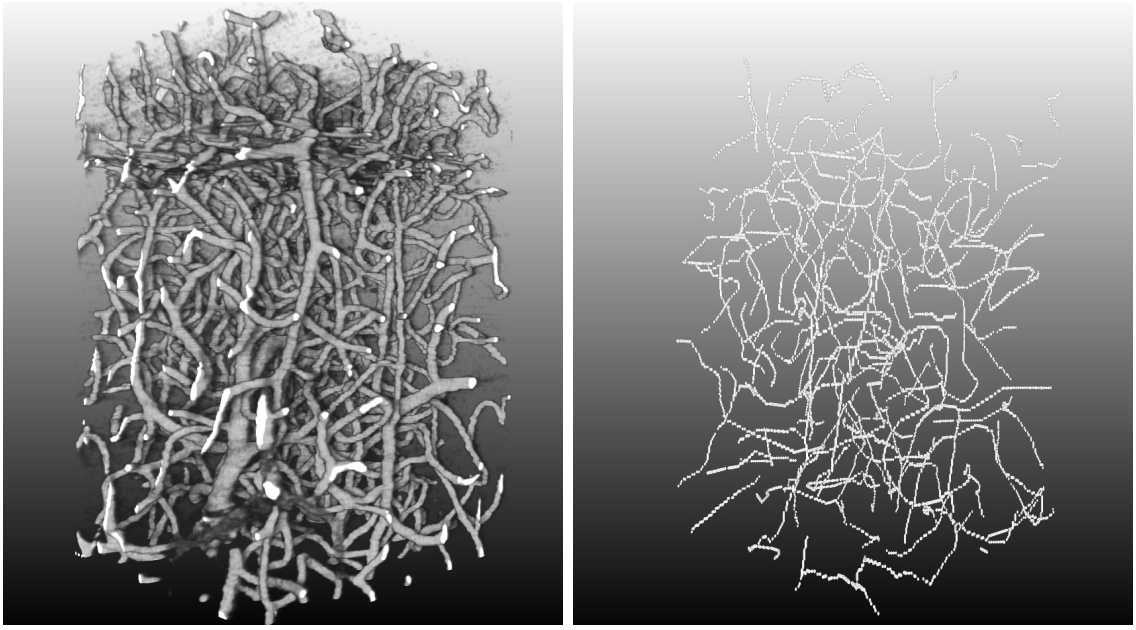

Figure 1.3: (Left) Volume data showing vasculature from a mouse brain (Right) A skeleton extracted from that data using an automated trace [6]

1.2 Ideal Model

Given the problem of combining various skeletons in an error reducing fashion, some key components of a result model should be:

- *Compact accessible representation*: Volumes (and thus the skeletons) can span very large volumes, and one of the main reasons to use skeletal representations is for data reduction. Thus, the representation should be compact while allowing access queries regarding subsections in a reasonable time.

- *Consensus function*: Since the result is a composite of the inputs, details of the confidence in various regions of the result should be available. Information

such as the amount of overlap and the standard deviation of inputs that map to that part of the result are important to understanding the error bound. This consensus function can help us understand where there is greater uncertainty, what parts are outliers, and where additional refinement is needed.

- *Order independence*: There may be many different inputs and an exponential number of possible orderings of the input. We desire a single robust output model that is independent of the ordering of the input skeletons.

- *Weighting of inputs*: Though almost all trace algorithms for our primary motivating example produce errors, some can be more reliable for specific volumes than others. Also, in the case where we have a majority of inputs from one trace (with different parameters) and only a few from other traces, we would like to be able to scale the inputs in inverse proportion to their trace frequency.

In this document, we introduce a combined skeletal model designed for such situations. We describe both the structure of the combined model, along with methods for combining multiple skeletons. Finally, we conclude with a brief discussion of how this model can be used for further querying, and present several results on real data sets.

2. BACKGROUND AND PRIOR WORK

Skeleton extraction is an important part of several different applications. A complete overview of methods for skeleton creation and use is beyond the scope of this document, but we give a brief overview here.

Some of the applications skeletons are used for include deformations of mesh models based on their skeleton deformations [5], similarity estimation between shapes by their skeleton structure (particularly a Reeb graph skeleton) [8], parameterizations of surfaces based on feature selection and subsequent skeletonization [20], and scientific visualization of fibrous and thread-like data [12]. Many of these applications have particular requirements for the skeletons, and incorporate additional data with the skeleton, such as a distance mapping to a surface from which they were extracted, or radius information.

Similarly, there are a large number of methods for computing skeletons (i.e., skeletonization). These generally fall into two categories: those extracting skeletons from volumetric data, and those computing a skeleton for a mesh model. Much volume data generally arises from biomedical scans, and a wide variety of methods are used to generate skeletons for these. Zhou et al. [21] use voxel coding (essentially clustering and thinning of the input) to achieve skeletons, while Yim et al. [18] perform ordered region growing and pruning for their skeletons (particularly of magnetic resonance angiography). Borgefors et al. [4] define and extract a curve skeleton based on thinning a volume, yet Reinders et al. [14] further tackle the level of detail for skeleton reconstruction of their volumes by means of skeletonograph (uses a thinning approach with distance transform information). Finally, two locality focused tracing algorithms of volumes to generate skeletons from 3 dimensional fibrous data include

Han et al. [6], with maximum intensity projection tracing, and Mayerich et al. [11], with a GPU accelerated volume trace by predicting trajectories. Often, such volumetric skeletons are used as a proxy for the object itself, and represent a medial axis approximation of the object. A variety of approaches are also used for skeletonization from meshes. Tierny et al. [9] identify feature points and constructs a dual Reeb graph for the skeleton, while Lien et al. [10] perform shape decomposition refinement for computing the skeleton. Au et al. [3] use feature points to perform constrained laplacian smoothing to contract a mesh to its curve skeleton, and Schaefer et al. [15] use multiple mesh poses for use extracting segmentation and then skeletonization and weighting of the skeleton from a mesh. Some approaches such as Yoshizawa et al. [19] also extract a "mesh skeleton" (one composed of surfaces as well as curves) by means of voronoi tesselation and smoothing, but we do not consider such skeletons, here.

There also is prior work on skeleton matching and comparison. One example is the work of Ward et al. [17], which uses perturbations of the boundary of a shape to generate several medial axis skeletons in order to find a better general skeleton of an object. While they do some similar matching of the skeletons, their work deals with proximities and only a single skeletonization algorithm and has yet to be extended to the 3rd dimension that is needed with data sets we use.

We are aware of only one other approach used on biological data sets for error quantification and merging of traced skeletons. Helmstaedter et al. [7] define a system, RESCOP, for comparing manually traced skeletons to achieve a final result skeleton for neurite reconstruction. Using a heuristic distance function for voting on the validity of existence of connecting line segments by the individual skeletons, they gauge certainty for the composite result by using an estimated error bound. While this paper is like-minded in motivation and similar in idea to our own, we

approach our solution of a composite skeleton by using various geometric and topological characteristics to aid in matching unordered segments generated by the more scalable automated methods rather than manual traces.

Our work here is motivated specifically by and focuses on biological data and use in visualization, but the methods should be applicable to other data sources and/or applications.

3. THEORY AND METHOD

We describe in this section the basic definitions and a brief overview of the method before diving into more details about input formats and the measurements and procedure for combining skeletal models.

3.1 Definitions

Three types of models must be described: the skeletal model, which is the general form for skeletons including input; the set model, which is the working model we create while trying to reconstruct; and the result model, our final model.

3.1.1 Skeletal Model

While the particular skeleton corresponding to a given model is not well-defined, there is a general consensus on what makes a skeleton. Skeletons consist of a set of points connected by curves and/or line segments, referred to as *edges* (Fig.3.1). In our discussion, we will assume that the skeleton is made up of (poly-)lines; this is the most common representation, and any curve skeleton can be converted to a polyline skeleton to a given precision.

Figure 3.1: Simple edge example, with endpoints containing data such as radius.

Points will be classified as one of three types: *endpoints*, which have only one outgoing edge and terminate a skeleton, *midpoints*, which have two outgoing edges and are basically interior points of polylines, and *branch points*, which have three

or more outgoing edges and denote bifurcation of the skeleton. This classification allows us to distinctly categorize a *segment* path along the skeleton, where a segment is composed of points going from any non-midpoint to any other non-midpoint (i.e., a polyline terminated by branch points or endpoints as in Fig.3.2).

These segments thus represent separate branch paths within a skeleton, and may be augmented to keep track of connectivity information by having each segment store two linked lists corresponding to the connections for the head and tail of the segment. With the connectivity being maintained at the segment level, the skeleton itself is maintained as a simple list of its segments.

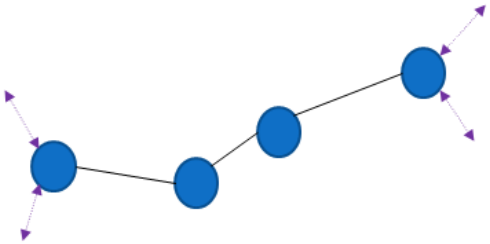

Figure 3.2: Simple segment example, with pointers to other connecting segments at head and tail.

As the segment already stores connectivity information, the skeleton itself needs only be composed of a list of corresponding segments (see Fig. 3.3). Additional information is often associated with skeletons, and this is typically stored by a table reference at each of the edge endpoints. This is perhaps a simplification of the underlying problem space, as information could change more rapidly than just at the indexed points, though one could refine the original skeletal partitioning to maintain higher levels of fidelity. For our motivating example, a radius value is associated with each edge endpoint, thus describing the radius of the tubular structure through

which the skeleton is passing at that point. Note that for an ideal tubular structure, including this information makes the skeleton a medial axis transform. For other skeletal structures, other types of information might be kept, such as density, color, or vertex weights for use in a deformation process. Note also that this information is kept for the endpoint of a line segment.

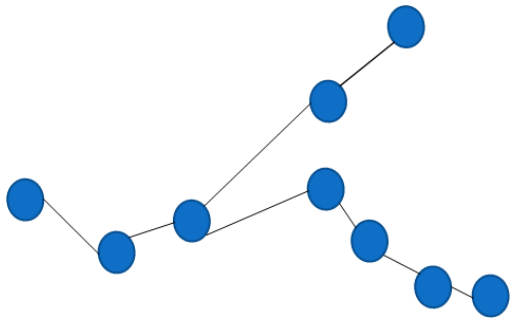

Figure 3.3: Simple skeleton example, which is composed of a list of its composite segments.

3.1.2 Set Model

Our *set model* represents the thus far added skeletons as sets of segments (see Fig. 3.4). A global skeletal graph is maintained. Each edge of this global graph tracks the individual segments that correspond to that edge, and vertices of the graph store additional information. A single segment (graph edge) of the combined skeletal model has a list of indices for the segments/subset of segments from the input that are mapped here. Consistency in the overall graph and the set information stored at edges and vertices is ensured by allowing modifications only through a set of rules for updating the combined skeletal model (Section 3.7.1).

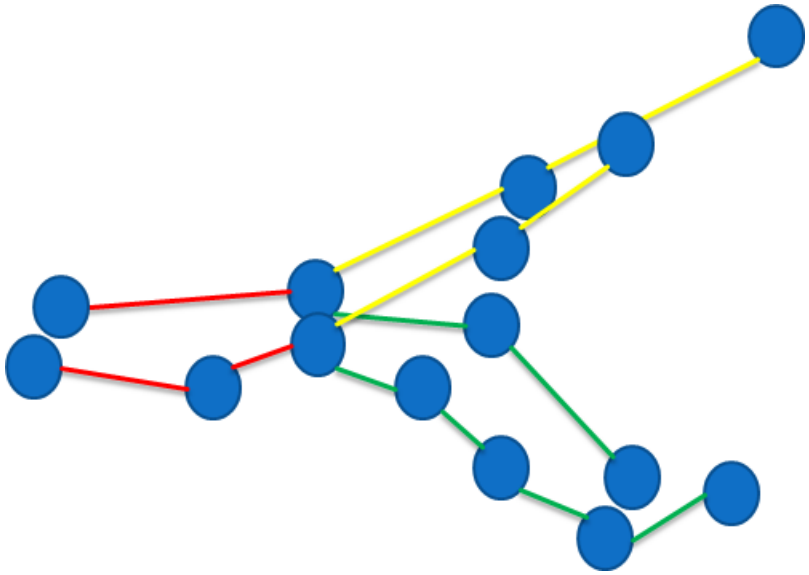

Figure 3.4: The set model is represented by a series of individual sets (color dictates single set) that each correspond to single final segments in the result model. Within each set is a list of segments that are used to form this final segment for the region, where the region is denoted by branch points or farthest extents of the list of segments.

3.1.3 Result Model

The *result model* is of the same form as a skeletal model (a list of segments and end-/mid-/branch-points), and is generated from the set model (Fig. 3.5). The graph edges and points of the set model, including the segments that map to each edge, are used to define the edges and branch points of the result model. As a result of the mapping process, the result model also computes and stores the uncertainty information in terms of interval mapping and variances. This uncertainty information may be useful in fine tuning the result model to be composed of non-outlier subsets of segments as well as for the creation of bi-directional maps from the result model to all input skeletal models.

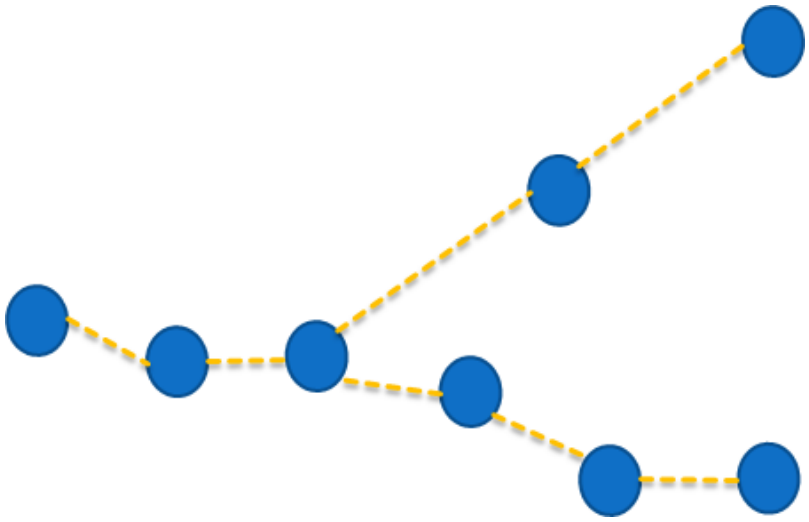

Figure 3.5: The result model is a skeleton that also contains information about the combined consensus at each of its points.

3.2 Method Overview

The basic overview of our method can split into four sections, each of which may have particular functions used in subsequent sections as needed. The following setup serves as a background for the layout of our method and each component is expanded in the following sections.

- *Input initialization*
  - Forming a global list of segments
  - Marking segment topology
- *Sequential addition of all segments to set model*
  - Distance mapping
  - Segment comparison
  - Set combination

  - Mapping Rules

- *Correction of set model for order independence*
  - Segment comparison
  - Merge
  - Set combination
  - Split

- *Composition of result model from set model*
  - Set combination
  - Repairing topology

3.3 Input Initialization

In practice, when the skeleton model is first received as a list of points and connections, we perform a simple flooding-type algorithm to find all the segments for the skeleton. The flooding algorithm adds all non-midpoints (endpoints and branch-points) to a list PL (Fig. 3.6). We then proceed to take each subsequent element

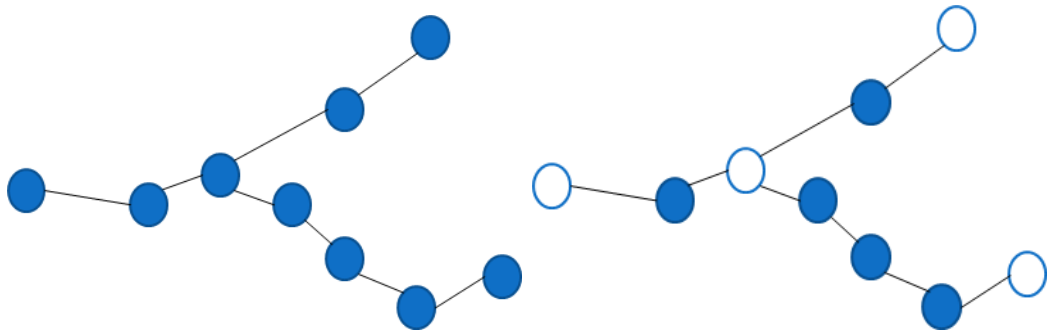

Figure 3.6: (Left)Sample input skeleton. (Right)Marked non-midpoints in white added to PL.

in PL and sequentially walk along each of its non-marked paths while marking the path as we go till we reach another element of PL. The completion of any path thus represents a new segment that we add to a growing global segment list SL (Fig. 3.7). It should be noted that as we assume no skeleton should include a loop internally, we

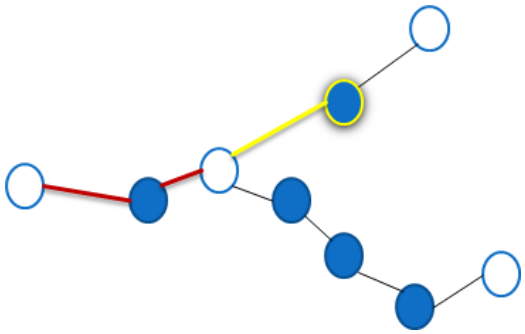

Figure 3.7: Walk/Mark along each unmarked path until another element in PL is located. Red is a completed segment and yellow is being traced.

discard any path that proceeds back to its orignal start location. After processing the entire list PL, we mark each segment that connects at its head or tail respectively (Fig. 3.8). As the input skeletons are not necessarily of the same partition length,

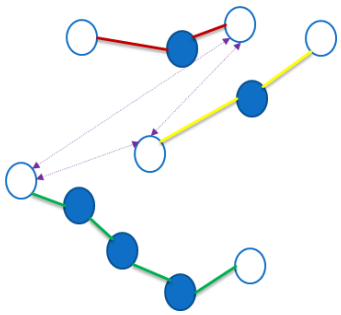

Figure 3.8: Mark each topological connection among the segments (contains the same endpoint at head or tail).

we resample (repartition) each of the segments to a fixed edge length where this is possible. Thus as the segments are all stored in a single global list SL, a skeleton (and the segment connections) need only be composed of a list of indices to this global list.

For the set model there are several differing options for initialization. In one basic approach, an empty set model may be created prior to the loading of any actual skeletal models, meaning the set model is the empty set. Another initialization option is to load any single skeleton model as the initial set model. There are other more complex options possible, but the choice of the initialization configuration makes little difference in the end result or the process of construction. The subsequent two phase merge/split set operations will help to aid in the robustness of the set model composition independent of the initialization and the order of addition to the set model.

The result model needs no initialization, but rather takes as input the last known set model. More information about the specific state of the set model at the time of forming the result model can be found in Composing the Set Model (section 3.7).

3.4 Combination

The following section seeks to describe the progression of the combination of geometric objects, from sets of points on up sets to segments. Given any two or more geometric objects that are intended to represent pieces of the same underlying model, their combination to a single result yields a more complex model if equally representing the inputs in the result or equal complexity in the case of averaging.

In the case of a set of points in space representing a single point of a model, a simple averaging operation allows us to find a good point with the assumption all inputs are equally plausible.

For line segments (edges), the orientation of each segment with regards to the other needs to first be determined before any averaging operation could be performed. This orientation could be found by the minimum of the sum of the pairwise distances to the endpoints (Fig. 3.9) and then pairwise averaging between end points could be performed. However, this method does not allow for the case of both line segments actually representing disjoint or overlapping pieces of the same underlying line segment. Thus for our purposes, we determine orientation and parameterization along that orientation by solving for the parameters for end points of the line segments along the principal component axis and then having a connected sequence of line segments that may then be smoothed as as desired (Fig.3.9).

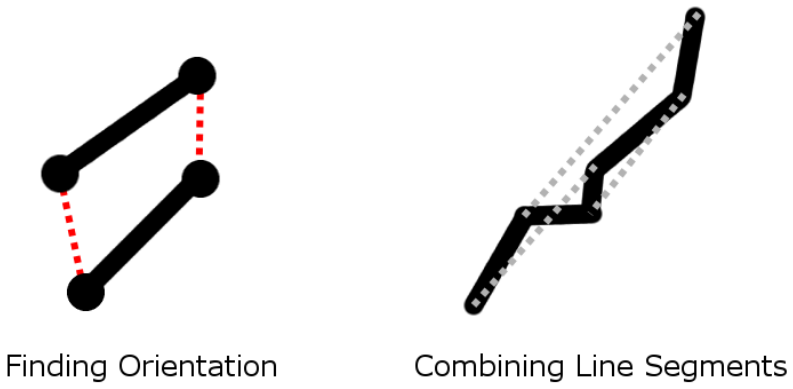

Figure 3.9: The left figure corresponds to using the summed minimun pair distance between endpoints to get a basis for orientation. The right figure is an illustration of the result of the combination of multiple line segments (dotted lines) to a segment containing all their endpoints.

To locate the principal axes of any set of n point data, principal component

analysis (PCA) can be used. To calculate the PCA by means of the covariance method, we first take each of the dimensions (p of them) for the points (stored in X, an n x p matrix), calculate the mean for each dimension, and store it in vector u.

$$u[i] = \frac{1}{n} \sum_{i=1}^{n} X[i, j]$$

We then subtract this mean from each of the points and store the values in an n x p matrix B.

$$B = X - hu^T (\text{where } h \text{ is an } n \text{ x } 1 \text{ column vector of 1's})$$

We then compute the covariance matrix C by taking the outer product of the matrix B with itself.

$$C = \frac{1}{n-1} B^T B$$

To find the principal axis, we need to locate the key eigenvectors V of the matrix C. These eignvectors correspond to the eigenvalues in the diagonal matrix D.

$$V^{-1} C V = D$$

Upon calculating V and D (in our implementation we use the GNU Scientific Library) we then sort the eigenvectors by their paired eigenvalues in D and use the largest eigenvalue corresponding vector as our principal axis. A visible example of PCA on a set of points in two dimensions can be seen in Fig. 3.10).

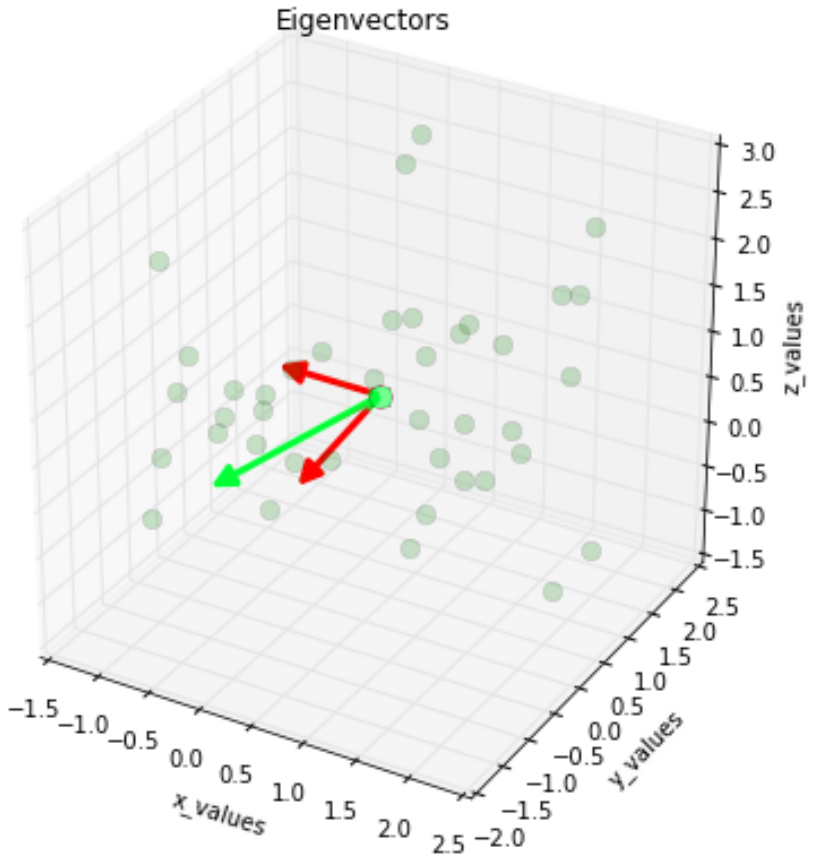

Figure 3.10: An example of the PCA of a set of points in space, with the green arrow representing the principal axis (generated following [13]).

For a series of consecutive end to end connected line segments with another that represent potentially disjoint or overlapping pieces, the orientation and parameterization are needed to find a single result. These can be found by creating an approximation of the result by using the method of polynomial fitting (described in detail below). To ensure a regular sampling, first each segment is partitioned into a set of points at equal distances along the segment. These points are then used as input for a PCA parameterization along the principal axis for a least squares fit of a polyno-

mial to the data, where the polynomial is of low degree to try to ensure the points of each segment's points retain their monotonic ordering in parameterization (see Fig. 3.11). Using this polynomial, the location of each line segment interval is mapped by finding the respective end point's closest parameter point on the polynomial.

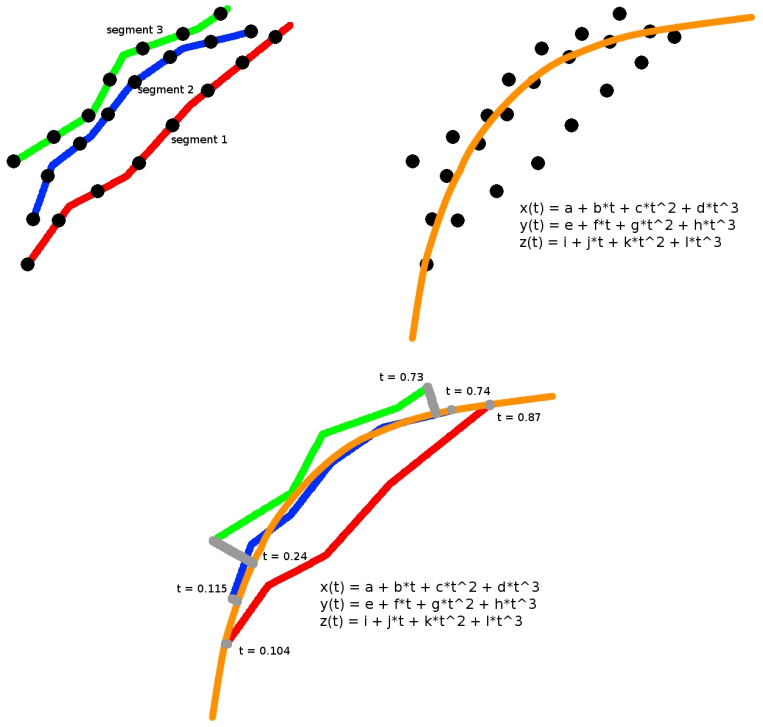

Figure 3.11: Points along each segment in the set are taken at regular intervals (top left). Using these points, a best fit polynomial is constructed using least squares optimization using PCA parameterization of points(top right). Finally, the parameter of the closest point on the polynomial is used as a basis for ordering the segments in a global manner with respect to the set (bottom).

This parameterization inherently grants a mathematically robust global parame-

terization, which can then be used to compose the combined result. The combination is made by generating the monotonicly enforced parameterization of the midpoints of each segment based upon length of the edges composing each segment (see Fig. 3.12).

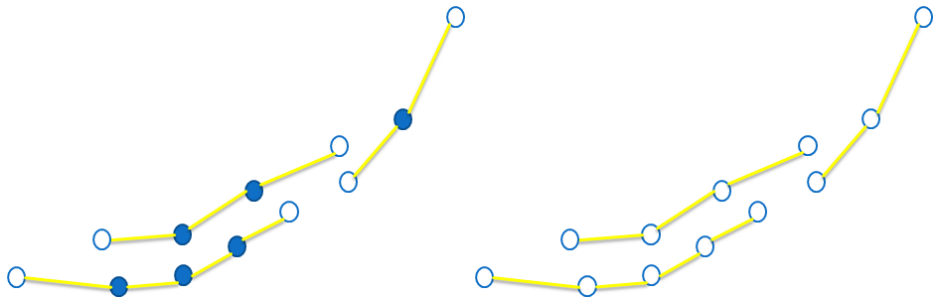

Figure 3.12: (Left) Parameterized endpoints based on polynomial. (Right) Monotonic parameterization of midpoints based edge length.

Now having all the key points of interest mapped along a common parameterization, we then perform an averaging of the segments mapped intervals. The resulting points store not only the average position, but also the number of combining segments and may store distance average and variance (see Fig. 3.13).

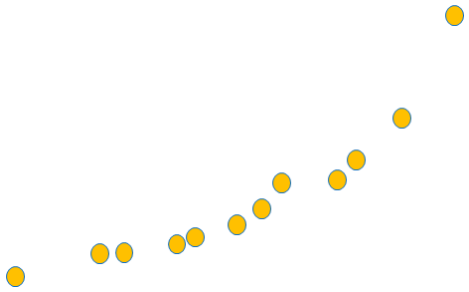

Figure 3.13: Averaged points resulting from averaging segments at each change along interval.

These points are composed into a single result segment and then subsampled for smoothing, just as in the input initialization phase (see Fig. 3.14).

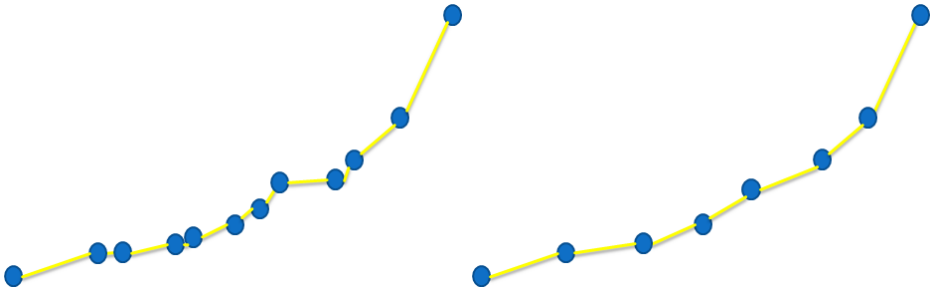

Figure 3.14: (Left) Average points composed into a segment. (Right) Segment re-sampled to achieve smoothness.

To perfom the least squares polynomial fitting for data points, we first do a PCA computation as has been explained previously for the parameterization of the points along the principal axis. Given this parameterization, we sort the points and then calculate the coefficients of the best fit parameters of a polynomial of degree $m-1$ for the n points. This least squares fitting is done by matrix manipulation of the

equations using a Vandermonde matrix with the parameter values of the points (t). Using this Vandermonde matrix W with the values of t taken from the PCA mapping, we compute the parameters of each dimension's polynomial equation of degree $m-1$ (with $m < n$).

$$\begin{bmatrix} 1 & t_1 & t_1^2 & \cdots & t_1^{m-1} \\ 1 & t_2 & t_2^2 & \cdots & t_2^{m-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & t_n & t_n^2 & \cdots & t_n^{m-1} \end{bmatrix} * \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$\begin{aligned} Wa &= x \\ a &= (W^T W)^{-1} W^T x \end{aligned}$$

Thus, with each of the parameters computed for each dimension we now have a least squares fit polynomial of degree $m-1$ for the point sets with parameter values t. A sample image of least squares fitting of polynomials in two dimensions can be seen in Fig. 3.15. While one may choose to a nonlinear least squares fit to determine the parameterization of the points at the same time as the polynomial with approaches, for our data sets we had trouble for convergence even when initialized with the PCA parameterization.

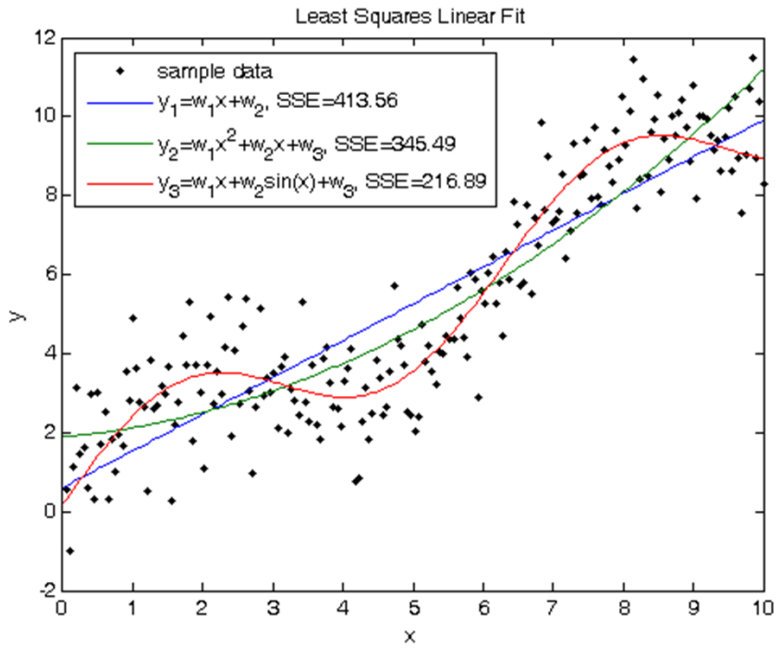

Figure 3.15: An example of the least squares polynomial fitting of a set of points in space (generated following [16]).

3.5 Segment Comparison

Determining wether a geometric object is representative of another and thus is a candidate to be combined can be analyzed in a variety of ways. In this section we look at the increasing level of complexity that should be considered for similarity of points on up to segments.

In the case of a point in space matching with another point or set of points in space, looking at the geometric quantity of distance allows us to readily check liklihood of combination, while other point data, like color, size, or density, can help to clarify the matter further. Each of these parameters can be used in conjuction with a threshold value to allow a neighborhood of acceptance that would then be

logically represented as a single point.

For line segments (edges), the geometric aspects of: orientational difference, average distance, and minimum distance are all additional criteria to check. The orientation particularly will allow the distinguishing of nearby edges of opposing orientation whereas distance alone would fail.

For a segment, new geometric aspects of line segment to line segment angle variance and smallest average distance from one segment to another come into consideration. Also, the case of topology regarding the number of connections at the ends of the segment become an issue and factor into classification of similarity.

Though each of the categories of geometric, topological, and data comparison allow for evaluation of the segments, their scoring is weighted so as to allow priority to be given to certain comparisons. We allow the user to set the associated weights for each aspect of the evaluation, though in practice we prioritize the aspects in the order given below and limit comparisons to sets within a certain radius of the combining segment.

Geometric

- The first metric we use is an approximation of the smallest average distance between the segments (or the least of the average distance from one segment's edge endpoints to the opposing segment). We compute an approximation of this average distance by taking the minimum of the two segments' average distance from each other at distinct points, where each segment's distance is the sum of the closest distance from each of its line segment endpoints to the closest point on the other segment, divided by the total number of distances. See Fig. 3.16. This metric is used to mitigate the trouble of matching small segments to larger ones. Also, complications due to different edge lengths for

the input segments are curbed by repartitioning each of the input segments.

- The second metric we take into account is the minimum distance between the segments, namely the minimum distance between any pair of opposite edges of the segments.

- The third metric is the angular difference between the segments, which is always bounded by 90 degrees for a maximum angle. This metric assumes segments are bi-directional.

- The final geometric metric is the difference between the sum of angular variations along sequential edges of the segments, or the sum of angle changes from line segment to line segment across the segment.

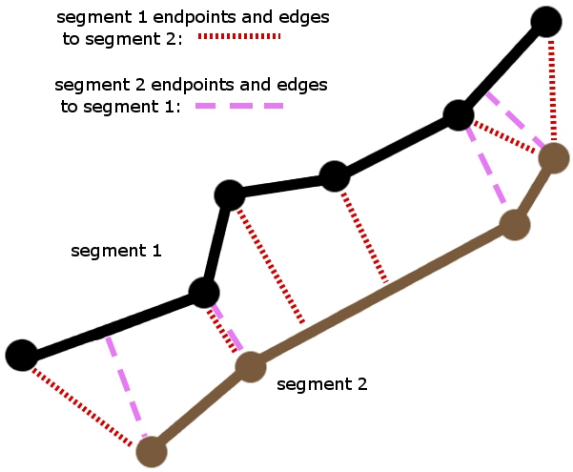

Figure 3.16: The one-sided distance is computed by averaging the distances to one segment from all of the other segment's edgepoints. The minimum average distance is the smaller of the two one-sided average distances.

Topological

- The topological metric measures the valence difference among the pairing of opposing segment endpoints, assuming the direction is the same.

Data

- For our motivating example, when tracing various tubular structures we also record the radius information at each edge endpoint. For the comparison between two segments we use this to calculate an average radius along the segment.

Other characteristics can also be considered, but for our data we have found the above comparisons to be sufficient. Since some of the characteristics can be quite varied with respect to each other, in practice we establish weights to key in on parameters the user is seeking.

3.6 Distance Mapping

To address the issue of mapping an interval match between two segments we perform distance mapping to achieve intervals used for both non-branching points and branching points (branching points will require a close cut). First, given two segments S_1 and S_2, we compute for each edge endpoint in S_1 the corresponding closest point along S_2. Next we find for each edge in S_1 the closest point along S_2 (this is needed as seen with the case of perpendicular edges with closest point lying in their middle). Third we use these first two distance computations to find the first and last points or edges that lie within the specified distance and mark their parameterization along the segments ([0-1] range) as the interval. These distances and a user specified distance threshold let us compute a mapping interval in which the entire segment is mapped to portions of the segment to which it is closest, or

else to a null region (if there is no point on the other segment sufficiently close). We then proceed to do these same three steps from S_2 to S_1, thus having two sets of interval mappings (S_1 to S_2 and S_2 to S_1). Note that because the comparison is for the minimum distance, the two sets of interval mappings likely will not be the same. Thus when we are using interval maps we project these parameterizations and perform a union in the case for loose intervals (ones not requiring a cut because of a branch point), and a intersection in the case of for tight intervals (only for the cut or branch part itself). One final word about this distance mapping is that this form of interval mapping allows to maintain the same underlying topology of the combining skeletons instead of introducing cuts for sections which match within a distance at the ends but go beyond that threshold in their middle.

3.7 Composing the Set Model

We construct the set model incrementally, adding one input skeleton at a time into the set model by a set of mapping rules stated below. Each segment of the added skeleton becomes part of the set model, either in a new set of segments, or joining with an existing set of segments. After all the segments are combined with the set model we then refine the set model for order independence by the two primary operations of merging and splitting.

3.7.1 Mapping Rules

Given a segment from the skeleton being added to the set model, we want to find the segment from the set model that it most closely matches. Using the comparison between segments (see section 3.5), we find the segment that closest matches the combining segment and then mark its set to be the key. This key set then has its average result segment computed (see section 3.4) and this result segment is used to compute the interval mapping (see section 3.6) and how it will be combined with the

set. Each pair of segments can be classified for combination in one of three ways (no match, match, or partial match), having five results (combine, split, test again with higher threshold, or new set), with an overview of the matching algorithm shown (Fig. 3.22 and Fig. 3.23).

A **no match** with the result segment occurs if the distance metric exceeds the threshold or there are no potentially matching sets (see Fig. 3.17). With either of these cases, the new segment cannot be added to the set model, so we create a new empty set of segments, add the combining segment into it, and add it to our set model.

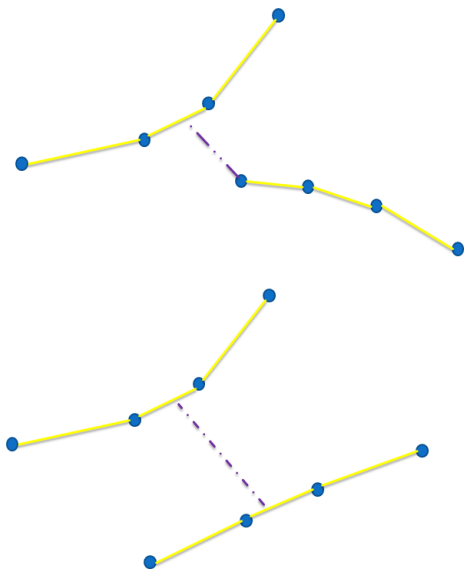

Figure 3.17: Cases where no match is detected and a new set is created include when the comparison score too far and when the closest distance between the segments is too great.

A **match** with the result segment occurs when the entire $[0-1]$ interval of both

segments is mapped to the other segment (see Fig. 3.18). In this case, one segment is treated as coinciding with the other for the full interval and thus we simply add the segment to the corresponding set of segments that were used to form the result segment. Note that in this case we do not need to worry about the connective topology of the end points of the segments as we do not subdivide one of them relative to the other, and thus segment status is unchanged.

A **partial match** with the result segment occurs when a subset of the $[0-1]$ interval is mapped for each segment. For partial matches, a more complicated procedure is involved, as described below. For this partial match case, more information about the valence of two segments' endpoints must be evaluated to determine how the segment might possibly be split and combined into to the set model. The aspects of the various cases are discussed in detail below.

1. *Combine:* This case occurs when the combining segment is deemed to be close enough and safe to combine with the set (see Fig. 3.18). There are two basic scenarios where this can happen. The first is when the mapped interval of each segment overlaps an opposing endpoint (i.e. [.. 1] and [0 ..]), with the overlapped endpoints for both segments having a valence of 2 or less. The second is where one segment interval is subsumed in the other interval (i.e. [.. ..] and [0 1]) and the endpoints of the subsumed segment have valence 2 or less. In both scenarios, the new segment is considered close enough to the other and thus either addeding to or extending the set segment will be "safe" operations.

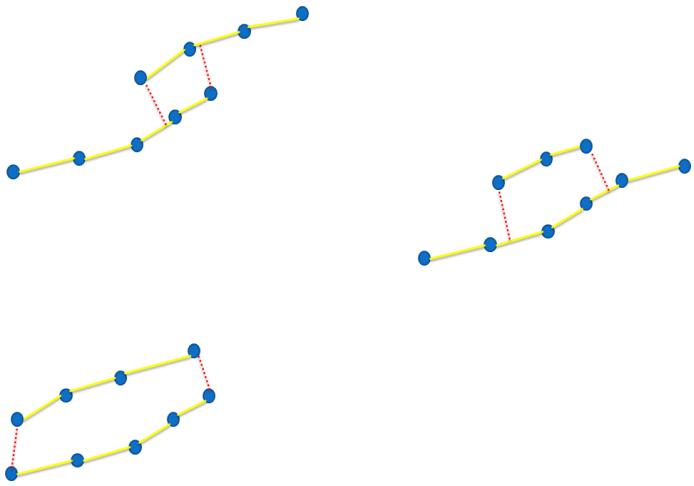

Figure 3.18: Cases where combining is detected and the set can safely and logically add the combining segment: (Top) overlapping non-branching endpoint, (Right) subsumed segment contains non-branching endpoints, (Bottom) match case.

2. *Split:* Splitting addresses the issues raised by the set model containing a branch that is not within the combining skeleton. This will occur when the mapping between segments places an endpoint of the set segment in the interior of the combining segment, and that set segment has valence 3 or more (see Fig. 3.19). In such cases, partitioning the combining segment into two or three pieces (depending on how the other endpoint is mapped and what its valence is) and re-testing the segment pieces individually allows for the safe mapping of segments to the set model.

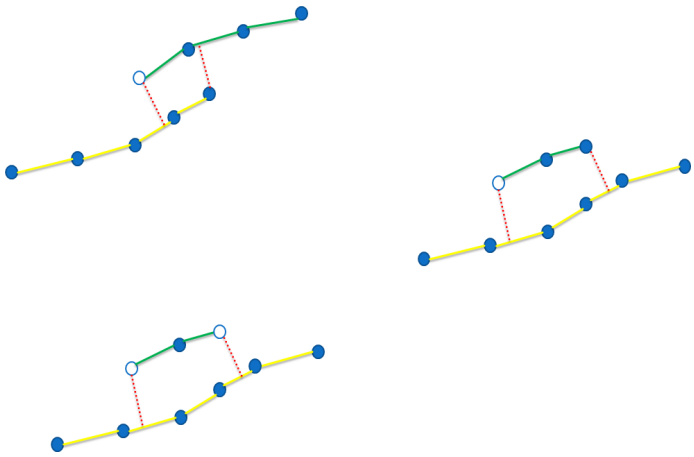

Figure 3.19: Cases where splitting is detected and the set must cut the combining segment (yellow) due to the valence of the result segment (green): (Top) overlapping branching endpoint, (Right) subsumed segment contains one branching endpoints, (Bottom) subsumed segment contains both branching endpoints.

3. *Swap:* Swapping is defined as when the combined segment becomes the sole occupant of the set and all the previous segments of the set are re-tested to find their new position in the set model. This case is made necessary whenever the combining segment has an endpoint of valence 3 or more that maps to the interior interval of the set segment - i.e. the combining skeleton has a branch that was not part of the set model (see Fig. 3.20). It should be noted that the set segment is always evaluated first for valence and partitioning the combining segment, thus preventing a looping swap call.

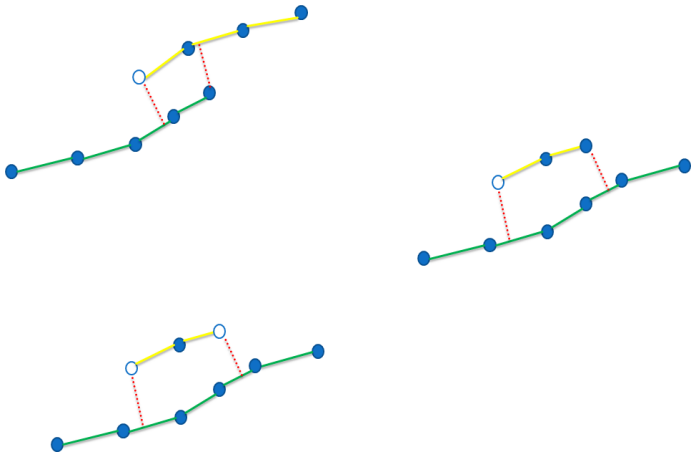

Figure 3.20: Cases where swapping is detected and the set must switch out the set due to the valence of the combining segment (yellow) and the valence of the result segment (green): (Top) overlapping branching endpoint, (Right) subsumed segment contains one branching endpoints, (Bottom) subsumed segment contains both branching endpoints.

4. *Retest:* The case where there is no good combination and there is no cause for swapping or splitting the segment (based on valence of endpoints in mapped intervals) can occur when no endpoint maps to the interval, a partial mapping where only one endpoint (valence 2 or less) is mapped to the interval, or when two endpoints from opposing segments map to the interval but no segment's endpoint pair is contained in the interval (see Fig. 3.21). We have two options for handling this. The first is to split both the combining and set segments and merge their pieces based on distance. The disadvantage to this approach is that it can unnecessarily complicate the segment structure and lead to many small sub-regions and loops in the result. The second option (the one we employ) is to modify our match criteria slightly and retest, with this most commonly

involving the increase of the allowed distance for matching. While this latter approach does allow a larger margin of error into the similarity evaluation, it also allows for contiguous mappings between the segment and the set model (thus things that are at least partially close will remain connected in single segment pieces in the end).

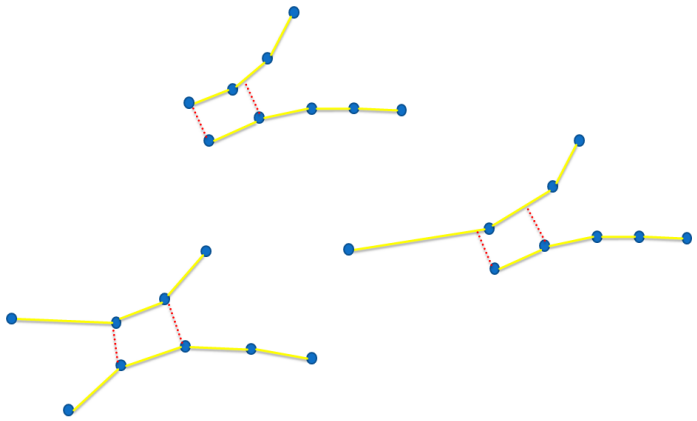

Figure 3.21: Cases where retesting is detected and the criteria for matching is increased slightly before testing again: (Top) same endpoints mapped but no segment's endpoint pair is mapped, (Right) only one non-branching endpoint mapped, (Left) no endpoint is mapped.

5. *New set:* The final case we handle is a bit rarer, and occurs when the combining segment cannot logically be combined with the existing set. In this case, a new set must be created for it. The scenarios where this occurs are when the endpoints of either segment map to a single point on the interval (as with perpendicular lines) and the valence of one of the endpoints is 3 or more. In such a situation, it might seem that splitting the segments and introducing branch

points would be the correct solution, but that would introduce connections in the final result that were inconsistent with all of the input skeletons. We choose instead to maintain topological input similarity and have two sets that end up passing within our normal allowable similarity space with regards to each other.

In the end, we have a mapping from the combining segment to the segment sets in the set model that does not subdivide the underlying segments more than would be required by the explicit input skeleton branch points. As the segments are split, each piece still maintains the connectivity with respect to both its counterpart and its endpoints. Thus, each segment set contains information about what other segment sets it connects with by the segments' recorded connections with each other.

This concludes the mapping rules for adding segments, though there still needs to be a stage in set maintenance to ensure order independence with skeleton combinations. For a diagram of the mapping rules and logical flow of deciding where a segment will go in the set model see Fig. 3.22 and Fig. 3.23.

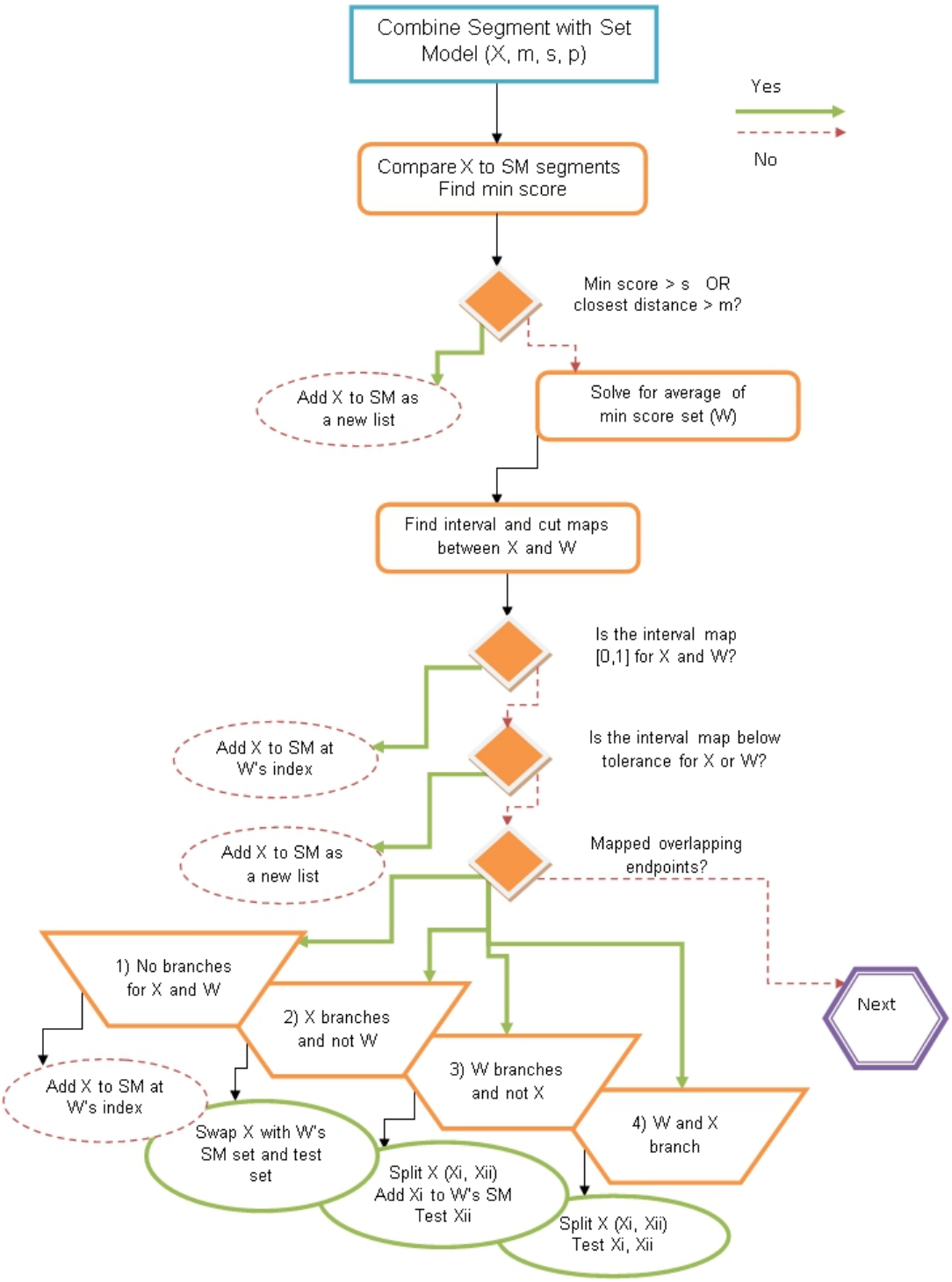

Figure 3.22: The flow of the decision tree for deciding how a segment X will be combined with the Set Model given a distance threshold m, score threshold s, and point tolerance threshold p (used for detecting regions too small).

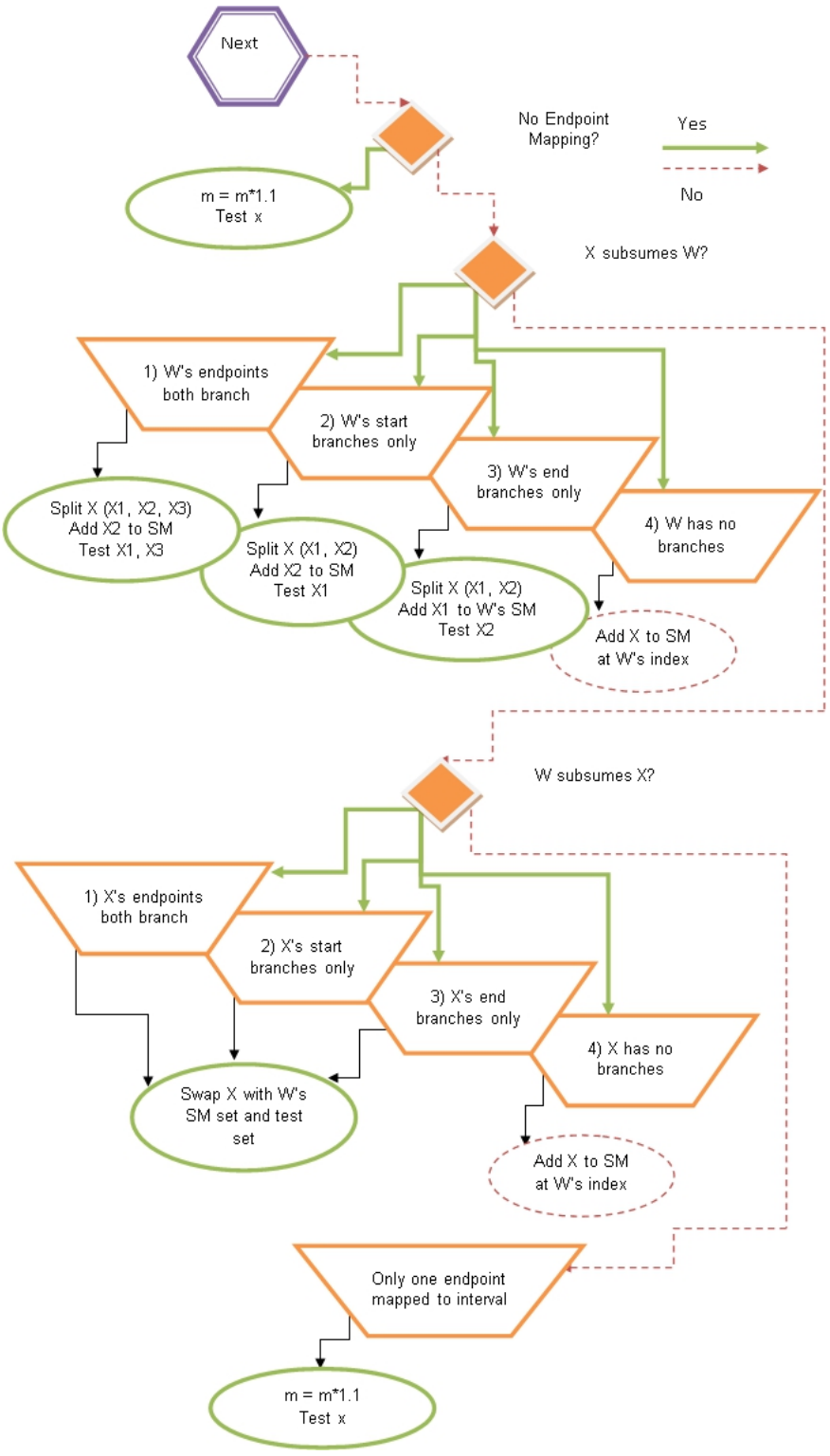

Figure 3.23: The second section of the decision tree for deciding how a segment X will be combined with the Set Model.

3.7.2 Maintaining Order Independence (Merge and Split)

In the establishment of sets of segments, the linear order of combination can play a key role in the final sets. To mitigate this concern we employ the following merge and split approach to achieve a more robust result. In this section we look at the result of applying a merge and split approach for sets of points on up to sets of segments.

For sets of points representing singular points in an underlying model the idea is to incrementally merge all point sets that have points within some threshold distance of each other. This merging effect tends to create large sets that are larger than or equally as large as the maximum potential set given all the possible combination orders. With this largest set distribution accomplished, we then compose an average point and check the similarity of each point in the set with regards to the average. If there exists a point or set of points outside a specified similarity score we will take the most offending point (point 1) and the point in the set most dissimilar to it (point 2). Thus with the two farthest opposing parts of the set we recategorize the set into two based on fitting better with point 1 or point 2, subsequently splitting on each subset as needed (failing the similarity threshold).

For sets of line segments (sets of edges) representing single underlying line segments one may perform a similar incremental merging based on the distance from any point on the edge of one set to an edge of another set. Again, this allows us to have large sets that encompass the maximum potential set given by any combination order. Using this encompassing set distribution, we then compose an average line segment (see 3.4 Combination) and check the similarity (see 3.5 Segment Comparison) of each to see if splitting of the set should commence given a similarity threshold. As with the point example, we take the most offending edge and its counterpart and

split the segment into two using them as the basis for the two new sets. We then subdivide further based on the similarity results within the two new sets.

Similarly to points and edges, in order to reduce order dependence of our result with regards to sets of segments we use this two phase operation of first merging sets based on distance and of secondly partitioning sets based on their variance in similarity (see 3.5 Segment Comparison) with the average result of the set (see 3.4 Combination).

This first phase of merging sets is done by a pairwise segment-segment comparison of nearby segments from each opposing set to see if there exists some segment pair that has a similarity score within a defined threshold. It should be noted that if the combination of the sets would internalize a branch point we do not proceed with the merge (to prevent over collapsing). In the case that no pair is found to match within the threshold, we conclude the sets are distinct with respect to each other. For the case where a pair is found to match similarity within the threshold, we say there is overlap in the sets (as there would be a set match given just the pair) and thus collapse the two sets into one. Taking into account this new set we compare sets until all nearby sets have been checked with regards to their pair similarities. As stated with points and edges, the idea in this phase is to combine neighboring segments (those mapped into the same set in the set model) into a minimal number of sets as if they had been given in the order that maximizes the size of the set. The necessity of partitioning these larger sets we handle in the next phase.

For the second phase we look at each set individually and see if it should be split based on individual segment similarity difference (see 3.5 Segment Comparison) from the average segment (see 3.4 Combination). We find the similarity score of every segment in the set with the average segment and evaluate whether the score is beyond the specified split threshold. As with the case of points and edges, if a split

is incurred we partition the set into two based upon the most offending segment and its least similar counterpart. Thereafter we recurse upon the splitting operation until the sets are within the specified split threshold similarity. It should be noted we no longer split apart segments at this stage, thus, if merging allows for collapsing over a branch point the segment is no longer faithful to the combined input skeletons. As the cutting position is dependent on the order, order indepence can truly be achieved with merging and splitting only when the cut positions are uniform in the final segment state (same regardless of ordering).

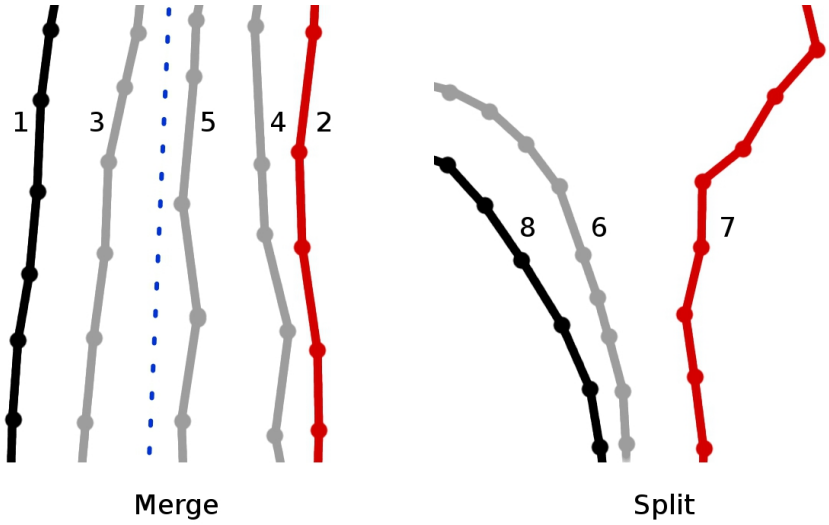

Figure 3.24: On the left: If segments are combined in monotonic fashion with regards to their numbering, the case on the left results in two segment sets in the the combined skeletal model even though adding them in the order 1,3,5,4,2 results in a single segment set. By merging sets we resolve this issue by ensuring all segments that would map together are in the same set. On the right: With segments continuing to map together and also their sets undergoing merging, sometimes the range of variability for an individual set grows beyond the desired bounds (say for a radius measure). By subdividing the set into two with regards to the farthest matches we then ensure that at each level of subdivision the set is rid of the largest outlier from its current median.

The underlying reasoning behind the merging and splitting is seen in Fig. 3.24. In practice we use the same scoring function and values for merging and splitting as we did for matching, though this could be changed to tailor the combination to fit a more specific style.

3.8 Forming the Result Model

After processing all the input skeletons into the set model, the combining of each set into their respective final segment and connection repairing is handled in creating the result model.

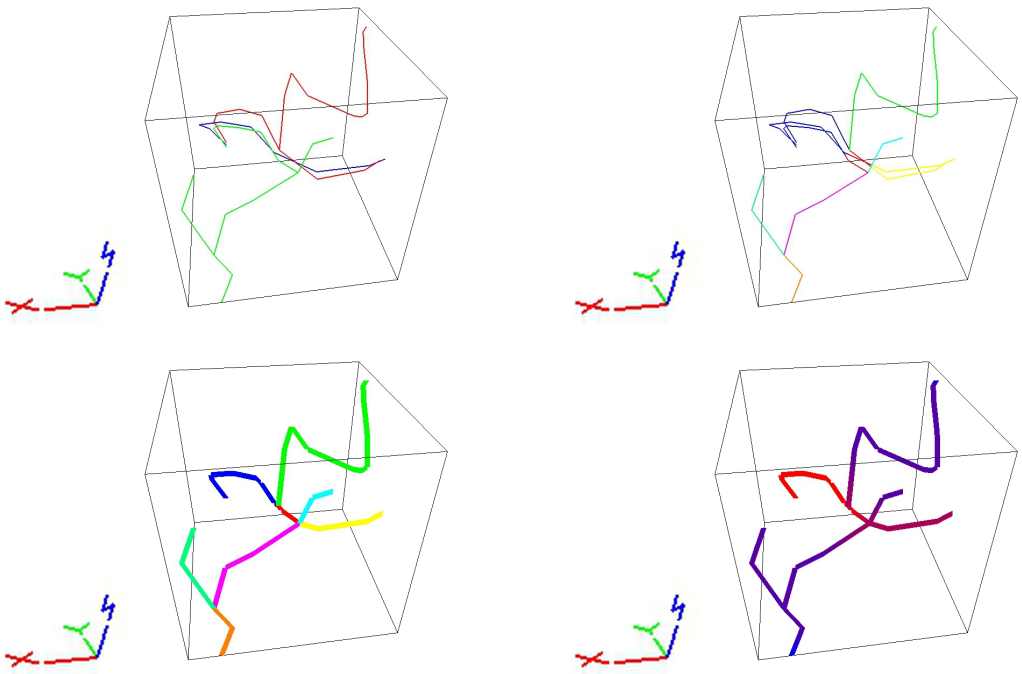

Figure 3.25: Combining synthetic skeletons from a model. Three different skeletons are combined into a single unified model. At top left are three skeletons, each in a separate color. At top-right is the combined skeletal model, with each segment set highlighted in a different color. At bottom-left is the composite result, with each segment highlighted in a different color. At bottom-right is the composite skeleton with uncertainty values color ranging from less certain (blue) to more certain (red).

3.8.1 Set Evaluation

The combination of the various segments in a set into a single result segment is handled by establishing a global ordering of their endpoints and monotonicly averaging their paths within their respective global intervals (see section 3.4).

Note that due to the ordering with the construction of the result segment we also have a two way mapping of each combining skeleton to the result skeleton via the individual segments (or subsegments in cases of splitting). Thus we can easily propagate data values stored on the result skeleton to the combining skeletons, and vice-versa.

3.8.2 Confidence Value

During this process of edge creation we also calculate a corresponding confidence level for individual points, the edges, and subsequently the segment, indicating how likely the segment is to reflect a "true" skeleton. The confidence value is based on how much that region is covered by a set of segments and is to be used in connection with the total number of skeletons added to the combined skeletal model. If a segment is contained in all the input skeletons, it will have higher confidence, while if it is missing in some, it will have lower confidence. We can also augment the confidence by using our similarity metric. For any segment, we can measure the largest distance in the set of segments in the set model to the segment in the result model, and treat (the inverse of) this as a second measure of confidence. If all segments in the set model are close together, that will increase confidence, while if the set is rather spread out, the confidence will be lower. The goal of this confidence value is that regions that have higher confidence values (more segments in the set and high coverage of the set bounds) will be those where mutual agreement among the various skeletons occurs. An example of this uncertainty and the process of creating the set model

can be seen with Fig.3.25.

3.8.3 Connection Repair

After generating result segments for each set in the combined skeletal model, their topological connectedness may no longer be satisfied geometrically. We thus perform a geometric adjustment of the segments to have them connect as in the topological structure of the set model. Using the connectivity lists from each segment of the set, we know which segment (and thus which set) the result segment should connect with at the fore and aft. With these connections we make a geometric 'snapping' modification if the existing geometric connection is invalid.

The snapping occurs by modifying the corresponding connected ends of the segments to a point formed by averaging all the incident vertices (Fig.3.26). Although this changes the geometric data we have at the endpoint (since the position is now based on the geometry of adjacent segments), it is necessary to preserve the topological constraint.

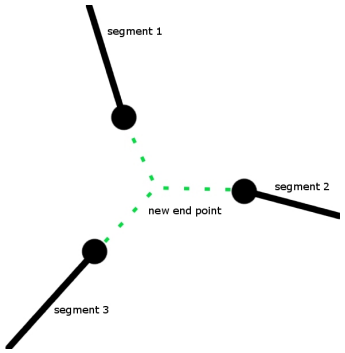

Figure 3.26: If the endpoints of mulltiple result segments match together topologically, geometric connectivity is ensured by setting their endpoints to be the average of the of the mapped endpoints.

With the connection repair finished, we now have our result model. As the combination process allows us to determine consensus data at each point along our segments, the result model now stores an accurate (albeit simplified) consensus measurement along each of the points composing the skeleton.

4. RESULTS AND EVALUATION

By implementing our method we now demonstrate its application in several following examples.

4.1 Combining Different Volume Skeletons

There are a wide variety of methods that can be used for skeletonization. Skeletonization is important to many aspects of visualization and analysis of certain volumetric data sets. One example is vascular data; the skeleton provides a medial axis for the data, and thus makes it much easier to perform analysis and visualization of the data (e.g. [12]). Different skeletonization algorithms or different parameter settings for the same algorithm are better at capturing different parts of the data, and thus there is a need to combine results from multiple skeletonizations of the same volume. Figure 3.25, Figure 4.1, and Figure 4.2 show examples of combined skeletons. shows an example.

4.2 Combining Adjacent Volume Skeletons

Recent imaging methods have begun to produce very large amounts (terabytes) of volume data. Given the size of such data sets, analysis of the data is performed over smaller subvolumes that fit in memory. Skeletonization is critical for visualization and for understanding the connectivity of structures over large distances (vascular and neuronal structures can extend a very long distance in the volume). To effectively deal with these large data sets, we need to be able to merge the skeletons generated from each subvolume into a unified skeleton (Fig. 4.3).

Figure 4.4 and Figure 4.5 show the results of a combined skeleton for such an application. As the figure demonstrates, skeletons from overlapping regions are eas-

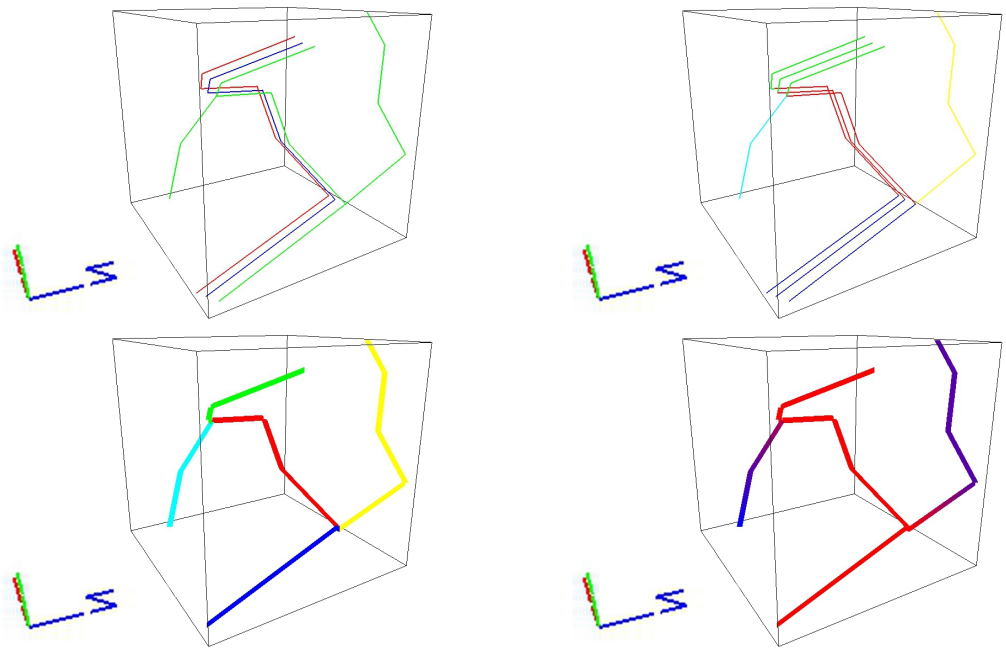

Figure 4.1: Combining synthetic skeletons from a model. Three different skeletons are combined into a single unified model. At top left are three skeletons, each in a separate color. At top-right is the combined skeletal model, with each segment set highlighted in a different color. At bottom-left is the composite result, with each segment highlighted in a different color. At bottom-right is the composite skeleton with uncertainty values color ranging from less certain (blue) to more certain (red).

ily combined into a single unified skeleton; this combined skeleton is suitable for use in further visualization, simulation, and analysis applications. A closer look at overlapping region can be found with Figure 4.6, and shows how well the the combined skeleton result matches a trace of the whole volume.

4.3 Combining Mesh Skeletons

Skeletonization is used in graphics applications to, among other things, provide intuitive control for deformation, parameterization across an object, and a base for simplification. We demonstrate that, like volume-based skeletons, we can combine mesh-based skeletons into a unified model. An example is shown in Figure 4.7.

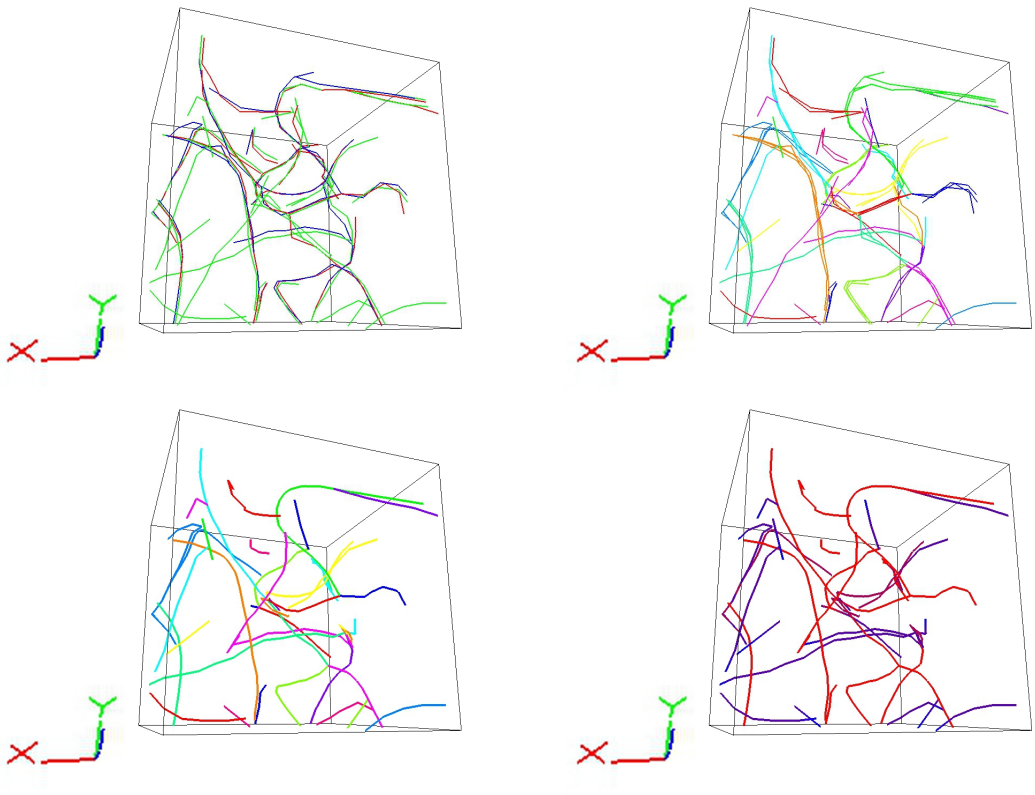

Figure 4.2: Combining skeletons from two different algorithms [11, 6], and several parameter settings for each. The skeletons trace the vascular structure from part of the mouse Cerebellum. At top-left are the individual skeletons, each in a different color. At top-right is the set model, with each segment set highlighted in a different color. At bottom-left is the result model with each segment a different color. At bottom-right is the result model colored based on uncertainty, ranging from less certain (blue) to more certain (red).

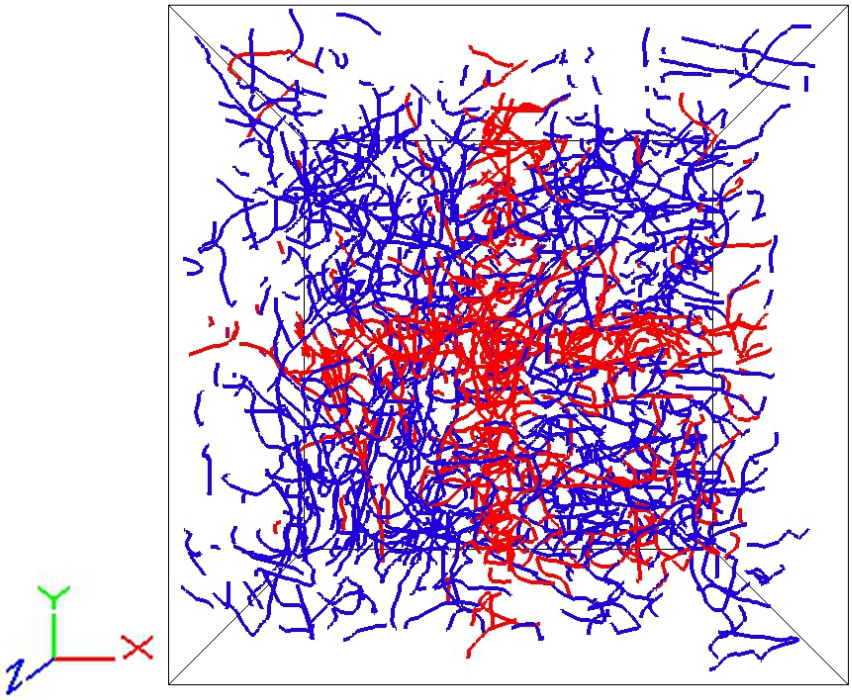

Figure 4.3: Combining 8 adjacent volumes' skeletons from different traces using the same algorithm [6] over overlapping volume regions. The image displays the combining segments with color denoting no combination (blue) or combination (red). For this case, each volume is 550^3 in space with 100 overlap, thus allowing us to piece together a skeleton for a 1000^3 volume we cannot trace normally in memory.

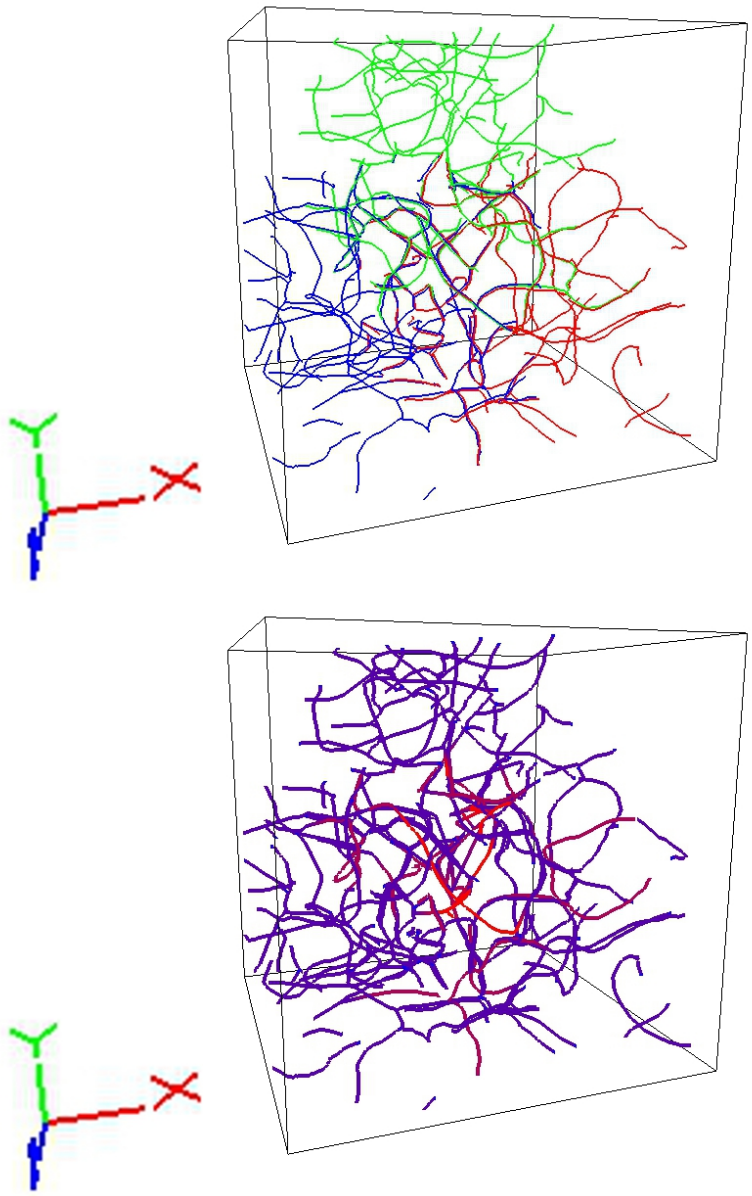

Figure 4.4: Combining 3 skeletons from different traces using the same algorithm [6] over overlapping volume regions. The skeletons trace the vascular structure from part of the mouse cerebellum, and the same algorithm (with the same parameter settings) is used for each of the three volumes. The top image displays the skeletons (denoted by individual colors), and the bottom image displays the result model with color denoting uncertainty ranging from less certain (blue) to more certain (red). In this case, the overlapping region is more certain, while the less overlapped area is less certain.

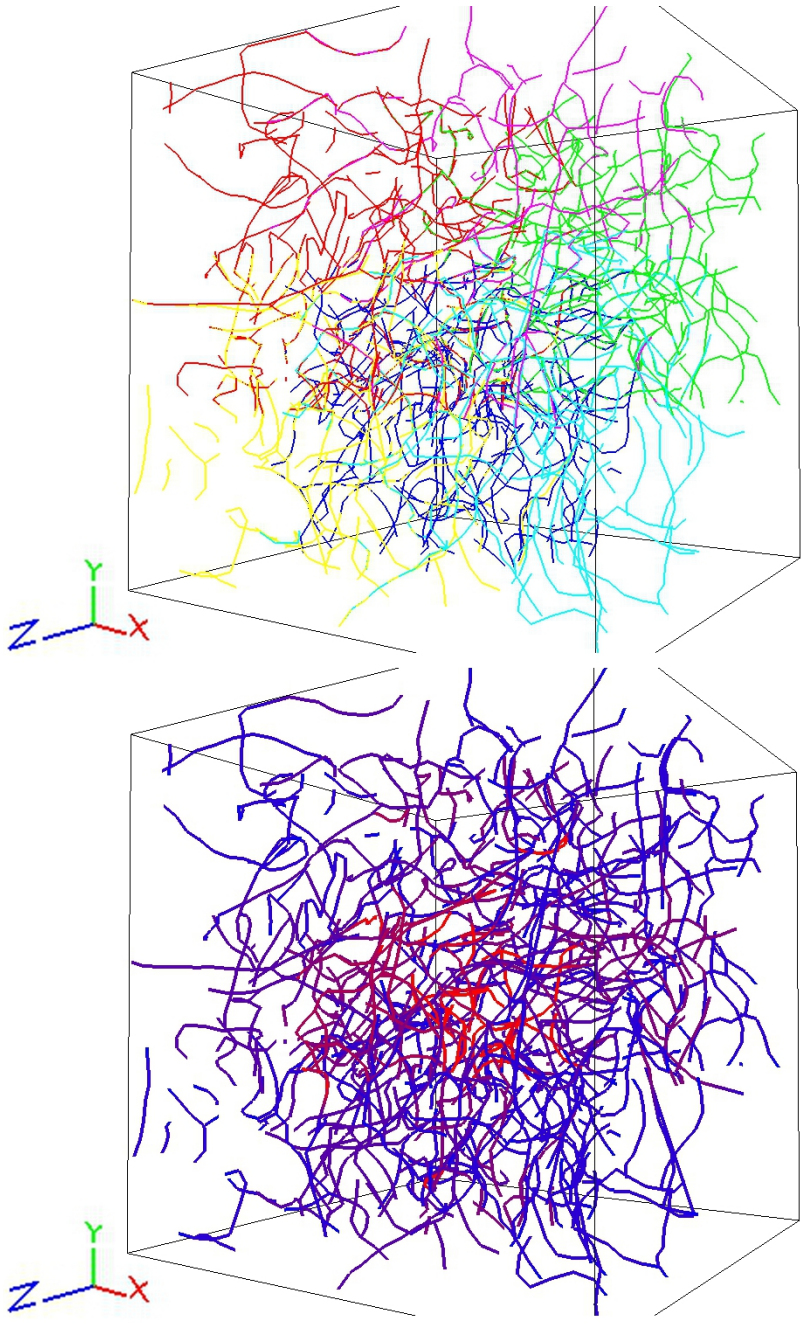

Figure 4.5: Combining 6 skeletons from different traces using the same algorithm [6] over overlapping volume regions. The skeletons trace the vascular structure from part of the mouse cerebellum, and the same algorithm (with the same parameter settings) is used for each of the three volumes. The top image displays the skeletons (denoted by individual colors), and the bottom image displays the result model with color denoting uncertainty ranging from less certain (blue) to more certain (red). In this case, the overlapping region is more certain, while the less overlapped area is less certain.

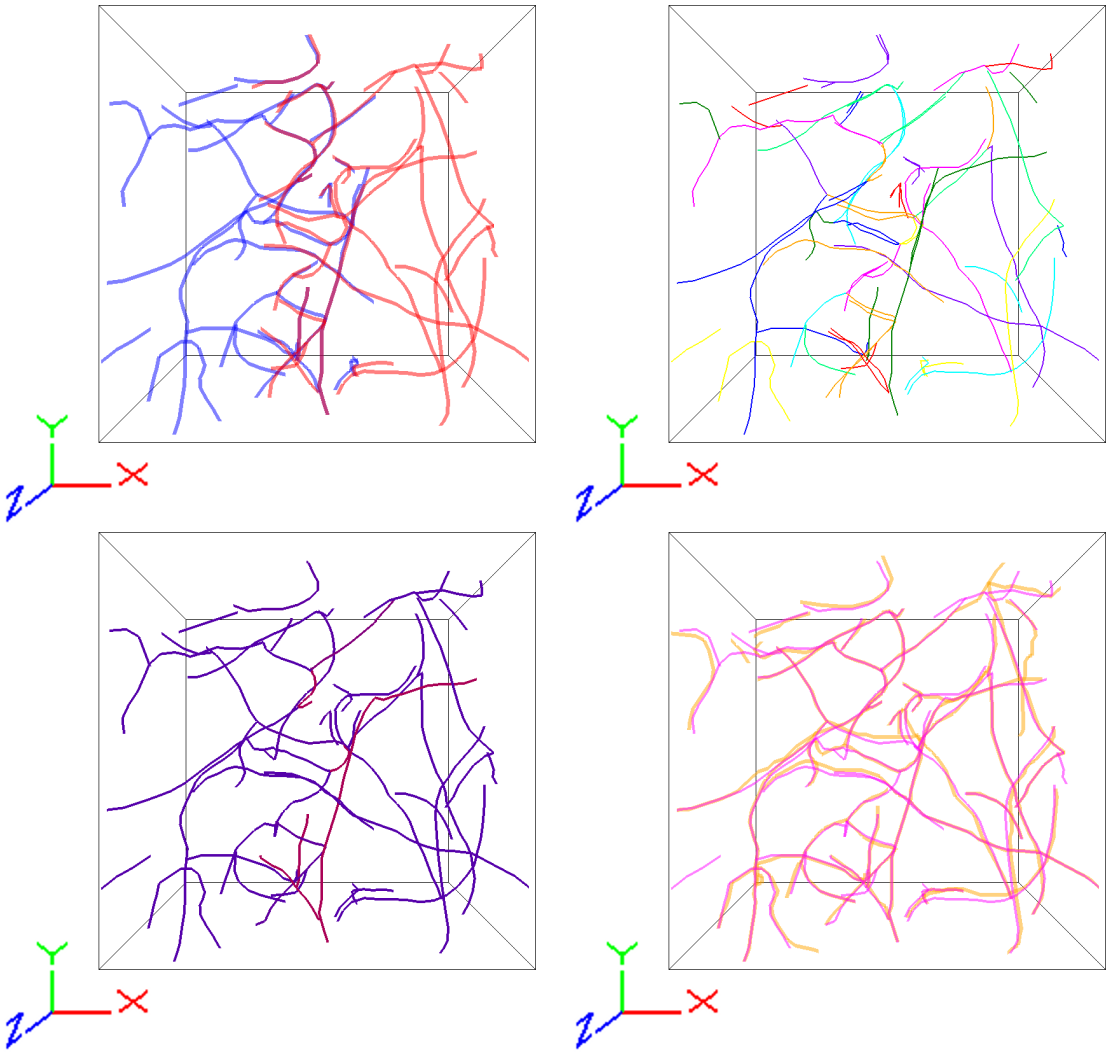

Figure 4.6: Combining 2 skeletons generated by [6] over subsets of a larger volume of the vascular structure of the mouse cerebellum. The top left image displays the input skeletons (denoted by individual colors), and the top right image displays the set model with color denoting individual sets. The bottom left displays the uncertainty ranging from less certain (blue) to more certain (red) and the bottom right displays the final result (pink) with a trace over the entire volume (orange).

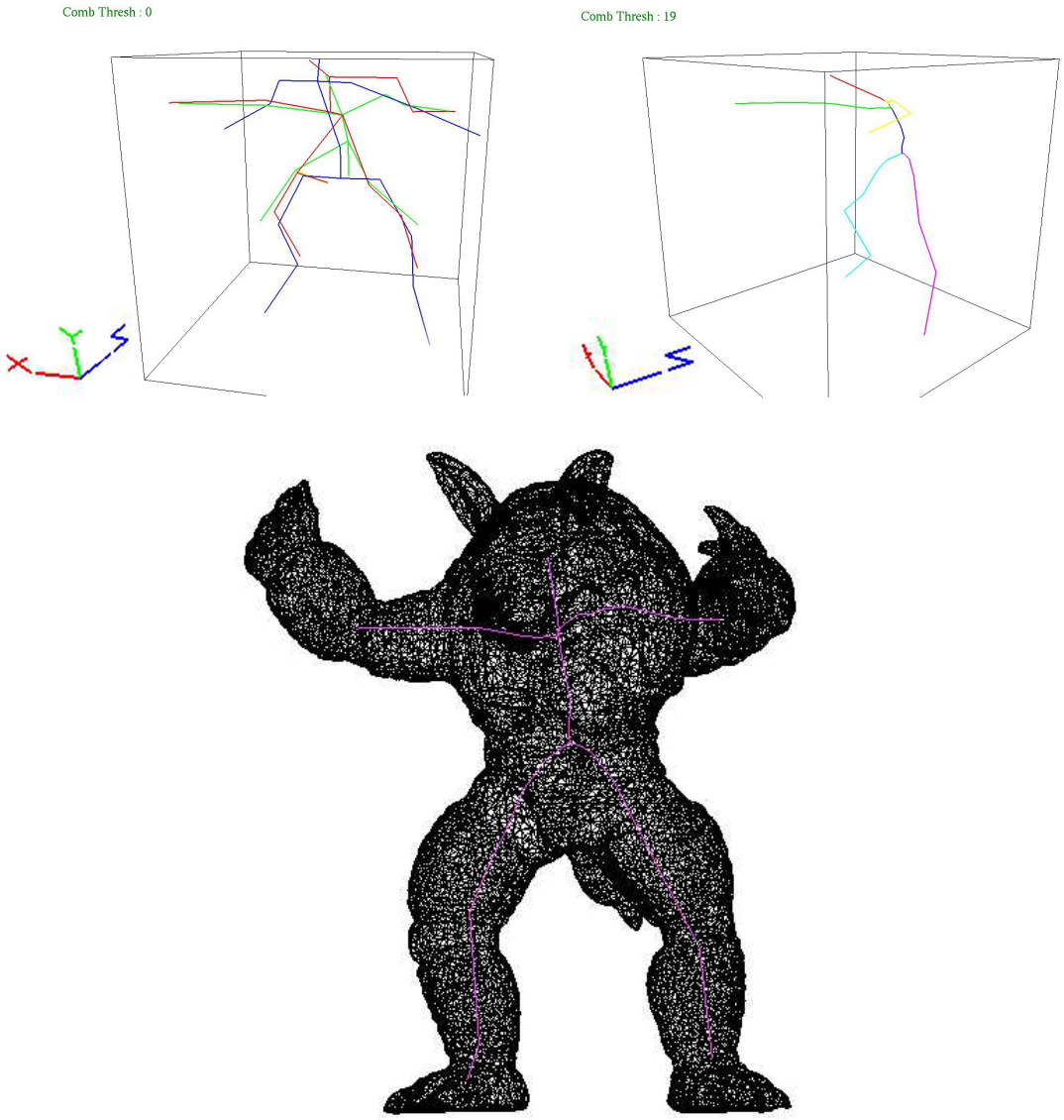

Figure 4.7: Combining skeletons from the armadilloman mesh model. Three different skeletons are combined into a single unified model. At top left are three skeletons, each in a separate color. At top right is the combined skeleton; the colors demonstrate different segments in the combined model. At bottom is the combined skeleton overlaid on the model.

5. DISCUSSION AND CONCLUSION

In this section we move on to a discussion of applications of our method, its contributions, concluding thoughts, and future work.

5.1 Queries

Using the final result skeleton, we can perform a number of geometric queries. While we can treat this final skeleton as a single model, we have additional information, namely confidence values and mappings to original skeletons, that will allow us to gather more interesting results from the skeleton. In particular, we can calculate a confidence value or range of confidence associated with most geometric queries. This can allow us to interactively query the structure to get a value with uncertainty that is not typically available when dealing with a single data set. We must note that this confidence value is not a reliable measure in any strict statistical sense, but rather a measure of how consistent the result is versus all of the input skeletons.

As one example, if measuring distance from a point to the skeleton, we can first find the nearest point on the result skeleton. We can then look at the mapped point on any of the input skeletons to find a range of possible distance values, from each of those points. While only approximate, this will give an interval of values that should be a reasonable range of measures to the "best" skeleton, and a better sense of distance than the single value returned from any one skeleton would have been.

As another example of such a query, when analyzing vascular data, information such as the fraction of space occupied by blood vessels, average length of a blood vessel between branches and the frequency of branching in a fixed volume are of interest. The confidence values we compute along segments can be used to determine the likelihood of each segment actually being part of the vasculature (this is often

not clear-cut in real-world data), and we can thus compute confidence values for the gross statistics within a region. That is, if all input skeletons contain the segment, we can include it with certainty, while if only a fraction do, we can weight that segment's contribution by that fraction.

Several other similar queries are also possible, including those that would make use of the statistical radius or other information that is stored in the combined skeleton.

5.2 Discussion

There are several applications this method for skeleton combination can be used in, and in our work is just one step towards the goal of automatic tracing and segmentation of large volumes of high resolution medical volumes. One use of skeleton combination is as a sort of signal reconstruction, using the assumption that the noisy/less reliable input skeletons are samples, and we are trying to reconstruct a more accurate structure underlying them. Through the construction of the result skeleton one can easily visualize which regions may need more analysis via low confidence values. Another use for skeleton combination is detail addition (as with level of detail) or expansion (as with adjacency or overlapping parts) of existing structures that can be applied in online visualizations. While these concepts are readily applicable to the area of trace combination for tubular networks, other uses such as obtaining a combined weight skeleton for mesh deformation and mapping topological changes to skeletons for meshes over time are not as conveniantly discerned due to the particular bone/weight assosiations and the topological inconsistency across models respectively.

Some limitations to the approach we have outlined are as follows. Our presented method does not allow for the cases of non-branching self connected structures due to

endpoint constraints (i.e. a loop with no branches). It should be noted that in cases where there are neighboring branches all of highly similar orientation and close proximity our algorithm will collapse them to one final segment. Also, the combination approach means that the method can only help to minimize error that is not globally (or near globally) consistent. This means that if the vast majority of input skeletons all possess the same error type, the result will likely possess that error, though that error may be identified through looking at the uncertainty matching for the data (or variance across the result model). Finally, our approach is limited to dealing with whatever input is provided, and problems in that input can be propogated in the output. For instance, if the majority input skeletons fail to identify some segment in the data, there is no chance that the output will show a favorable certainty for that segment. However, the advantage to our approach is that if even one input provides that data, we will have some way of representing it (even if with only low confidence).

5.3 Revisiting the Ideal Combined Model

- *Compact accessible representation*: The result of our method still remains a skeleton with only a small addition of data at the points, thus the compactness and accessibility reasons for using the skeleton remain.

- *Consensus function*: As the result stores the mapping amount (as well as can store average deviation and variance data) at its points, the consensus function requirement is satisfied.

- *Order independence*: Provided the cut position of segments is uniform, the merge and split operations ensure a robust set model state and thus result. This condition however is rather stringent and should be improved in the future.

- *Weighting of inputs*: Any combining segments may have an individual scoring threshold and be combined using a weighted average, therefore enabling a particular skeleton more effect on the composed result.

5.4 Conclusion

We have presented a new method for combining skeletal information from multiple skeletal structures to obtain a single skeletal model. This can allow us to:

- Have a more complete representation of the object (when different skeletons provide complementary information). Thus, a we can visualize more complete and accurate models than we would be able to otherwise.

- Compute skeletons over larger regions (i.e., stitching skeletons together). This will allow us to visualize larger regions of data than we could otherwise, as well as to have more complete models over a larger region.

- Compute an "average" skeleton with order independence (provided uniformity of cuts in final segment state).

- Possess consensus data that is available at any point along the result skeleton (slightly simplified due to resampling), allowing us to visualize the result and at the same time display a measure of uncertainty in the result.

5.5 Future Work

There are several tasks to pursue regarding this work in the future. First, relaxing the order independence constraint by ensuring cut partitions are uniform across segments would be beneficial. Second, improving on the scalability of the method would be a favorable change as currently the set model requires access to the index of all segments being combined. Currently the solution to scalability is to have a

pyramid type structure for composing levels of result skeletons that are then fed as input skeletons to the next iteration of a more encompassing result skeleton (which loses some finer details of the lower skeletons). An interesting venture would be to extend this approach to combinations of higher-order skeletal models. Finally, the uncertainty information that this approach makes available should allow an even wider range of uncertainty visualization approaches than are mentioned here.

REFERENCES

[1] Skeleton transform, 2014. http://reference.wolfram.com/language/ref/ SkeletonTransform.html.

[2] Upper body armature, 2014. http://wiki.blender.org/index.php/Doc:2.4/ Tutorials/Animation/BSoD/Character_Animation/Upper_body_armature# Testing_the_rig_and_adjusting_the_arms_for_Auto-IK.

[3] Oscar Kin-Chung Au, Chiew-Lan Tai, Hung-Kuo Chu, Daniel Cohen-Or, and Tong-Yee Lee. Skeleton extraction by mesh contraction. *ACM Transactions on Graphics*, 27:44:1 – 44:10, August 2008.

[4] Gunilla Borgefors, Ingela Nystrm, and Gabriella Sanniti Di Baja. Computing skeletons in three dimensions. *Pattern Recognition*, 32(7):1225 – 1236, 1999.

[5] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. Interactive skeleton-driven dynamic deformations. *ACM Transactions on Graphics*, 21:586 – 593, July 2002.

[6] Donghyeop Han, John Keyser, and Yoonsuck Choe. A local maximum intensity projection tracing of vasculature in knife-edge scanning microscope volume data. In *IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 1259 – 1262, July 2009.

[7] Moritz Helmstaedter, Kevin L Briggman, and Winfried Denk. High-accuracy neurite reconstruction for high-throughput neuroanatomy. *Nature Neuroscience*, July 2011.

[8] Masaki Hilaga, Yoshihisa Shinagawa, Taku Kohmura, and Tosiyasu L. Kunii. Topology matching for fully automatic similarity estimation of 3d shapes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH, pages 203 – 212, New York, NY, USA, 2001. ACM.

[9] Mohamed Daoudi Julien Tierny, Jean Vandeborre. 3d mesh skeleton extraction using topological and geometrical analyses. In *Proceedings of Pacific Graphics*, pages 85 – 94, 2006.

[10] Jyh-Ming Lien, John Keyser, and Nancy M. Amato. Simultaneous shape decomposition and skeletonization. In *Proceedings of the 2006 ACM symposium on Solid and physical modeling*, SPM, pages 219 – 228, New York, NY, USA, 2006. ACM.

[11] David Mayerich and John Keyser. Hardware accelerated segmentation of complex volumetric filament networks. *IEEE Transactions on Visualization and Computer Graphics*, 15(4):670 – 681, July-August 2009.

[12] Zeki Melek, David Mayerich, Cem Yuksel, and John Keyser. Visualization of fibrous and thread-like data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1165 – 1172, September-October 2006.

[13] S. Raschka. Implementing a principal component analysis (pca) in python step by step, 2014. http://spartanideas.msu.edu/2014/04/ 13/implementing-a-principal-component-analysis-pca-in-python-step-by-step/.

[14] Freek Reinders, Melvin E. D. Jacobson, and Frits H. Post. Skeleton graph generation for feature shape description. In *Joint Eurographics and IEEE TCVG Symposion on Vizualization*, pages 73 – 82. Springer Verlag, 2000.

[15] Scott Schaefer and Cem Yuksel. Example-based skeleton extraction. In *Eurographics Symposium on Geometry Processing*, pages 153 – 162, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[16] V. Strijov. Least squares linear fit, 2014. http://strijov.com/sources/demo_least_squares_fit.php.

[17] Aaron D. Ward and Ghassan Hamarneh. The groupwise medial axis transform for fuzzy skeletonization and pruning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(6):1084 – 1096, June 2010.

[18] P.J. Yim, P.L. Choyke, and R.M. Summers. Gray-scale skeletonization of small vessels in magnetic resonance angiography. *IEEE Transactions on Medical Imaging*, 19(6):568 – 576, June 2000.

[19] Shin Yoshizawa, Alexander G. Belyaev, and Hans-Peter Seidel. Free-form skeleton-driven mesh deformations. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, SM, pages 247 – 253, New York, NY, USA, 2003. ACM.

[20] Eugene Zhang, Konstantin Mischaikow, and Greg Turk. Feature-based surface parameterization and texture mapping. *ACM Transactions on Graphics*, 24:1 – 27, January 2005.

[21] Y. Zhou and A.W. Toga. Efficient skeletonization of volumetric objects. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):196 – 209, July-September 1999.