2011

A Topological Approach to Shape Analysis and Alignment

Benjamin David O'Gwynn
University of Alabama at Birmingham

A TOPOLOGICAL APPROACH TO SHAPE ANALYSIS AND ALIGNMENT

by

B. DAVID O'GWYNN

JOHN JOHNSTONE, COMMITTEE CHAIR
NIKOLAI CHERNOV
THAMAR SOLORIO
ALAN SPRAGUE
EDWARD SWAN

A DISSERTATION

Submitted to the graduate faculty of The University of Alabama at Birmingham,
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

BIRMINGHAM, ALABAMA
2011

A TOPOLOGICAL APPROACH TO SHAPE ANALYSIS AND ALIGNMENT

B. DAVID O'GWYNN

COMPUTER AND INFORMATION SCIENCES

ABSTRACT

When analyzing multiple mesh models, an important first step is to find the rigid transform that best aligns them. Many mesh alignment techniques exist, but because they treat input meshes as a collection of surface samples, they tend to be sensitive to surface variations between those meshes. We instead wish to align shapes based upon an analysis of mesh topology, thus making it possible to align shapes with similar part structures but varying surface geometry. To this end, we present a novel algorithm that extracts a curve-skeleton from a mesh based on its connectivity, rather than its surface geometry. We show that this method is robust to surface variation and achieves consistent results across shapes in the same object class. We then will present our system for mesh alignment based on our topological analysis of input meshes, and we will show that our technique is able to align shapes with similar part structures but differing surface shapes.

Keywords: Shape analysis, Mesh alignment, Skeleton extraction, Segmentation, Algorithms

TABLE OF CONTENTS

APPENDIX

DEDICATION

To my parents, for teaching daily that my best ought to be my norm.

To my siblings, for showing by example how to be my best.

To my advisor, for never accepting anything less than my best.

To my Leah, for making me better than my best. This exists because of you.

LIST OF TABLES

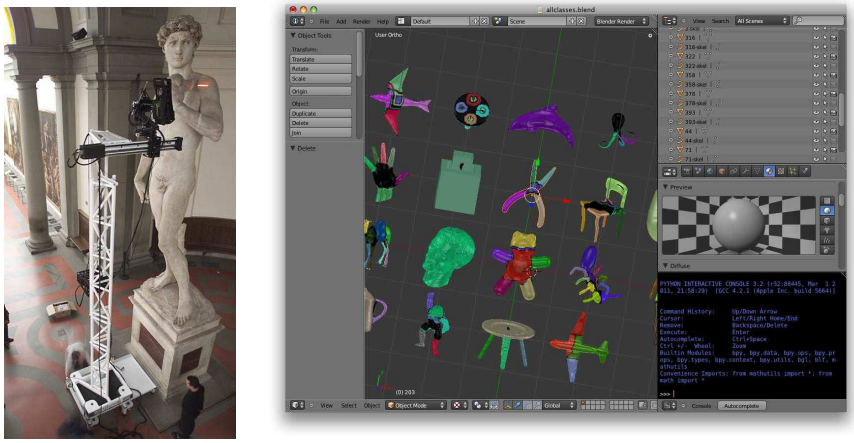
LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Recent technological advancements in the creation of 3D shapes have led to substantial, easily accessible repositories of 3D data. This data can come from many different sources such as range scans and 3D modeling packages (Figure 1.1). They are stored in many different types of databases, and these databases generally store models in arbitrary positions and orientations in $\mathbb{R}^3$ with, at best, the coarsest level of semantic information. Most tasks involving 3D models, such as quantitative shape comparison and object recognition [38], shape retrieval [70], determining upright orientation [33], view selection [15] and scene composition [84], require much more information than what is given explicitly in the database. As such, they are usually analyzed and aligned as a preliminary step.

Our goal is to improve this preliminary step by decomposing the models into their semantic parts and aligning them based on that part structure. In this work, we explore the problems of shape analysis and alignment and provide a set of methods for their solution. Specifically, we present a simple, robust means of decomposing articulated shapes into their semantic parts and embedding that part structure in a skeletal graph. We then use this skeletal graph to perform pairwise alignments of shapes.



(a) Range scans [49] (b) 3D modeling suites (Blender 3D) [16]

Figure 1.1: 3D shape data can come from many different sources of shape data. (a) Digital Michelangelo Project of Levoy *et al*. [49]. (b) Blender 3D modeling and design suite [16]. Given these variety of sources, the position and orientation of the data can arbitrary and non-uniform. This makes analysis difficult. Some means of automatic shape alignment would increase the value of existing shape databases and the applications that use them.

Our primary contribution, and our primary departure from existing literature, is that we use a topological approach to extract a shape's part structure, as opposed to most existing approaches based on low-level geometric characteristics such as curvature. The advantage of this approach is two-fold. First, by integrating part structure into the process of alignment, it allows for more stability in the face of surface variation due to noise, differences in physiology, differences in surface sampling, and pose variation. Second, our topological approach to shape analysis is efficient, automatic, easy to implement and requires no more knowledge of geometrical analysis than the ability to calculate the centroid of a set of vertices. Our system is an efficient, fire-and-forget means to analyze and align large shape databases, and it requires no more than a basic data structures and algorithms course to implement.

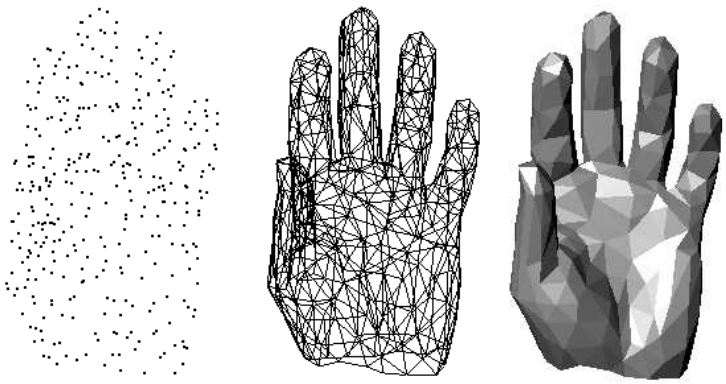
1.1 Shape Analysis and Alignment

In this section, we discuss the problems of shape analysis and alignment and the high-level motivation for our approach to these problems. By shape analysis, we mean the problem of decomposing a shape into its semantic part structure and encoding that part structure into a skeletal graph. By shape alignment, we mean rigidly transforming one shape so that it is aligned with another in a way that is meaningful and satisfactory to a human observer. We motivate the problem of shape analysis through the specific problem of alignment.

We first discuss how the primary form in which shapes are stored, the triangle surface mesh, complicates the alignment process and how most existing solutions approach this problem. We then explain how decomposing the shape into parts first and aligning them based on that part structure more closely matches how humans tackle the problem of alignment. This then motivates a discussion of the importance of shape analysis to any system of shape alignment and how we take a different tack from existing shape analysis methods.

1.1.1 Aligning triangle surface meshes

When aligning two shapes, the form in which they are stored is relevant. They are most often stored as *triangle surface meshes* (or simply *mesh*). The only explicit information that is stored in a mesh are the individual samples on the surface and the local connectivity of each vertex (Figure 1.2). Based solely on this information,



(a) Surface sample points (vertices) (b) Local connectivity of sample points (edges, faces)

Figure 1.2: The only explicit data given in a triangle mesh are (a) the sample points on the surface and (b) the local connectivity of those sample points.

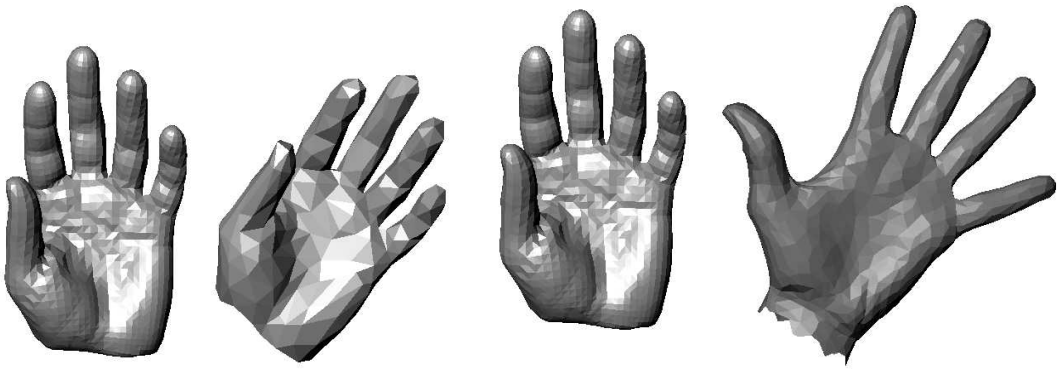
it is difficult to know exactly how to align two shapes. The two dominant approaches to this problem either:

(i) correspond surface samples and align based on that correspondence, or

(ii) find the principal axes (best-fitting ellipsoids) and/or planes of symmetry of each shape and normalize their orientation using that information.

If the models share common surface geometry, *i.e.* are either the same shape in slightly different poses or are incomplete subsurfaces of the same underlying shape, then approach (i) works well. Similarly, if the models share common planes of symmetry then approach (ii) works well, even if the shapes are of different objects. Essentially, these alignment methods treat input shapes as geometric monoliths.

However, surface noise, variation of sampling density, part articulation and differing physiology (Figure 1.3) can present difficulties for these methods. When



(a) Variations in sampling density (b) Variations in pose, physiology

Figure 1.3: Existing alignment methods treat input meshes as geometric monoliths, either as a collection of surface samples or as planes of symmetry. Variations in sampling density (a) and especially pose or shape physiology (b) can present difficulties for these methods.

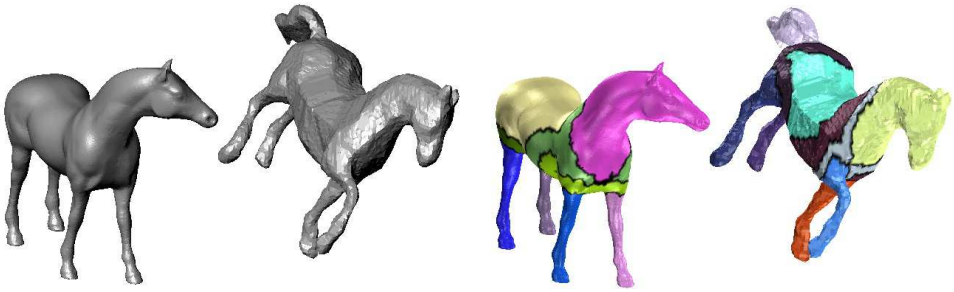
these complications are introduced into the alignment process, it becomes apparent that more information about the input shapes is needed.

1.1.2 The human approach to alignment

When humans align objects, the above concerns play less of a role than the part structure of the shapes. That is, humans tend to decompose the objects into parts first and align core regions to core regions and appendages to appendages. It would be helpful if we could approach the problem from that standpoint.

1.1.3 Part structure and alignment

This leads us to those areas of shape analysis dealing with the semantic, or human-oriented, aspects of shape. Specifically, we need some means to decompose the shape into its semantically relevant part structure. Then we could find the shared



(a) How do we align these horses? (b) Find shared part structure

Figure 1.4: (a) How can we align these horse meshes? Their surface geometries are very different, as are their principal axes and planes of symmetry. What we need is an understanding of their shared part structure. That would allow us to solve for a correspondence of parts which we could use to align them.

part structure between two similar shapes and align them based on that shared part structure (Figure 1.4). This would allow us to account for part articulation and surface variation more robustly. That is, if we can solve for the part correspondence of input shapes, then surface variation between corresponding parts presents less of a problem.

Many powerful techniques exist for this purpose. These methods examine low-level geometric characteristics such as curvature, geodesic distances and dihedral angles to find part boundaries that are then used to decompose the shape into its semantic parts. Our contribution to the literature lies in the fact that we eschew all complex geometric analysis and instead exploit the graph structure of the triangle mesh to decompose the shape. Instead of framing semantic shape analysis as a question of geometric surface characteristics, we approach it as graph problem. The result is a simple yet surprisingly expressive and robust encoding of an articulated shape's part structure into a skeletal graph.

1.2 Hypotheses

It is with this insight that we introduce the two fundamental hypotheses of this work. The first involves the value added of integrating part information into the alignment process:

Hypothesis 1.1 *Integrating the semantic aspects of shape, specifically the part structure of input shapes, into the process of alignment will significantly improve its robustness to surface variation due to sampling density, noise, pose variation and differing physiology.*

The second is less obvious, but it involves our novel approach to the problem of shape decomposition which will be introduced in Chapter 2, given a formal basis in Chapter 3 and fleshed out in Chapter 4:

Hypothesis 1.2 *It is possible to use a direct analysis of mesh topology to encode the shape's part decomposition into a 1D skeletal graph for highly articulated shapes. This approach is simple, efficient, robust to surface variation and consistent for shapes within the same object class.*

We will gather data and test Hypothesis 1.2 first, since it is the basis of our methods for Hypothesis 1.1.

1.3 Benefits of Our Approach

The goal of this work is to formalize and test our novel algorithms for:

(i) encoding the part decomposition of shapes into a graph which can be realized in $\mathbb{R}^3$ as a curve-skeleton (Figure 1.5)

(a) (b)

Figure 1.5: Our topological approach to shape analysis can take this chair shape (a) and encode its part structure into a graph. This graph can then be realized in $\mathbb{R}^3$ as a curve-skeleton (b).

(ii) alignment of shapes based on the graph of (i)

1.3.1 Existing applications

Some common shape applications that would benefit from such a system are:

- **Shape retrieval**: The fundamental task of a 3D shape repository involves returning a model to the user based on some criteria. Examples of repositories would be Princeton's 3D Model Search Engine [70], the Blender Model Repository [17], and the Shape Analysis and Research Project (SHARP) [50]. A query is most commonly text or tag based, but in some systems the query can be in the form of an example shape or even a 2D sketch (Figure 1.6). Our method of shape analysis produces curve skeletons that can be used as the referent for sketch-based queries.

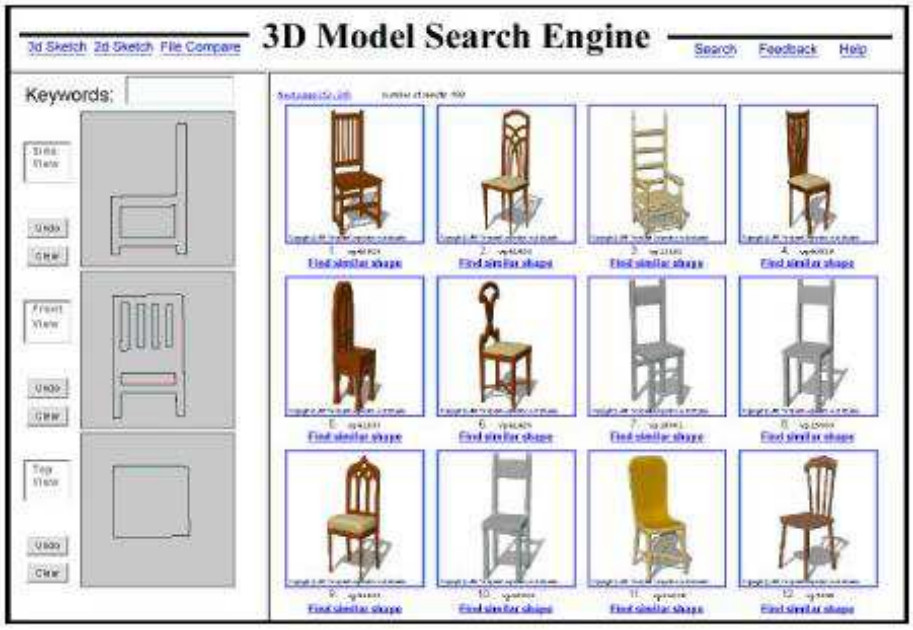


Figure 1.6: The Princeton Shape Database (Shilane *et al.* [70]), an example of a sketch-based shape retrieval system. Our shape analysis methods (Figure 1.5) would be useful to such a system.

- **Determining upright orientation**: The study of how humans determine the upright orientation of objects is an active area of research. Fu *et al.* [33] restricted their solution to the subset of man-made objects, but our methods might be used to generalize their results to all articulated shapes, both man-made and organic.

- **View selection**: Automatic selection of good views for 3D shapes is one critical aspect of browsing large model databases. Those methods based on shape similarity (Yamouchi *et al.* [85]) or those using exemplars (Saleem *et al.* [65]) would benefit from our robust shape analysis.

1.3.2 Simple first-pass solution

The most important benefit of our approach is that it provides solutions to a number of problems in shape analysis using simple graph traversal methods, which we will see in Chapter 4. For the cost of two breadth-first traversals of mesh vertices, it automatically decomposes a mesh into its semantic parts and encodes that part structure into skeletal graph. Many very powerful methods exist to solve these individual problems, but these methods rely on expensive pre-processing or complex geometric analysis. For some applications, this level of detail is necessary. However, given the ever-increasing volume of 3D data, there is a need for an easily implemented and efficient technique that can give an analyst the gist of a set of data. We see our methods as a perfect fit for that kind of first-pass analysis.

1.4 Overview of Thesis

1.4.1 Contributions

In this work, we use an analysis of the topology of triangle meshes to produce a robust system for the comparison and alignment of 3D shapes. The contributions of this work are as follows:

- By viewing the triangle mesh as a topological space, specifically as a simplicial 2-complex, we provide a formal basis for semantic shape analysis within the field of combinatoric topology.
- We develop a novel method for shape analysis based on a modified breadth-first traversal of mesh vertices. The result of this analysis, the breadth-first graph

$\hat{G}$, is an encoding of the semantic part structure of the shape into a graph. This graph can be used to segment the mesh vertices into parts. It can also be used to realize that part structure as a 1D curve-skeleton in $\mathbb{R}^3$.

- By employing the curve-skeletons of input shapes, we develop a novel method of shape alignment that is robust to surface variation and moderate pose articulation.

1.4.2 Outline

The rest of this work is organized as such:

- **Chapter 2**: We discuss the problem of shape alignment and provide some background into the state of the practice in this area. We will give the two major conceptual frameworks for alignment and describe some of the existing methods within those frameworks. We will show how an understanding of semantic part structure would improve the quality of alignments and show how existing methods do not take it into account. Finally, we provide justification for using a simple topological approach to shape analysis and part decomposition, as opposed to a more complex geometry-based approach.

- **Chapter 3**: We lay down a theoretical framework for our shape analysis within the field of combinatoric topology. We show how a triangle mesh can be viewed as a simplicial complex and how operations on simplicial complexes can be used to generate a Discrete Reeb Graph. This can then be used to analyze the mesh

shape's whole topology. We will also discuss some of the external tools that are integral to our methods.

- **Chapter 4**: We present our novel tool for shape analysis based on mesh topology, the breadth-first graph. We describe the process of generating the breadth-first graph based on seed vertices, as well as the issues involved with automatically generating a graph. We then present two algorithms for automatic graph generation.

- **Chapter 5**: We give a description of the requirements and constraints on our experimental data set, and we describe the data set we chose that meets those requirements.

- **Chapter 6**: We provide testing and verification of our breadth-first graph methods' ability to solve two important shape analysis problems, mesh segmentation and curve-skeleton extraction. We test our methods against the benchmark for mesh segmentation of Chen *et al*. and discuss the results. We perform an in-depth analysis of the properties of the curve-skeletons produced by our methods.

- **Chapter 7**: We apply our breadth-first graph to the problem of shape alignment. We first describe how we use the breadth-first graph to solve the relevant subproblem of alignment, correspondence. We then present our complete algorithm for automatic shape alignment. Finally we provide testing and analysis of our alignment method on our data set.

- **Chapter 8**: We provide some concluding remarks about the concepts and methods presented here. We then discuss the validity of the central hypotheses of our work. Finally, we give some future directions for research.

CHAPTER 2

PRIOR WORK

This chapter provides a summary of the problem of shape alignment and the prior work in that area. Our intent is twofold. First, we want to show how our problem is distinct from those being solved by most existing methods, namely that we treat input shapes as collections of interrelated semantic parts rather than as monolithic geometric entities. Second, we want to show how our work intersects with other areas of shape analysis in order to facilitate the integration of semantic part structure into the problem of shape alignment.

In Section 2.1, we discuss the general problem of shape alignment, present the major methodological themes and discuss the distinctions between our problem and those solved in existing literature. In Section 2.2, we look into the importance of understanding the semantic part structure of shapes when aligning them. We integrate part structure into shape alignment by relying on two related areas of shape analysis, mesh segmentation and skeleton extraction, so we also discuss prior work in those areas. In Section 2.3, we look at a particularly important result of Isenburg *et al*. [42] that inspired our original research and provided us with the imprimatur for our approach to part-aware shape alignment. We also discuss our reasons for building from Isenburg's results.

2.1 Shape Alignment

2.1.1 Problem Definition

Given two input shapes $\mathbf{P}$ and $\mathbf{Q}$, we want to find the rigid transformation α such that $\alpha(\mathbf{P})$ and $\mathbf{Q}$ are semantically aligned. We are purposefully imprecise about the meaning of semantic alignment but will develop an understanding of this term through this chapter and especially in Chapter 7. $\mathbf{P}$ and $\mathbf{Q}$ are complete shapes, described as triangle meshes M_P and M_Q, respectively. In general, they are not the same shape, but they are shapes from the same object class, *i.e.* having the same or similar part structures. So, the problem can be stated as the alignment of triangle surface meshes that are complete expressions of the same object class.

Since $\mathbf{P}$ and $\mathbf{Q}$ can be different shapes (*e.g.* the same shape in different poses or different shapes of the same class) some shape pairs cannot be perfectly aligned. However, it can be assumed that there exists a transform α that aligns the two objects well in a semantic sense. That is, there is some α that aligns $\mathbf{P}$ and $\mathbf{Q}$ in a way that is sensible to a human observer. Our goal is to find that α in a way that is robust to noise and object part pose variation between input shapes and achieves consistent results among shapes of the same object class. The details of how the quality of α is calculated will be discussed in Chapter 7 on alignment, but it will involve the manual selection of semantically relevant landmarks and measuring how well α aligns those landmarks.

There are two dominant methodologies for aligning two input shapes $\mathbf{P}$ and $\mathbf{Q}$. *Surface registration* treats $\mathbf{P}$ and $\mathbf{Q}$ as sub-surfaces of some ground truth surface

(a) (b) (c) (d)

Figure 2.1: Surface registration in action. (a) Range scans are taken of the original object model, producing scanned surface strips of the model (b). These strips are registered (c) to produce a final model (d).

and solves for α based on points of overlap (Figure 2.1). *Orientation normalization* instead solves for the local coordinate frame of $\mathbf{P}$ and $\mathbf{Q}$ and then aligns those frames to the global coordinate system.

2.1.2 Surface Registration

If $\mathbf{P}$ and $\mathbf{Q}$ are known to be constructed from samplings of the same object, then they are related by some rigid transform. The problem of finding this transform is an active area of research known as *surface registration*. These techniques are commonly used for the problem of multi-view registration, *i.e.* the alignment of surfaces reconstructed from multiple passes of a 3D range scanner on an object, the end result being a complete mesh approximation of that object. A famous application of multi-view registration is the *Digital Michelangelo Project* of Levoy *et al.* [49] (Figure 2.1).

Solutions to this problem in the literature generally come in two forms. *Local registration* techniques assume that the input shapes are approximately aligned and search only for the closest local minimum of the distance function. *Global registration* techniques make no assumptions about initial placement and seek the global minimum of the distance function.

Local registration

Given some initial guess of rigid transform α, this class of registration methods finds the local minimum of the distance equation ($\min_\alpha d^2(\alpha(\mathbf{P}), \mathbf{Q})$). Iterative Closest Point (ICP), first formalized by Besl and McKay [11] and Chen and Medioni [25], is a well-known local registration method. Rusinkiewicz and Levoy [64] modified the name to Iterative *Corresponding* Point and defined ICP to be an algorithm in three stages: given some initial position of input point sets P and Q, find point correspondences between the sets, compute the transform for P that minimizes some error metric and apply the transform to P iteratively until some convergence condition is met. The most common types of algorithms to compute the aligning transform are the closed form solutions based on least-squares. One of these least-squares solutions, that of Arun *et al.* [7], is discussed in more detail in Section 3.3. Both the selection of which points from the input meshes make up P and Q and the relative weighting of point correspondences varies (see [64]). Local registration methods are sensitive to the initial choice of α. If the initial placement is poor, ICP will converge to an incorrect alignment.

Global registration

Global surface registration algorithms are not dependent on initial placement, instead of searching directly for the globally optimal α. Wolfson *et al*. [83] use the low dimensionality of transform space to solve directly for α. Others solve for the globally optimal set of corresponding points and compute the aligning transform from that. Gelfand [35] uses shape descriptors based on integral volume at each sample point to identify a small number of unique feature points which are then used as candidates for correspondence. Aiger *et al*. [4] use congruent 4-point sets to solve for the correct alignment in the presence of significant noise.

Surface registration and shape alignment

The surface registration techniques most amenable to our problem are those that reduce surface alignment to the point correspondence problem. In the case of local registration techniques like ICP, the correspondence problem is solved through iterative refinement. For global registration methods of this type, it is solved through an optimized search of correspondence space. Section 2.2 will highlight the importance of integrating the part structure of input shapes into the alignment process. If we can frame that integration of part structure within the context of point correspondence, then we can leverage the powerful tools of the surface registration literature.

The problem with using these methods directly is that they focus *exclusively* on surface samples when building point correspondences. Input surfaces are expected to be incomplete overlapping sub-samplings of a static ground-truth model shape,

and they are aligned in order to build a complete mesh approximation of the original model. In our problem of shape alignment, there is no single ground truth model (*i.e.* there are two).

There are techniques that deal with some of these concerns. Anguelov *et al.* [6] use a joint probabilistic model over all point correspondences to handle shifts in object pose, but they assume that input meshes are of the same object in different poses. Chang and Zwicker [21] transform the alignment problem to one of discrete labeling of object parts to handle both shifts in pose and missing surface data in input shapes, but they also require significant surface similarity. As such, it is unclear whether either of these techniques would be suitable for our problem.

2.1.3 Orientation Normalization

Instead of thinking of each input shape as a subset or slightly modified version of a ground truth model as in surface registration, *orientation normalization* finds each input shape's local coordinate system and aligns both shapes to the global coordinate system. It should be noted that in the literature, the more common name for this class of methods is *pose normalization*, where the qualifier "pose" is defined as the orientation of the whole shape in three dimensional space. We choose to modify that qualifier to "orientation" so as not to confuse the reader with our definition of pose (*i.e.* the interrelated distribution of individual parts of an individual shape).

There are two major types of solution to the problem of orientation normalization. The first type are those based on principal component analysis (PCA). The

second type are those that transform the alignment problem to some function space and solve directly for the optimal rotation within that space.

PCA

The most common method of orientation normalization uses principal component analysis (PCA) [70, 86] as its basis. The principal components of a point cloud are the eigenvectors of its covariance matrix. They are the principal axes of its best fitting ellipsoid. By aligning the principal components of the mesh vertices to the coordinate axes, each input shape can be rotated to a consistent orientation. If some uniformly distributed subset of mesh vertices is chosen, this method is very efficient and provides an easily implemented solution. However, it is limited to input shapes whose principal axes are well defined. Isotropic models (*i.e.* those with strong axial symmetry such as vases) and amorphous models (such as plants or molecules) give rise to poor alignments [45]. PCA also ignores part structure, as it is based solely on the best-fitting ellipsoid.

Solutions have been posed to the problem of axial symmetry within PCA-based alignment algorithms. Podolak *et al.* [61] introduce the Planar Reflective Symmetry Transform (PRST), which encodes the reflectional symmetry of a shape with respect to all planes in 3-space. With this, they were able to define two new concepts that help express shape symmetry within the global coordinate system, *the center of symmetry* and *the principal symmetry axes.* The principal symmetry axes are the normals of the planes of maximal symmetry of an object, and the center of symmetry is the point at

which the three planes meet. Chaouch and Verroust-Blondet [22, 23] use these ideas of planar symmetry along with the continuous PCA (CPCA) technique of Vranic *et al.* [82] to achieve positive results.

Function-space

A more recent approach involves transforming each input shape to some functional space on either the unit sphere or in 3D and finding the rotation that minimizes the distance between those functions. A brute-force search over the space of rotations will find the optimal rotation. However, even using signal processing techniques to decrease running times, this method is still too slow for alignment tasks involving large shape databases. To address this issue, Makadia and Daniilidis [52] parameterize the space of rotations into the axis and angle of rotation, solving first for the axis using the Fast Spherical Harmonic Transform [56] and then solving for the angle via Fast Fourier Transform. Kazhdan [45] first computes an axially symmetric descriptor of each input shape within a spherical function-space, rotates them so that the axis with largest symmetry maps to the z-axis, then solves for the angle relating them similarly to Makadia. Martinek and Grosso [55] implement such an approach on the GPU to achieve dramatically decreased run-times.

Orientation normalization and shape alignment

The essence of normalization is to think of each shape as having a canonical orientation within some space. This orientation is defined either by the principal axes

or by the projection of the mesh vertices into some function-space. Once this canonical orientation of each input shape is known, their orientations are made to agree. The primary strength of this approach is that it is designed to accommodate complete input shapes (unlike surface registration) and does not depend on the existence of a ground truth model.

However, like surface registration, normalization techniques view the shape as a geometric monolith. The primary issue with using normalization for our problem of shape alignment is their reliance on well-defined principal axes that are similarly defined among input shapes. Since changes in the poses of individual object parts necessarily affect the principal axes of a shape, it is unclear how these techniques would behave in the presence of such changes. Indeed, it would be difficult to modify these techniques to take their part structure into account.

2.2 Shape Parts and Alignment

In this section, we wish to lay out how we will deviate from existing literature by integrating the part structure of input shapes into the process of shape alignment. The predominant approach to alignment in the literature treats each shape as a monolithic geometric entity, devoid of an articulated part structure. If the part structures of input shapes are taken into account, as in the case of Anguelov [6] and Chang [21], it is specifically to deal with pose variations of a single object. We wanted to approach the problem differently.

A mesh that encodes a whole shape is more than just a collection of sample points with local connectivity. What makes it a mesh, as opposed to a polygon soup, is the fact that its vertices, edges and faces are themselves connected to each other. Starting from the explicit local connectivity at a single vertex and expanding outwards in scope, a global macro-structure of parts emerges. That part structure is as relevant to alignment as the surface geometry. Indeed, that part structure might be *more* important than the surface geometry when trying to align shapes with significant surface dissimilarity. When aligning objects of the same class whose surface geometry is different, it is more natural to think in terms of aligning their part structure than aligning their surfaces.

Bronstein *et al*. [20] explore this idea when dealing with the similarity of shapes under deformation. They break the problem of similarity into two classes, intrinsic and extrinsic similarity. Extrinsic similarity involves their layout in Euclidean space. Intrinsic similarity involves intrinsic properties of shapes, *i.e*. those properties that are invariant to object deformation. In our case, we rely on an intrinsic property of input shapes, namely their part structure and the internal distances between those parts.

There are two related areas of research that are concerned with finding and representing an object's part structure. The first, *mesh segmentation*, deals with decomposing the shape into disjoint subsets. The second, *skeleton extraction*, attempts to encode the shape's structure by a 1D curve. We will discuss each of these topics, presenting the works in the literature that are relevant to our understanding of part structure. Then we will discuss how they apply to our goal of shape alignment.

2.2.1 Mesh Segmentation

In order to integrate part structure into the process of shape alignment, we must first have some means of decomposing input shapes into parts. Mesh segmentation is an active area of research concerned with exactly that. Given a mesh $M = \{V, E, F\}$ of vertices $V = \{v : v \in \mathbb{R}^3\}$, edges $E = \{(v_i, v_j) : v_i, v_j \in V\}$ and faces $F = \{(v_i, v_j, v_k) : v_i, v_j, v_k \in V, (v_i, v_j), (v_j, v_k), (v_k, v_i) \in E\}$, a segmentation of M is the set of sub-meshes $S_M = \{M_0, M_1, ..., M_{k-1}\}$ induced by a k-way partition of either V, E or F. See Shamir [67, 68] for a thorough treatment of the problem of boundary mesh segmentation, where a boundary mesh is a 2D surface embedded in 3D. Both Shamir and Attene *et al.* [9] separate solutions to the segmentation problem into two types defined by their objective: geometric segmentation and semantic segmentation ([68] refers to them as patch-type and part-type, respectively). Geometric (patch-type) techniques seek to decompose M into disk-like patches meeting low-level geometric criteria such as planarity, size or convexity, and semantic (part-type) techniques seek to decompose M into visually or semantically meaningful sub-parts, such as limbs or appendages. Our goal is an understanding of the overall semantic structure of parts, so we will limit our discussion to part-type solutions. See Shamir and Attene [9], for information on patch-type techniques.

Prior work

Solutions to the problem of decomposing a mesh into semantically meaningful parts have several forms. These forms can be classified by the metrics used to deter-

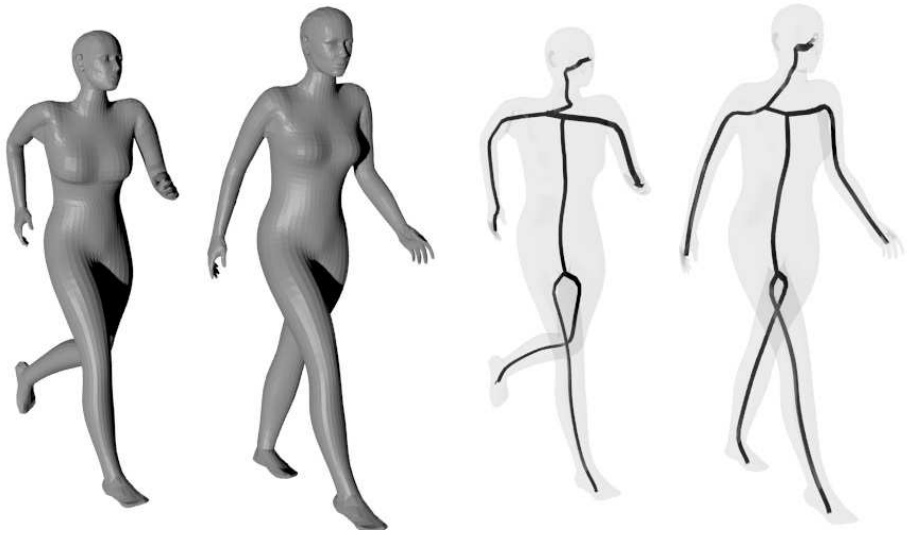
mine part boundaries. This list of metrics is by no means exhaustive, or indeed fully separable, but it gives the flavor of what the dominant solutions involve. See [3,9,67] for a review of existing methods. Chen *et al.* [24] also provide formal benchmarks for comparison of segmentation techniques.

Salience Many results in the segmentation literature deal with analyzing the surface curvature characteristics of a mesh to find part boundaries, in an attempt to segment the mesh the way a human might. This is based on a seminal result in cognitive psychology by Hoffman and Singh [40] that showed that humans tend to place visually salient part boundaries along lines of negative minimum curvature. Lee *et al.* [48] took this idea and developed a mesh scissoring method using geometric snakes selected under part salience conditions. The "Tailor" technique of Mortara *et al.* [58] studies how salient curvature features evolve when the surface is intersected with expanding spheres. Attene *et al.* [8] use salient features for fitting primitives that approximate mesh parts and form the basis for mesh segmentation. Golovinsky *et al.* [37] propose the method of normalized cuts that starts with each face in its own segment and hierarchically combines segments in a bottom-up fashion based on area-normalized cut cost: the sum of each segment's perimeter divided by its area, where perimeter is weighted by concavity so that boundaries are along salient concave seams.

Geodesic distance Other shape-centered solutions take overall mesh connectivity into account by using globally defined properties like geodesic distance. Katz and Tal [44] use geodesic distance as the basis for hierarchically separating a mesh into fuzzy clusters, or clusters of mesh faces with fuzzy boundaries, which are

then cut along edges with highly concave dihedral angles. Katz *et al.* [43] use multi-dimensional scaling to transform the mesh into a canonical form where Euclidean distances between points are similar to geodesic distances, label feature points based on this transform, and segment based on the relationship of those feature points to the "core" region of the mesh. Agathos *et al.* [2] build on the core-appendage premise of [43] to construct a protrusion-oriented segmentation. Lien and Amato [51] use a divide-and-conquer approach to combine skeletonization and decomposition of the mesh into a single interdependent process. Tierny *et al.* [81], in a continuation of their skeleton extraction results of [80], use a Reeb graph of the mesh based on geodesic distances with respect to mesh features which, in concert with curvature considerations, is used for segmentation. Similarly, Berretti *et al.* [10] use an average geodesic distance function to induce the Reeb graph that is used to segment the mesh.

Medial and functional simplification Some methods rely on shape simplifications based on medial structures or other functional representations. de Goes *et al.* [29] use medial structures based on diffusion distance for segmentation. In addition to normalized cuts mentioned above, Golovinsky *et al.* [37] also propose a hierarchical segmentation method that starts with a decimated version of the mesh in a single segment and uses randomized minimum cuts to recursively split the mesh into binary segments. Shapira *et al.* [69] use the "Shape Diameter Function", which approximates the diameter of the object's volume in the neighborhood of a point on the surface, to apply energy minimization techniques to cluster faces in a way that maximizes boundary smoothness and location along concave seems.



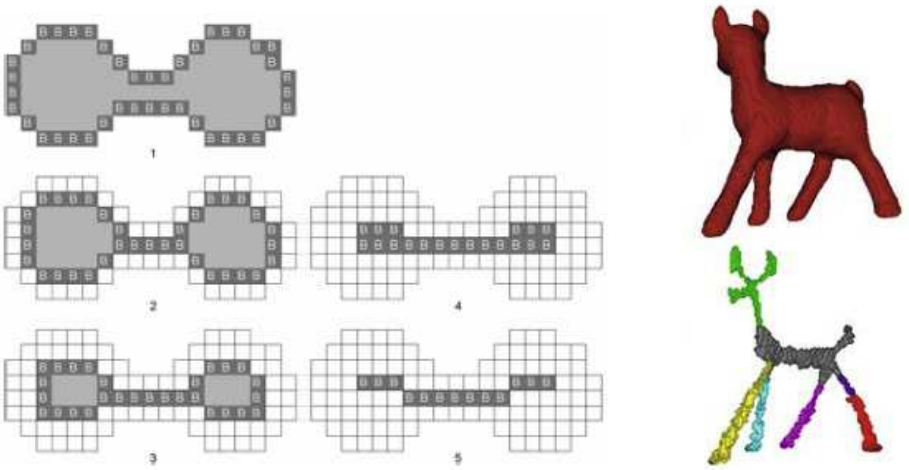
(a) (b)

Figure 2.2: Usefulness of curve-skeletons in shape alignment. Given two humanoid shapes (a) in different poses, we need some way to represent their individual parts (*e.g.* arms and legs) so that the parts' lengths are encoded and preserved under deformation. (b) Curve-skeletons provide a way to do so.

2.2.2 Skeleton Extraction

In order to properly integrate part structure into alignment, we need more than just the raw shape decomposition that segmentation provides. We also need some understanding of the connectivity and relative distances of parts within the overall shape context. Take the example of a humanoid shape (Figure 2.2). Given that an arm can be straight or bent, we need some way of representing that arm such that the distance from shoulder to fingers is both encoded and preserved under deformation. A straight-line representation of the arm would accomplish this. The area of shape analysis concerned with producing linear simplifications is *skeleton extraction*.

In shape analysis, the term *skeleton* can mean a number of things. It can refer to the medial axis of 2D objects [18] or the medial surface of 3D objects [5]. The better definition for us would be the *curve-skeleton* in the sense of Dey and Sun [30],



(a) Thinning process in 2D (b) 3D example

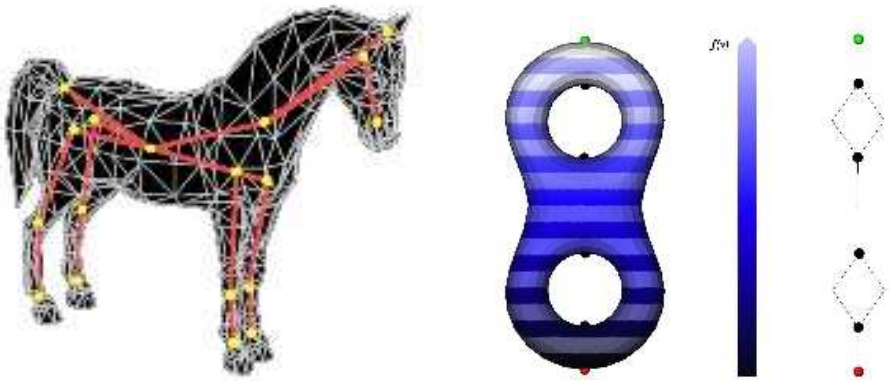
Figure 2.3: (a) The thinning process on an example 2D shape. Boundary points are marked at the beginning of each iteration and then removed if they are simple (*i.e.* their removal does not change the topology). (b) An example of this process on a 3D shape (Siddiqi [73]).

that is, a 1D simplification of a 3D surface. See Biasotti *et al.* [12, 13] and Cornea *et al.* [28] for a review of curve skeletons in shape analysis.

Prior work

According to Cornea [28], solutions to the problem of skeleton extraction come in various forms. We will focus our attention on two of those forms, *volumetric thinning* and *geometric* methods. We chose these forms since they are well-suited to use on polygonal mesh data.

Volumetric thinning These methods require a discrete volumetric (*i.e.* voxelized) representation of the polygonal surface, which can be generated using a voxelization technique such as that of Noorudin and Turk [60]. Volumetric thinning methods then remove voxels iteratively from the boundary of the shape until some



(a) Segmentation (b) Reeb graph

Figure 2.4: (a) Example of segmentation-based curve skeletons of Katz *et al*. [44]. (b) Example of the use of Morse functions on a surface to produce a curve-skeleton [80]. The critical points in the height function correspond to junctions in the Reeb graph, which becomes the skeleton.

thinness condition is met. At each step, boundary voxels are tested against some set of topology preserving conditions as well as conditions to prevent excessive shortening of curve-skeleton branches. See Figure 2.3 for an example. This method is also referred to as *morphological thinning* [19, 46, 53]. See also Gagvani and Silver [34] for a review of volumetric thinning. Svensson *et al*. [77] thin from a voxelization of the medial surface while preventing the removal of junctions between surfaces. Sundar *et al*. [76] work from the results of [34] to build curve-skeletons for shape matching and retrieval. Cornea *et al*. [27] use the generalized potential field of Chuang *et al*. [26] to guide the thinning process, producing topology-preserving skeletons that capture part structure. Siddiqi *et al*. [73] use a voxelization of medial surfaces to produce a directed acyclic graph of components for shape retrieval (Figure 2.3(b)).

Geometric methods These curve-skeleton methods are those that apply specifically to objects represented by polygonal meshes or point clouds. One well-

developed class of geometric methods are those associated with the medial surface. Siddiqi and Pizer [72] is an excellent reference for these medial representations. Katz and Tal [44] use their segmentation to extract 1D curve-skeletons by representing the segmentation hierarchy as a tree graph embedded in 3D with individual tree nodes at the centroids of segment boundaries (Figure 2.4(a)). Tierny *et al.* [80] use a Reeb graph to construct the curve-skeleton of a mesh, as does Biasotti *et al.* [14] (Figure 2.4(b)). Zheng *et al.* [87] use the curve skeleton extraction technique of Tagliasacchi *et al.* [78] to construct consensus skeletons of point clouds. Agathos *et al.* [1] apply a graph-based representation of articulated meshes to the problem of shape retrieval, restricting to those meshes with a core-appendage topology.

2.2.3 From Parts to Alignment

If the part structure of a shape can be represented as a graph of curvilinear segments whose segments connect at *junction* points for part boundaries and terminate at *cap* points for part tips, then the problem of surface alignment would reduce to the alignment of those junction and cap points. By finding the correct correspondence between those points, we could solve for the rigid transform that aligns them. That is, given a curve-skeleton whose topology is a direct encoding of part structure of each mesh, we have enough information to perform shape alignment in a general form.

What we need is a simple, robust algorithm for skeleton extraction that produces skeletons with junction and cap points for meshes of the same object class

that correspond to part structures shared by those meshes. This algorithm must be invariant to rigid transformations. It must also produce a minimal skeletal structure. That is, we only want it to capture the coarsest level of part decomposition. This will minimize the number of junction and cap points in the skeleton and thus the correspondence space for our alignment algorithm.

The best example of such a simple, robust algorithm in the literature is the affine-invariant skeleton of Mortara and Patane [57]. It produces a skeleton that meets the first criteria of invariance to rigid transform, but because these skeletons capture all protrusions in the mesh shape, the second criteria of minimal structure is less promising. The Reeb graph curve skeletons of Biasotti *et al.* [14] and Tierny [80] would suffice, but the Morse functions used in these techniques require expensive geometric computation on the mesh surface. The consensus skeletons of Zheng *et al.* [87] might be an option, but their technique assumes that the input shapes are of the same object in differing poses, which is overly restrictive for our problem.

In the next section, we discuss the reasons for looking elsewhere for a mesh segmenting curve-skeleton extraction algorithm.

2.3 Connectivity Shapes of Isenburg *et al.*

Early in our search of the literature, we were intrigued and inspired by the work of Isenburg *et al.* [42]. They found that there is an astounding amount of shape part information embedded in the pure connectivity of the mesh. They demonstrated this idea by interpreting the natural geometry of a connectivity as an embedding in

(a) Connectivity shapes (b) Our skeleton

Figure 2.5: Connectivity shapes of Isenburg *et al.* [42]. (a) A startling amount of the geometric information is embedded in the connectivity of this cow mesh. (b) Our skeleton is similarly based on the connectivity of the mesh.

space of uniform edge length. That is, they throw out the actual geometry of a mesh's vertices and place them in space such that the edges between vertices have roughly equal length. The result, which they refer to as connectivity shapes (Figure 2.5), have a cartoonish appearance and bear little resemblance to the original model in terms of fine detail, but they contain a considerable amount of information about the mesh part structure.

Others have applied these ideas in other areas of mesh analysis, which gave us additional confidence that we could use it to our ends. For example, Sorkine and Cohen-Or [74] use mesh connectivity along with a sparse set of control points to construct a fair surface reconstruction of an original mesh geometry. Sorkine *et al.* [75] then generalize this idea to approximate the geometry of a mesh using a linear combination of basis vectors that are generated from a connectivity mesh.

2.3.1 Connectivity-based Approach to Part Extraction

We seek to extract part information directly from the connectivity, without a rescaling of edge lengths, and encode it into a more compact form. We would then use this new connectivity-based representation in our end goal of shape alignment. Our goal is aided by such an approach for a few reasons.

- The approach is independent of the initial position of input shapes. Sensitivity to initial position is an inherent difficulty with many existing alignment methods (see Section 2.1 for more details).

- High-frequency shape characteristics would be ignored (*i.e.* smoothed out) in favor of large-scale parts and appendages. This would dramatically decrease the effect of noise. Having fewer parts to align means a smaller correspondence search space, thus a more efficient algorithm.

- The part topology would be largely invariant to pose.

In the next chapter, we will provide a basis for this connectivity-based approach within the field of combinatoric topology. By viewing the mesh as a specific kind of topological space, we can apply primitive operations on that space to form a scalar-value function, and this function will allow us to frame our analysis within the well-established area of Morse theory. In Chapter 4, we will present our algorithm for shape analysis and part extraction.

CHAPTER 3

DEFINITIONS AND TOOLS

In this chapter, we will provide an explanation of those concepts and techniques fundamental to our topological approach to mesh shape analysis and alignment. The intent is not to give an exhaustive treatment but to provide a quick reference on how they will be used throughout this work. In order to provide some context for our approach, Section 3.1 explains the distinction between topology and geometry with respect to mesh input. Section 3.2 establishes a formal basis in existing combinatorics for our approach to mesh shape analysis.

We will also introduce some of the external tools and software that will be used extensively both in our methods and in testing our methods. Section 3.3 introduces and explains the least-squares method of Arun to find the distance-minimizing transform of a point correspondence, a technique that is integral to our alignment solution. Section 3.4 discusses the Generalized ICP technique of Segal *et al*. and how we use it as a baseline for good alignment performance.

3.1 Geometry and Topology

In this work, we use the terms *geometry* and *topology* when discussing mesh input. These terms are very broadly defined, so for the sake of clarity, we would like to define their usage here. According to Munkres [59], topology is an area of math-

ematics defined by the search for homeomorphisms between spaces, *i.e.* continuous bijective maps, with continuous inverse, mapping one space to another. Geometry is another branch of study concerned with the questions of size and distance between entities within a particular field, such as $\mathbb{R}^2$ or $\mathbb{R}^3$. In this work, we are dealing with a particular kind of topological space geometrically embedded in the field $\mathbb{R}^3$, the triangle surface mesh. Moreover, this mesh is 2-manifold, or locally homeomorphic to a plane. Our use of these terms is confined to that context.

So, given the context of a 2-manifold triangle surface mesh, when we speak of geometry, we are referring to the location in $\mathbb{R}^3$ of the vertices V. When we speak of topology, we are referring to the edges E and faces F, *i.e.* the *connectivity* of the vertices (we will often use the two terms interchangeably). It is this combination of connectivity and placement that determine the surface characteristics of the shape approximated by M. Thus a mesh is defined by both its geometry and its topology.

We mention this dichotomy between geometry and topology in order to highlight the differences between the methods proposed in this work and prior work in shape analysis. Most existing techniques focus on the specific geometry of a mesh and only use the mesh topology indirectly in the recovery of geometric surface characteristics (*e.g.* curvature, dihedral angles). Inspired by the results of others (Section 2.3), we wished to know what the topology could tell us about the shape. In the following section, we will give a brief discussion of how the mesh can be expressed and analyzed as a purely topological entity.

3.2 Mesh as Topology

It is the goal of this work to demonstrate the possiblity of performing meaningful analysis of shape based on mesh topology. In Section 3.1, we saw that topology deals with continuous bijective maps between topological spaces. Therefore, we need to be able to describe the mesh as a topological space. In this section, we will show that under certain conditions, mesh input is equivalent to a well-known topological space, the simplicial complex, giving our topological approach a basis in existing theory.

We will break our discussion into three parts. As background, we first define what a simplex and a simplicial complex are. We then discuss some important operations on simplices and simplicial complexes. Finally, we show how careful constraints on mesh input allow us to transform mesh analysis into the analysis of simplicial complexes. This provides the formal basis for our analytical methods of Chapter 4.

The foundational work on algebraic topology is Munkres [59]. Edelsbrunner [31] is another standard work on computational and combinatoric topology. For a more thorough review of topics given here, the reader is referred to these texts.

3.2.1 Simplex and simplicial complex

Simplex

A *simplex* is a generalization of geometric concepts such as points, lines and triangles. A set of points S is affinely independent (*a.i.*) if no point in S is an affine combination of the other points in the set. A simplex of dimension k, or *k-simplex* σ,

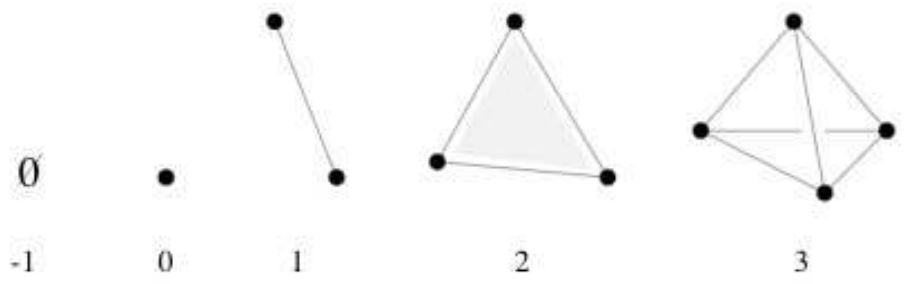


Figure 3.1: **Simplices** of various dimensions. The simplex of dimension -1 is the empty set. A 0-simplex is a vertex, 1-simplex is an edge, 2-simplex is a triangle and 3-simplex is a tetrahedron. A triangle surface mesh thus can be thought of as a pure simplicial 2-complex, *i.e.* a simplicial complex composed purely of triangles.

is the convex hull of $k+1$ *a.i.* points S, $\sigma = \text{conv}(S)$ [31]. We denote the dimension of σ as $\text{Dim}(\sigma)$. Figure 3.1 demonstrates the simplices of dimension -1 through 3. Every subset of points $T \subseteq S$ of σ is *a.i.*, so the convex hull of T is itself a simplex, $\tau = \text{conv}(T)$. This simplex τ is called a *face* of σ, denoted $\tau \leq \sigma$ (as opposed to $\tau \in \sigma$, which is a characteristic of simplicial complexes below). So, in the case of the triangle $S = \{v_i, v_j, v_k\}$, the 2-simplex (or face simplex) defined by S contains:

- $\emptyset$ and S as *improper faces*
- the 1-simplices v_i, v_j, v_j, v_k and v_k, v_i and the 0-simplices v_i, v_i, or v_k as *proper faces.*

Simplicial complex

A *simplicial complex* is a set of simplices $\mathcal{K}$ that satisfies the following conditions:

(i) If $\sigma \in \mathcal{K}$ and $\tau \leq \sigma$, then $\tau \in \mathcal{K}$

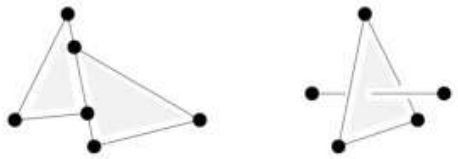


Figure 3.2: An example of a set of simplices which are not a simplicial complex (they fail condition (ii)). On the left, the triangles meet along an edge not shared by both. On the right, a triangle and an edge meet at a vertex not in either.

(ii) If $\sigma_1, \sigma_2 \in \mathcal{K}$ then $(\sigma_1 \cap \sigma_2) \leq \sigma_1$ and $(\sigma_1 \cap \sigma_2) \leq \sigma_2$

Informally, condition (i) requires that for any simplex in $\mathcal{K}$, its component faces must also be in $\mathcal{K}$. For example, if $\mathcal{K}$ contained a triangle 2-simplex, it must also contain all three of its edge 1-simplices and vertex 0-simplices. Condition (ii) states that for any two simplices in $\mathcal{K}$, their intersection must be a face shared by both. An example of when this condition is not true is given in Figure 3.2.

A *simplicial k-complex* $\mathcal{K}$ is a simplicial complex where $\text{Dim}(\sigma) \leq k, \forall \sigma \in \mathcal{K}$. A *pure* simplicial k-complex $\mathcal{K}$ is one where every simplex of dimension less than k is a face of a simplex of dimension k.

3.2.2 Operations on simplicial complexes

A *subcomplex* of $\mathcal{K}$ is a subset of $\mathcal{K}$ that meets the conditions of a simplicial complex. All subsets of $\mathcal{K}$ satisfy condition (ii). In order to enforce condition (i), we must find the closure of that subset.

Closure

The *closure* of $\mathcal{L} \subseteq \mathcal{K}$, is the minimal simplicial subcomplex of $\mathcal{K}$ containing all faces in $\mathcal{L}$:

$$\mathrm{Cl}(\mathcal{L}) = \{\tau \in \mathcal{K} : \tau \leq \sigma \in \mathcal{L}\} \tag{3.1}$$

In Figure 3.3(a), a triangle τ_{ijk} and an edge τ_{lm} face are chosen, $\mathcal{L} = \{\tau_{ijk}, \tau_{lm}\}$, and the closure of $\mathcal{L}$ is:

$$\{\emptyset, \tau_i, \tau_j, \tau_k, \tau_l, \tau_m, \tau_{ij}, \tau_{jk}, \tau_{ki}, \tau_{lm}, \tau_{ijk}\}$$

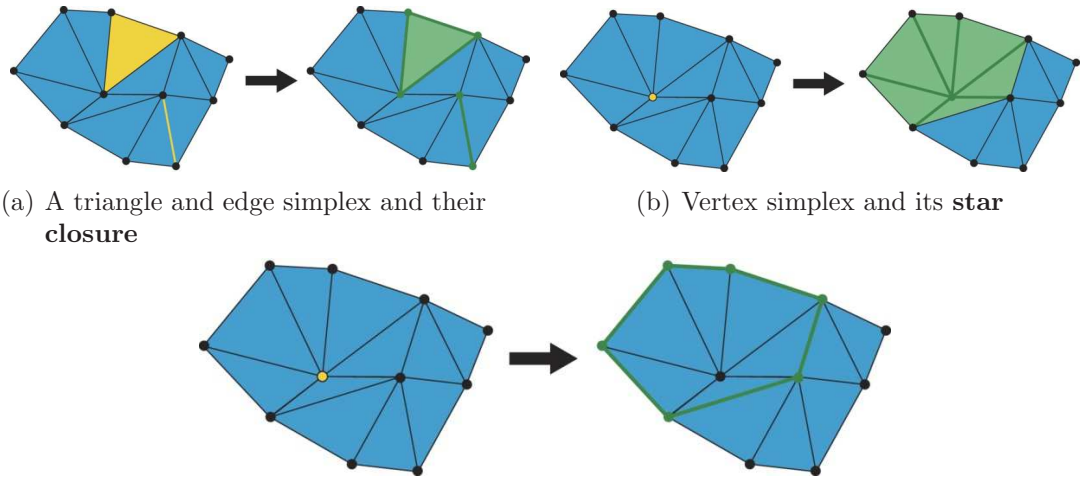
where the indices of each face simplex correspond to the vertices of which they are composed.

Star

Let τ be a simplex in $\mathcal{K}$. The *star* of τ is the set of all simplices in $\mathcal{K}$ that contain τ.

$$\mathrm{St}(\tau) = \{\sigma \in \mathcal{K} : \tau \leq \sigma\} \tag{3.2}$$

Figure 3.3(b) demonstrates this with a vertex simplex τ. The star of this vertex contains the edge and face simplices that connect to τ but *not* the outermost edges and vertex simplices. Because of this, the star is usually not closed, *i.e.* a subcomplex of $\mathcal{K}$. If we were to find the closure of the star of τ, then those outermost edge and vertex simplices would be included, which leads us to the last important operation.



(a) A triangle and edge simplex and their **closure**

(b) Vertex simplex and its **star**

(c) Simplex and its **link**

Figure 3.3: Operations on simplex subsets of simplicial complexes. (a) The **closure** of a subset of faces $\mathcal{L} \subseteq \mathcal{K}$ is the smallest simplicial subcomplex of $\mathcal{K}$ that contains each face in $\mathcal{L}$. (b) The **star** of a particular simplex τ is all simplices in $\mathcal{K}$ that contain τ. (c) The **link** of τ consists of the faces of simplices in the star of τ that don't intersect τ.

Link

The **link** of some simplex τ is the set of simplices in the closure of the star of τ that do *not* contain τ.

$$\mathrm{Lk}(\tau) = \{\sigma \in \mathrm{Cl}(\mathrm{St}(\tau)) : \sigma \cap \tau = \emptyset\} \tag{3.3}$$

We see the link of vertex simplex τ in Figure 3.3(c). Whereas the star of τ did not include the outermost vertex and edge faces, the link consists solely of those missing faces. What's more, the link is always a simplicial subcomplex of $\mathcal{K}$.

3.2.3 Mesh as simplicial complex

In general, a *polygon mesh* is some collection of vertices, edges and faces that defines the shape of some polyhedral object. This might be an approximation of some continuous function or reconstructed from sample points taken from the scan of a real object. In this work, we will deal only with a specific kind of polygon mesh, the triangle mesh.

A *manifold triangle mesh* $M = \{V, E, F\}$ is defined by its vertices $V \subset \mathbb{R}^3$, edges $E \subset V \times V$ and triangles $F \subset V \times V \times V$ such that the surface defined by M is 2-manifold. Formally, every point on the surface defined by M has a neighborhood homeomorphic to an open subset of $\mathbb{R}^2$. That is, at every vertex v, the surface can be locally "flattened" to a plane without tearing or gluing.

We can see right away that the manifold triangle mesh, or simply *mesh*, is composed of simplices in that it contains vertices (0-simplex), edges (1-simplex) and triangles (2-simplex). What remains to be seen is whether it can be described as a simplicial complex. Suppose we define mesh M as $\mathcal{M} = \{\mathcal{V}, \mathcal{E}, \mathcal{F}\}$ where:

$$\mathcal{V} = \{\tau_i = (\emptyset, v_i) : v_i \in V\}$$

$$\mathcal{E} = \{\tau_{ij} = (\emptyset, \{v_i, v_j\}) : \{v_i, v_j\} \in E, v_i \in \tau_i, v_j \in \tau_j, \tau_i, \tau_j \leq \tau_{ij}\}$$

$$\mathcal{F} = \{\tau_{ijk} = (\emptyset, \{v_i, v_j, v_k\}) : \{v_i, v_j, v_k\} \in F, v_i \in \tau_i, v_j \in \tau_j, v_k \in \tau_k,$$

$$\tau_i, \tau_j, \tau_k, \tau_{ij}, \tau_{jk}, \tau_{ki} \leq \tau_{ijk}\}$$

$\mathcal{M}$ meets condition (i) of a simplicial complex, since for any vertex, edge or triangle in $\mathcal{M}$, its component faces are also contained in $\mathcal{M}$. Condition (ii) is satisfied by the fact that the mesh M is a 2-manifold surface. That is, the surface local to every mesh vertex v, *i.e.* those faces adjacent to v, is a complete disk of triangles. In particular, if E_v and F_v are the set of edges and faces adjacent to v, then $|E_v| = |F_v|$ and every edge in E_v is adjacent to exactly two faces in F_v. The intersection of two faces corresponds to some edge or some vertex.

The highest dimension of any simplex in M is 2, so it is a 2-complex. Because M is constructed exclusively of connected triangles (*i.e.* all edges are adjacent to exactly two triangles) it is also pure. So, M is a pure simplicial 2-complex.

3.2.4 Operations on mesh topology

Knowing that the mesh M is also a simplicial complex, we can now apply the same operations described in Section 3.2.2 to M. These topological operations will form the basis of our shape analysis of Chapter 4. Here we will discuss the connection between these operations and our approach.

Let M be a manifold triangle mesh as in Section 3.2.3, and thus, a pure simplicial complex. Because M is a 2-manifold surface, we know that every mesh vertex v is adjacent to D triangles and D edges where D is the degree of that vertex. Let τ be the simplex of v and $\mathcal{E}_\tau = \{\sigma_E : \tau \leq \sigma_E\}, \mathcal{F}_\tau = \{\sigma_F : \tau \leq \sigma_F\}$ be the respective edge and face simplices containing τ as a face. The star of τ is:

$$\mathrm{St}(\tau) = \{\tau\} \cup \mathcal{E}_\tau \cup \mathcal{F}_\tau \tag{3.4}$$

So $|\mathrm{St}(\tau)| = 2D + 1$. Let $\mathrm{vert}(\mathcal{F}_\tau)$ and $\mathrm{edge}(\mathcal{F}_\tau)$ be the vertex and edge simplices contained in $\mathcal{F}_\tau$. The link of τ is:

$$\mathrm{Lk}(\tau) = (\mathrm{vert}(\mathcal{F}_\tau) - \{\tau\}) \cup (\mathrm{edge}(\mathcal{F}_\tau) - \mathcal{E}_\tau) \tag{3.5}$$

and $|\mathrm{Lk}(\tau)| = 2D$. This can be seen clearly in the example of Figure 3.3.

We now introduce a new notion, the path of a set of vertices, which will become useful in our shape analysis methods of Chapter 4 and in the definition of the discrete Reeb graph below. Let $\mathcal{V}$ be a simplicial 0-subcomplex of $M = \{V, E, F\}$, *i.e.* a subset of mesh vertices in V. Let $\mathrm{Lk}(\mathcal{V}, M)$ be the link of simplical subcomplex $\mathcal{V}$ on the simplicial complex M. The *path* of $\mathcal{V}$ on M is a partition of the vertices of M into discrete level-sets some integer distance from $\mathcal{V}$. Each element of the partition is constructed in the following recursive manner.

$$\mathcal{P}_0 = \mathcal{V}$$

$$\mathcal{P}_1 = \mathrm{Lk}(\mathcal{V}, M)$$

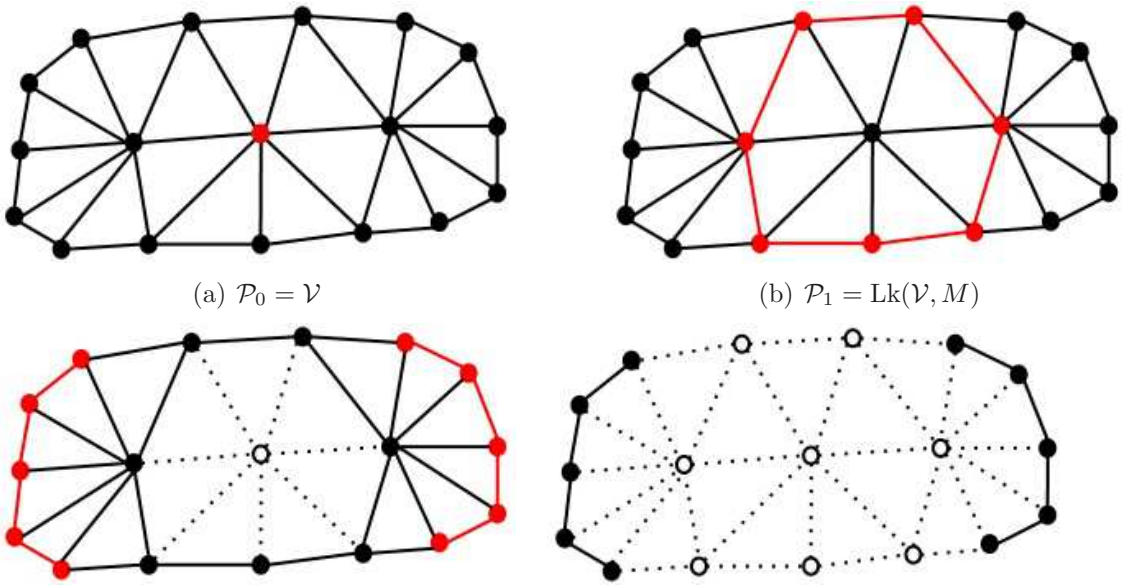
$$\mathcal{P}_2 = \mathrm{Lk}(\mathcal{P}_1, M - \mathrm{St}(\mathcal{V}))$$

$$\mathcal{P}_3 = \mathrm{Lk}(\mathcal{P}_2, M - \mathrm{St}(\mathcal{P}_1 \cup \mathcal{V}))$$

$$...$$

For some $\mathcal{P}_i$:

$$\mathcal{P}_i = \mathrm{Lk}\left(\mathcal{P}_{i-1}, M - \mathrm{St}(\bigcup_{j=0}^{i-2} \mathcal{P}_j)\right) \tag{3.6}$$



(a) $\mathcal{P}_0 = \mathcal{V}$ (b) $\mathcal{P}_1 = \mathrm{Lk}(\mathcal{V}, M)$

(c) $\mathcal{P}_2 = \mathrm{Lk}(\mathcal{P}_1, M - \mathrm{St}(\mathcal{V}))$ (d) $\mathcal{P}_3 = \mathrm{Lk}(\mathcal{P}_2, M - \mathrm{St}(\mathcal{P}_1 \cup \mathcal{V}))$

Figure 3.4: Example of Path($\mathcal{V}$) for 2D mesh. Given some simplicial 0-subcomplex $\mathcal{V}$ of M, each step finds the link of the previous step, but it ignores those parts of M previously seen. Thus it expands outward in a breadth-first manner.

This partition, $\mathrm{Path}(\mathcal{V}) = \{\mathcal{P}_0, \mathcal{P}_1, ..., \mathcal{P}_i\}$, is a simplicial 1-subcomplex of M, as is each of its elements. We will refer to its size, $|\mathrm{Path}(\mathcal{V})| = N$, as the length N of the path.

3.2.5 Discrete Reeb Graph

A Reeb graph [63] is a topological structure that describes the connectivity of the level sets of a scalar function. It is part of a larger area of mathematics called Morse theory [39] that uses differentiable functions on manifolds to understand their topology. Morse theory states that for a real-valued smooth function $f : X \to \mathbb{R}$ on a differentiable manifold X, the points where the differential of f vanishes, *critical*

points, correspond to relevant topological changes. The Reeb graph tracks these changes in graph form.

We would like to adapt these concepts from Morse theory to our discrete case, so that we can build a graph that captures critical points similarly to a Reeb graph. We know that our mesh M is a 2-manifold. Using Path($\mathcal{V}$), we can describe a function $f : (M, \mathcal{V}) \rightarrow \mathbb{N}$. Given some 0-simplex $\tau \leq M$, define f as follows:

$$f(\tau) = i \text{ if } \tau \leq \mathcal{P}_i$$

$$f^{-1}(i) = \mathcal{P}_i$$

Discrete Reeb graph: Let $f : (M, \mathcal{V}) \rightarrow \mathbb{N}$ be the scalar function defined on the simplicial 2-complex M that we just introduced based on the path. Let τ_1 and τ_2 be two 0-simplices of M. The discrete Reeb graph of f is defined by the equivalence relation $\tau_1 \approx \tau_2$, if and only if:

i. $f(\tau_1) = f(\tau_2)$, *i.e.* 0-simplices τ_1 and τ_2 lie in the same component of the partition induced by Path($\mathcal{V}$), and

ii. τ_1 and τ_2 are subcomplexes of the same connected component of $f^{-1}(f(\tau_1))$

Because our f is neither continuous nor differentiable, the discovery of critical points on which to base the construction of a traditional Reeb graph is impossible. This is the point where our discrete graph departs from existing Morse theory. Instead of constructing the graph from the critical points, we discover the critical points after constructing the graph. When partitioning the mesh M into a path based on the

subcomplex $\mathcal{V}$, there is sufficient information available to decide the connectivity of the level sets in Path($\mathcal{V}$).

Let $\mathcal{P}_i$ be a level set in Path($\mathcal{V}$). Let $\mathcal{P}_i = \{\mathcal{K}^i_0, \mathcal{K}^i_1, \ldots, \mathcal{K}^i_j, \ldots, \mathcal{K}^i_{M-1}\}$ be the connected components in $\mathcal{P}_i$. We know that there must exist some edge between the connected components of $\mathcal{P}_i$ and its neighbors $\mathcal{P}_{i\pm1}$. An edge exists between $\mathcal{K}^i_j$ and $\mathcal{K}^{i\pm1}_l$ if $\mathrm{St}(\mathcal{K}^i_j) \cap \mathrm{St}(\mathcal{K}^{i\pm1}_l)$ is nonempty. By connecting together the connected components of each element in the path, we are able to construct a discrete Reeb graph. That is, the vertices of the discrete Reeb graph are the connected components of $\mathcal{P}_i$, and the edges of the graph exist when those connected components' stars intersect.

The critical points of that graph then are those connected components $\mathcal{K}$ which represent end-points (have degree 1 in the discrete Reeb graph) and branches (have degree 3 or more). In Chapter 4, we refer to them as *caps* and *junctions*, respectively. They will form the basis of our alignment technique of Chapter 7.

3.3 Least-squares Fitting of Two 3D Point Sets

In this section we describe the fundamental results of Arun *et al.* [7] that play a critical role later in Chapter 7. Given two point sets $\{p_i\}$ and $\{p'_i\}$ of size N, they solve directly for the rotation R and translation T that minimizes the equation

$$\Sigma^2 = \sum_{i=1}^{N} ||p'_i - (Rp_i + T)||^2$$

using singular value decomposition (SVD).

Their algorithm is as follows:

Step 1: From $\{p_i\}$, $\{p'_i\}$, calculate their centroids p, p' and $q_i = p_i - p$, $q'_i = q_i - q$

Step 2: Calculate the 3×3 matrix

$$H = \sum_{i=1}^{N} q_i q_i'^t$$

where the superscript t denotes matrix transposition.

Step 3: Find the singular value decomposition (SVD) of H,

$$H = U\lambda V^t$$

Step 4: Calculate

$$X = VU^t$$

Step 5: Calculate $\det(X)$, the determinant of X. If $\det(X) = +1$, then $R = X$. if $\det(X) = -1$, the algorithm fails.

There are three general cases for possible point sets, two of which are degenerate.

(i) $\{p_i\}$ *are not coplanar* - There is a unique solution for R with no reflections. This is the non-degenerate case.

(ii) $\{p_i\}$ *are coplanar but not collinear* - There is a unique solution for R as well as a unique reflection. This is a degenerate case, but its answers are useful.

(iii) $\{p_i\}$ *are collinear* - There are infinitely many rotations and reflections. This is a problematic degenerate case, as the answer returned by this algorithm is unpredictable.

The degenerate case (iii) will play an important role later on in analyzing decreased alignment performance results.

This method provides a closed form solution to the least-squares minimizing transform that is integral to many different methods of alignment. We will be using it extensively later on in our alignment algorithms.

3.4 Generalized Iterative Closest Point

In Chapter 7, we will be using the Generalized Iterative Closest Point (Generalized ICP) software of Segal *et al*. [66] as the standard for the state of the art in surface registration. We will be comparing our method to Generalized ICP to determine the validity of our hypotheses. As mentioned in Chapter 2, ICP is an algorithm for the alignment of surfaces in three stages. Given some initial position of input point sets P and Q:

1. Find point correspondences between the sets using a (potentially modified) nearest neighbor method.

2. Compute the transform for P that minimizes the distance between Q and the transformed P in a least-squares sense.

3. Apply the transform to P and return to step 1 until some convergence condition is met.

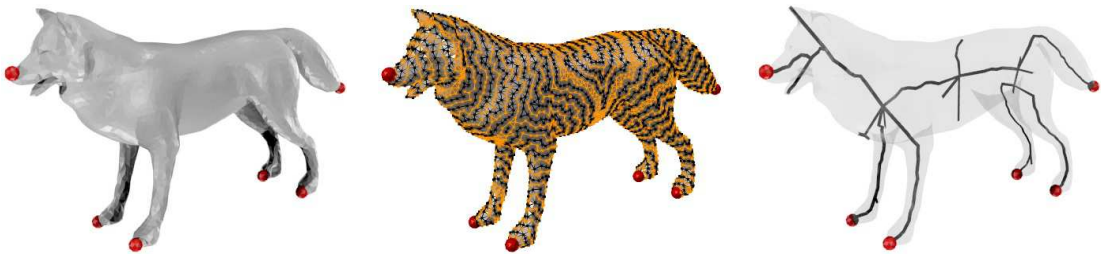
The Generalized ICP combines the Iterative Closest Point [11] and point-to-plane ICP [25] algorithms into a single probabilistic framework. They maintain speed and simplicity by computing the correspondences of step 1 using highly efficient nearest-neighbor approaches based on Euclidean distance. They then use principle component analysis (PCA) to calculate the normals at each point, and they integrate this into step 2 via a probabilistic version of the point-to-plane ICP of Chen and Medioni [25]. This allows them to incorporate structural information from both input surfaces to decrease the influence of incorrect correspondences.

CHAPTER 4

THE BREADTH-FIRST GRAPH: $\hat{G}$

We wish to formulate a method that simultaneously segments the mesh into parts and generates a curve-skeleton suitable for alignment. In this chapter, we develop a means to do so through the use of a modified breadth-first traversal of the mesh and the use of that traversal to build a simplified graph representation of the mesh, the *breadth-first graph*. The mesh itself is essentially treated as a graph, and we find that our breadth-first graph, much like the connectivity shapes of Isenburg, encodes the part structure of the mesh (Figure 2.5(b)). This approach has two main advantages. First, its implementation is straightforward. Second, because it largely ignores surface geometry, it is robust to surface variation.

To best explain, we will split our discussion into the following sections. In Section 4.1, we will describe the breadth-first graph algorithm and its user-defined input (*seed vertices*). Section 4.2 will describe how to segment a mesh using the breadth-first graph. In Section 4.3, we will describe how to extract a skeleton using the breadth-first graph. Finally, we then provide our final automatic breadth-first graph generation method in Section 4.4, in which seed vertices are generated automatically.



(a) (b) (c)

Figure 4.1: The breadth-first graph generated from manually selected seed vertices. (a) Given a wolf mesh, we select 6 seed vertices manually at the nose, tail and four paws. (b) We perform a breadth-first traversal of the mesh encoding each front (seen here as alternating black and orange edge loops) as a node in a new graph. (c) The graph structure can be seen by embedding each of these fronts in $\mathbb{R}^3$ as the centroid of the vertices it encodes.

4.1 Breadth-first Graph Algorithm

Starting with a mesh $M = \{V, E, F\}$, we use a breadth-first traversal of M to produce the associated breadth-first graph $\hat{G} = \{\hat{V}, \hat{E}\}$. Each vertex $\hat{v} \in \hat{V}$ of the graph encodes a subgraph of M associated with a single step in the breadth-first traversal, or *front*. A vertex of the breadth-first graph is called a *front vertex* to distinguish it from a mesh vertex. Let $\text{vert}(\hat{v})$ and $\text{edge}(\hat{v})$ be the mesh vertices and mesh edges of M associated with front vertex $\hat{v}$. The edges $\hat{E}$ of the breadth-first graph encode the propagation relationship between fronts. They are called *front edges* to distinguish from mesh edges. A front edge exists between two front vertices $\hat{v}$ and $\hat{w}$ if there is a mesh edge between an element of $\text{vert}(\hat{v})$ and an element of $\text{vert}(\hat{w})$.

The primary distinction between a standard breadth-first traversal, which begins from a single vertex on the mesh, and the breadth-first traversal that produces our graph is that we allow traversal to begin from multiple mesh vertices. We call

these starting vertices *seed vertices*. These seed vertices, $V_S \subseteq V$, can be distributed arbitrarily across the mesh and are required input to the algorithm.

We now present an algorithm to compute the breadth-first graph $\hat{G}$ of a mesh M, given a set of seed vertices V_S (Algorithm 1). In this algorithm, V_{new} represents the mesh vertices under consideration in the current iteration. V_{seen} are those vertices already seen in the breadth-first traversal. $\hat{V}_{prev}$ and $\hat{V}_{new}$ are the set of front vertices created in the previous iteration and the set of front vertices created in the current iteration, respectively. $ring(v)$ is the set of vertices in the link (Section 3.2.2) of mesh vertex v. Given a graph $G = \{V, E\}$ and a set of vertices, W, the subgraph induced by W is $\{W, E_W\}$ where $E_W = \{(v_i, v_j) : v_i, v_j \in W, (v_i, v_j) \in E\}$, *i.e.* those edges whose endpoints both lie in W.

4.2 Segmentation from $\hat{G}$

One of the important characteristics of the breadth-first graph $\hat{G} = \{\hat{V}, \hat{E}\}$ is that the front vertices $\hat{V}$ define a disjoint cover of V. That is, $\{\text{vert}(\hat{v})\}_{\hat{v} \in \hat{V}}$ is a partition of V. While this can be shown to be true in a formal sense, it is far too fine a partition to be of much use. To produce a more useful partition, some method for combining the vertices in $\hat{V}$ into meaningful clusters is necessary.

We first look at the characteristic topology of $\hat{G}$. The front vertices $\hat{V}$ fall into three different types:

- *Cap vertices*: front vertices of degree 1
- *Pipe vertices*: front vertices of degree 2

Algorithm 1 BreadthFirstGraph(M, V_S)

Require: Mesh M, seed vertices V_S

$\hat{V} = \emptyset$; $\hat{E} = \emptyset$
$V_{new} = V_S$; $V_{seen} = \emptyset$; $\hat{V}_{prev} = \emptyset$
while $V_{new} \neq \emptyset$
// create new fronts
Γ = subgraph of M induced by V_{new}
$\hat{V}_{new}$ = front vertices induced by the connected components of Γ, one per component
$\hat{V} = \hat{V} \cup \hat{V}_{new}$
// define front edges
for all $\hat{v} \in \hat{V}_{prev}$
for all $\hat{v}_{new} \in \hat{V}_{new}$
// if fronts share a mesh edge, connect them
if $\exists (v_i, v_j) \in E, v_i \in \text{vert}(\hat{v}), v_j \in \text{vert}(\hat{v}_{new})$ **then**
$\hat{E} = \hat{E} \cup \{(\hat{v}, \hat{v}_{new})\}$

$V_{seen} = V_{seen} \cup V_{new}$
$V_{new} = (\bigcup_{v \in V_{new}} ring(v)) - V_{seen}$
$\hat{V}_{prev} = \hat{V}_{new}$

$\hat{G} = \{\hat{V}, \hat{E}\}$

- *Junction vertices*: front vertices of degree > 2

We can use this topology to make semantic inferences about the mesh vertices each front vertex encodes. That is, the type of front vertex to which a mesh vertex belongs (cap, pipe or junction) tells us something about where it belongs in the overall part structure. Cap vertices correspond to the "tips" of parts, pipe vertices correspond to the length or body of a particular part, and junction vertices define part distinctions. Using this simple interpretation, we can segment the mesh into parts by using junctions as part boundaries. That is, mesh M can be segmented using a segmentation of $\hat{V}$ such that:

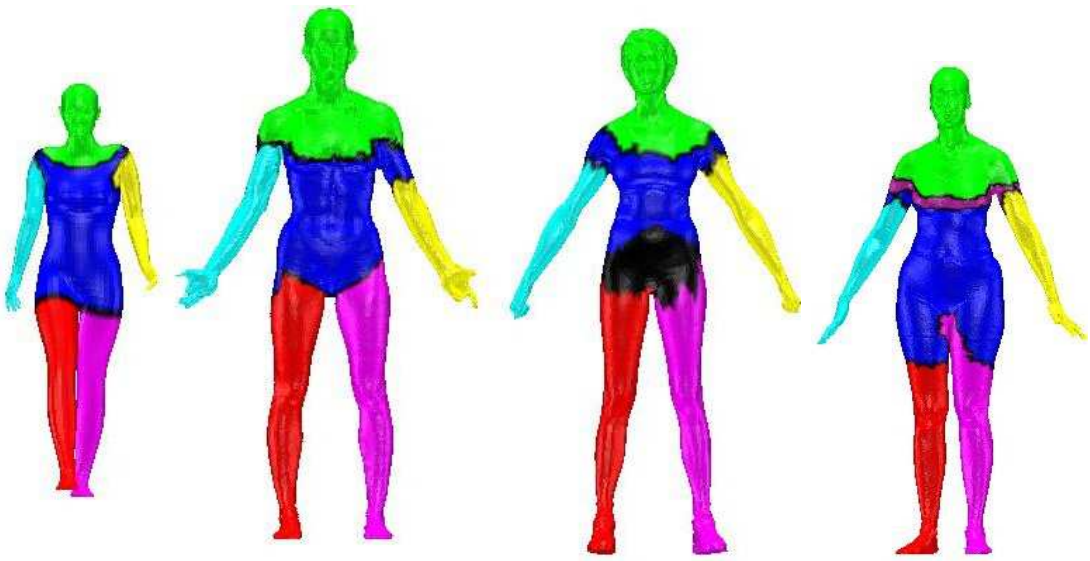


Figure 4.2: The BFG segmentation. By interpreting junction vertices (front vertices in $\hat{G}$ with > 2 neighbors) as part boundaries, the breadth-first graph segments the mesh into parts. It does so consistently for different meshes of the same object class. Colored faces belong to cap or pipe segments, and black faces belong to junction segments.

1. Any two adjacent non-junction (pipe or cap) vertices are part of the same segment.

2. All junction vertices belong to a unique segment.

This can be accomplished by progressively combining adjacent non-junction vertices and their associated mesh vertices until all non-junctions are adjacent to junction vertices (or the graph collapses to a single vertex in the case of a graph with no junctions). Then, each vertex in the graph is assigned a unique segment. Figure 4.2 demonstrates how the breadth-first graph segments the mesh into parts consistently for four different meshes from the same object class.

Applying the same terminology to segments that we used for front vertices, we can define the individual segments in this way:

- *Cap segments*: Segments that contain at least one cap vertex
- *Pipe segments*: Segments that contain no cap vertices and are connected to 2 or fewer junctions (0 in the case of pure ring topologies, 1 for rings connected to another segment and 2 for the most-common straight-line segment)
- *Junction segments*: Segments that are connected to more than 2 other segments

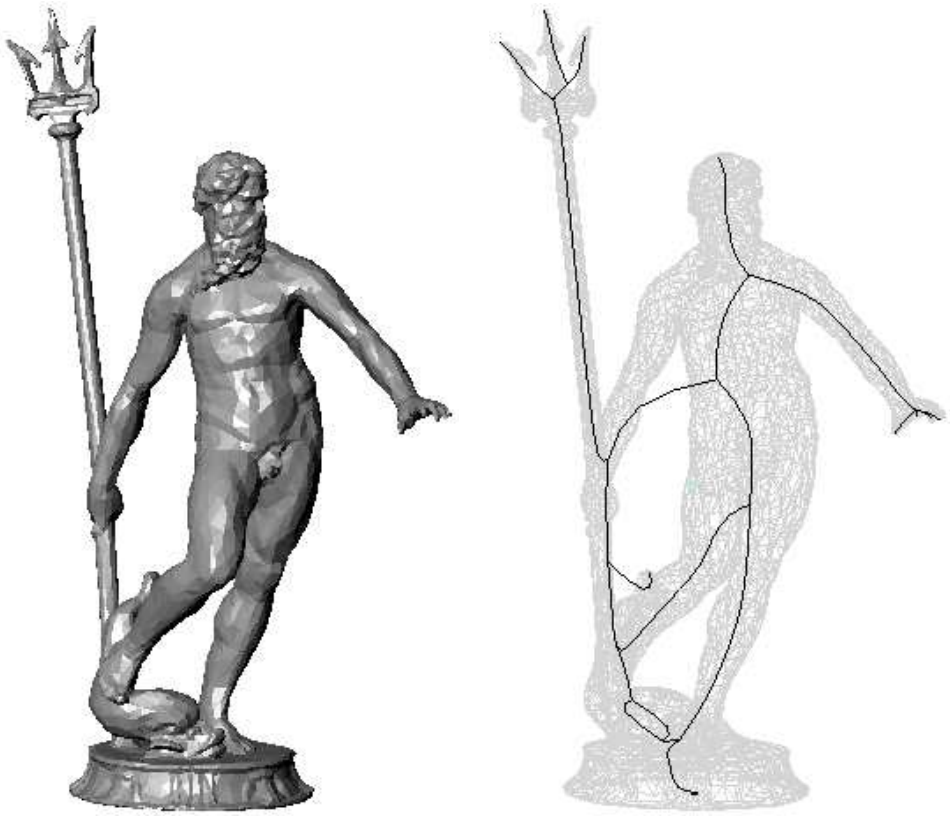
We call this segmentation the BFG segmentation.

4.3 Curve-skeleton from $\hat{G}$

$\hat{G}$ can also be used to create a skeleton of the mesh in a straightforward manner. We interpret each front vertex $\hat{v}$ as the centroid of its corresponding mesh vertices $\text{vert}(\hat{v})$, and connect these centroids via front edges accordingly. This skeleton is called the BFG skeleton. Figure 4.3 demonstrates this process on the high-genus surface of Neptune.

We can use this skeleton to measure the relative distances of parts by measuring their distance along the graph $\hat{G} = \{\hat{V}, \hat{E}\}$. Let all $\hat{v} \in \hat{V}$ be represented as the centroid $\mathbf{v}$ of their mesh vertices $\text{vert}(\hat{v})$. Let $\mathbf{v}_i$ and $\mathbf{v}_j$ be the centroids of front vertices $\hat{v}_i$ and $\hat{v}_j$. $\text{graphDist}(\hat{v}_i, \hat{v}_j)$ is the shortest path from $\mathbf{v}_i$ to $\mathbf{v}_j$ along the graph, as calculated via Dijkstra's algorithm.

Much like our surprise at the expressiveness of the connectivity shapes of Isenburg *et al*. [42], we were surprised to find just how much part information is encoded purely in the connectivity of a mesh. We want to use this information to find the shared structure between mesh shapes. The primary roadblock in using the



(a) (b)

Figure 4.3: The BFG skeleton. We can generate a curve skeleton from the breadth-first graph by interpreting each front vertex in graph as the centroid of its corresponding mesh vertices. This curve skeleton captures significant topological features even in high-genus surfaces.

breadth-first graph in an automated manner is the necessity of user input in the form of the seed vertices V_S. Another concern is that of over-segmentation. Since our ultimate end is finding the correspondence of parts between similar shapes, in order to save time and computation, we would rather work with a coarse segmentation than a fine-grained one. The next section addresses these two issues.

4.4 Automatic $\hat{G}$ Generation

We wish to automatically generate a breadth-first graph for a mesh that segments it into a coarse yet sufficiently expressive part structure. This requires an intelligent choice of seed vertices. In general, we have found that using mesh vertices at the tips of appendages as seeds leads to balanced part structures. So, a logical choice is to use these mesh vertices at the tips of appendages as our input seed vertices, V_S. This leads to a bit of a chicken/egg problem: we want to use the breadth-first graph to understand the semantic structure of a mesh, but we need that structure (e.g. the tips of appendages) to produce a usable graph.

4.4.1 Priming

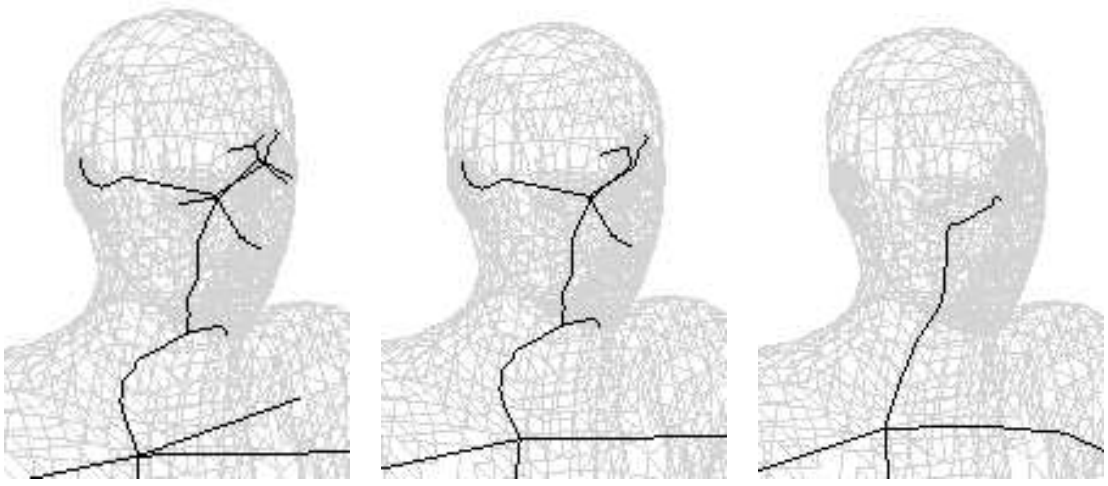
The original impetus for the breadth-first graph came while observing the propagation of one-rings during a breadth-first traversal across a mesh. We observed that, regardless of the vertex chosen as the starting point for propagation, the rings would terminate (i.e. run out of unseen vertices) at or very near the tips of appendages. Recalling our earlier discussion of front vertex types, we described cap vertices as those that encode mesh vertices at appendage tips. Therefore, we should be able to use the cap vertices from a priming run as input to a second, final run, thus extracting usable semantic information from our mesh for the price of two traversals of its vertices.

4.4.2 Hairs

In practice, however, not all cap vertices are actual appendage tips. Indeed, in most cases, the algorithm produces some cap vertices that are essentially artifacts of the traversal process and have no semantic significance whatsoever. We refer to these artifacts as "hairs" (Figure 4.4). What's more, even if we had no artifacts in our graph and all of our caps were actual appendage tips, the resulting part-structure is often too finely grained to use for our ultimate goal of alignment.

4.4.3 Distilling the Graph

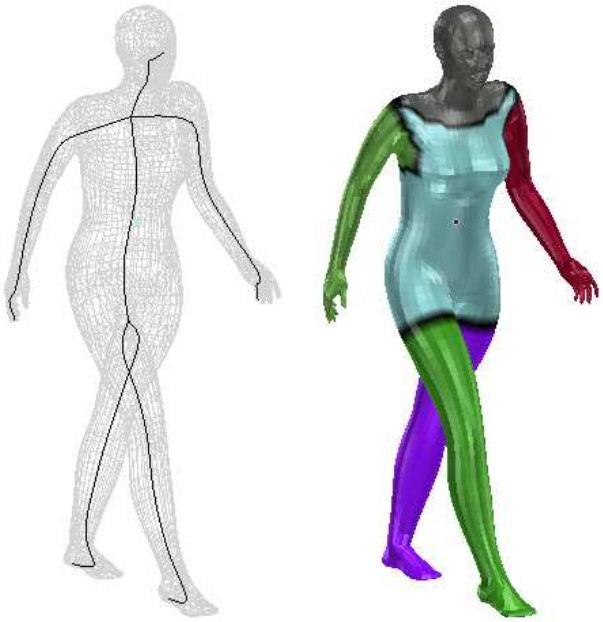
We solve both problems by iteratively *trimming* individual segments from the graph in a process we refer to as *distilling* the graph. We define a *segment* $\hat{S} \subset \hat{V}$ to be any maximal connected set of pipe and cap vertices. Note that the union of all segments is a partition of all cap and pipe vertices. We define the length of a segment $|\hat{S}|$ to be the number of front vertices in that set. Nontrivial graphs (ones with at least one junction) are composed of two types of segments: segments containing only pipe vertices, which we call *pipe segments*, and segments composed of pipe vertices ending in a cap vertex, which we call *cap segments*. We trim only cap segments. Trimming a segment involves removing its front vertices from the graph, associating those fronts' mesh vertices with the junction vertex to which the segment is connected, and decreasing the junction's degree by one. It should be understood that trimming multiple segments from the graph can and usually does cause junction vertices to be reclassified as pipe (or even cap) vertices, thus reducing the total number of segments



(a) Original skeleton (too many "hairs")

(b) After trimming segments length 1

(c) After iterative trimming



(d)

(e)

Figure 4.4: Distilling the breadth-first graph. When first generated, (a) the breadth-first graph $\hat{G}$ contains a number of cap segments that are artifacts of the traversal process and not semantically meaningful. (b) We first trim all segments of length one. We then trim the graph iteratively (c) to distill it to a minimal state (d) with segments of near-uniform number of edges. (e) $\hat{G}$ encodes both the curve-skeleton and the part structure.

and distilling the graph to a simpler structure. Figure 4.4 demonstrates the graph distillation process.

Algorithm 2 defines graph distillation. We first trim all cap segments $\hat{S}_i \subset \hat{V}$ where $|\hat{S}| = 1$. Through experimentation, we found that this initial trimming removes most if not all "hairs", or noise-related segments, and produces a much better starting point for the following iterative process. Given the list of cap segment sizes, we calculate its standard deviation σ and trim all cap segments $\hat{S}_i$ where $|\hat{S}_i| < \sigma$. This is repeated, each time producing a simpler graph with fewer segments, until no trimming occurs. The goal is to produce a graph with minimal variance in segment length. In practice, this usually takes between one and three iterations. Table 4.1 shows the average number of iterations for distillation convergence for each object class in the Chen database.

Algorithm 2 Distill($\hat{G}$)

Require: Breadth-first graph $\hat{G}$ of mesh M
  $\hat{S}_{caps}$ = set of cap segments of $\hat{G}$
  Trim all $\hat{S}_i \in \hat{S}_{caps}$ where $|\hat{S}_i| = 1$
  repeat
    $\hat{S}_{caps}$ = set of cap segments of $\hat{G}$
    $X = \{|\hat{S}_i| : \hat{S}_i \in \hat{S}_{caps}\}$
    σ = standard deviation of X
    Trim all $\hat{S}_i \in \hat{S}_{caps}$ where $|\hat{S}_i| < \sigma$
    N_{trim} = number of trimmed segments
  until $N_{trim} = 0$
  return Modified $\hat{G}$

Class	Avg Iterations
Airplane	1.1
Mech	1.15
Glasses	1.4
Bearing	1.5
Pliers	1.6
Bust	1.65
Cup	1.7
Fish	1.8
Table	1.85
Bird	1.9
Teddy	1.9
Octopus	2.0
Vase	2.1
Ant	2.25
Fourleg	2.3
Chair	2.5
Human	2.55
Hand	2.6
Armadillo	2.9

Table 4.1: Average number of iterations for distillation per class. After trimming all cap segments with one edge, the graph distillation algorithm (Algorithm 2) iteratively trims segments. This table shows the average number of iterations, per class, for that process.

4.4.4 Automatic $\hat{G}$ algorithm

We are now ready to present our algorithm for the automatic generation of a breadth-first graph for a mesh M (Algorithm 3). We first generate an initial graph $\hat{G}_0$ using the mesh vertex closest to the centroid of all mesh vertices (v_{cent}) as the seed. This is the only time the geometry of the mesh is used in the process, and its choice is based on the assumption that a seed vertex from a central location is most likely to find caps at extremities. After distilling $\hat{G}_0$, we use the mesh vertices corresponding

to its cap vertices as input to a second run to generate $\hat{G}_1$. The distilled version of this graph is the final breadth-first graph.

Algorithm 3 AutoBFG(M)

Require: Mesh $M = \{V, E, F\}$
  // Primer run, to get appendage tips
  v_{cent} = mesh vertex closest to centroid of V
  $\hat{G}_0$ = BreadthFirstGraph($M, \{v_{cent}\}$)
  $\hat{V}_{caps}$ = cap vertices of Distill($\hat{G}_0$)
  $V_{tips} = (\bigcup_{\hat{v} \in \hat{V}_{caps}} \text{vert}(\hat{v})) - \{v_{cent}\}$ // remove v_{cent}

  // Using tips from primer run, build final graph
  $\hat{G}_1$ = BreadthFirstGraph(M, V_{tips})
  return Distill($\hat{G}_1$)

After the final breadth-first graph is generated, we produce a skeleton from this graph (Section 4.3). We lock the skeleton's junction and cap vertices and perform Laplacian smoothing on the pipe vertices [32]. Figure 4.4(d) demonstrates the mesh skeleton for a woman mesh.

One of our goals for the AutoBFG algorithm was for it to be invariant to rigid transform, *i.e.* AutoBFG(M) = AutoBFG($\alpha(M)$) for some rigid transform α. This required the use of the geometry of M only as part of the primer run when finding v_{cent}, the mesh vertex nearest the centroid of all mesh vertices. Since the proportionality of Euclidean distance is preserved under rigid transformation, the centroid of mesh vertices would also be preserved. The v_{cent} found for M will be the same v_{cent} found for $\alpha(M)$. Thus, their breadth-first graphs will be the same.

If we were to relax the requirement of invariance to rigid transform, we could remove all reliance on geometry in generating the breadth-first graph. Algorithm 4 defines the automatic generation of a breadth-first graph for a mesh M using no

geometric information at all. Using the name TopoBFG to distinguish it as a purely topological version of AutoBFG, the initial priming seed vertex is chosen randomly, but the rest of the algorithm is equivalent to AutoBFG.

Algorithm 4 TopoBFG(M)

Require: Mesh $M = \{V, E, F\}$
  // "Primer" run, to get appendage tips
  v_{rand} = mesh vertex randomly chosen from V
  $\hat{G}_0$ = BreadthFirstGraph($M, \{v_{rand}\}$)
  $\hat{V}_{caps}$ = cap vertices of Distill($\hat{G}_0$)
  $V_{tips} = (\bigcup_{\hat{v} \in \hat{V}_{caps}} \text{vert}(\hat{v})) - \{v_{rand}\}$ // remove v_{rand}

  // Using tips from primer run, build final graph
  $\hat{G}_1$ = BreadthFirstGraph(M, V_{tips})
  return Distill($\hat{G}_1$)

4.5 Conclusions

We have shown in this chapter that the topology of a mesh contains a significant amount of part information. To extract that part information, we have provided a novel approach based on a modified breadth-first traversal of mesh vertices. We have demonstrated how this traversal can be encoded into a graph structure and how that graph can be used for shape analysis tasks such as segmentation and skeleton extraction. The difficulty with using the breadth-first graph for shape analysis is its dependence on user input in the form of seed vertices. We have shown that through the use of an initial priming traversal of the mesh, we can use the result of that priming run to construct a final graph in an automated manner.

CHAPTER 5

EXPERIMENTAL DATA

In this chapter, we discuss the shape data set we used for our shape analysis and alignment experiments and testing. Given the problem definition from Section 2.1, the data set will need to meet certain requirements, which are given in Section 5.1. In Section 5.2 we discuss the shape database we chose that meets those requirements.

5.1 Requirements

In making our selection of a 3D model data set, we consider 5 main criteria. The models in our data set should:

1. **Be watertight, singly-connected, 2-manifold triangle meshes**: The basis in existing combinatoric topology of our approach to alignment (discussed in Chapter 3) presupposes that our meshes are 2-manifold and triangular. The watertightness requirement prevents potential boundary conditions and simplifies our overall approach. We want our models to be singly connected so that the complications involved in dealing with compound objects (*e.g.* cars with disconnected wheels) can be avoided.

2. **Cover a broad set of object categories**: Since our goal is the semantically meaningful alignment of shapes within the same object class (*e.g.* humanoid,

animals, furniture), our data should have a number of object classes to test. Also, there should be a substantial number of meshes within each class to prevent bias towards particular shapes.

3. **Exhibit shape variation within each class**: The shapes of a class should differ in non-trivial ways (*e.g.*, significant surface variation, part articulation, differences in surface sample density). This will allow us to get an impression of the robustness of our methods to shape variation within each class.

4. **Be neither too simple, nor too complex to decompose into parts**: Our alignment hypothesis is that knowledge of part decomposition improves the alignment of different shapes of the same object class. To prove or disprove this, we need shapes that have a semantically meaningful part decomposition.

5.2 Data Set

To meet the requirements of Section 5.1, there are a number of candidate databases. One obvious possibility is the Princeton Shape Benchmark [70]. It contains nearly two thousand 3D object models and is organized by class with many different models in each class. Unfortunately, many of the models contain mesh degeneracies, requiring significant repair before use. Also, many of the models are compound objects, composed of multiple connected components. Finally, the database is skewed towards rigid, man-made objects, although there are some natural objects such as animals, plants and humans. Only a few, however, demonstrate much part articulation.

Figure 5.1: Princeton Shape Benchmark [70]. This shape database is widely used as a benchmark for shape recognition and retrieval. For our purposes, it was not ideal due to its shapes being non-manifold and multiply connected.

Instead, we chose the mesh database of Chen *et al*. [24] that was designed specifically for the purpose of constructing a benchmark for 3D mesh segmentation. Their database is based on the the Watertight Track of the 2007 SHREC Shape-based Retrieval Contest 3D models of Giorgi *et al*. [36] that they processed into manifold surfaces. This set contains 400 models split into 20 object classes. It contains nine classes with significant part articulation and three classes where objects have non-zero genus. Every object is represented by a watertight triangle mesh with a single connected component. Of the 20 different classes in the 2007 SHREC database, Chen selected 19 classes (the unused class were spring models whose segmentation would have been trivial), each of which can be seen in Figure 5.2. These classes of the Chen

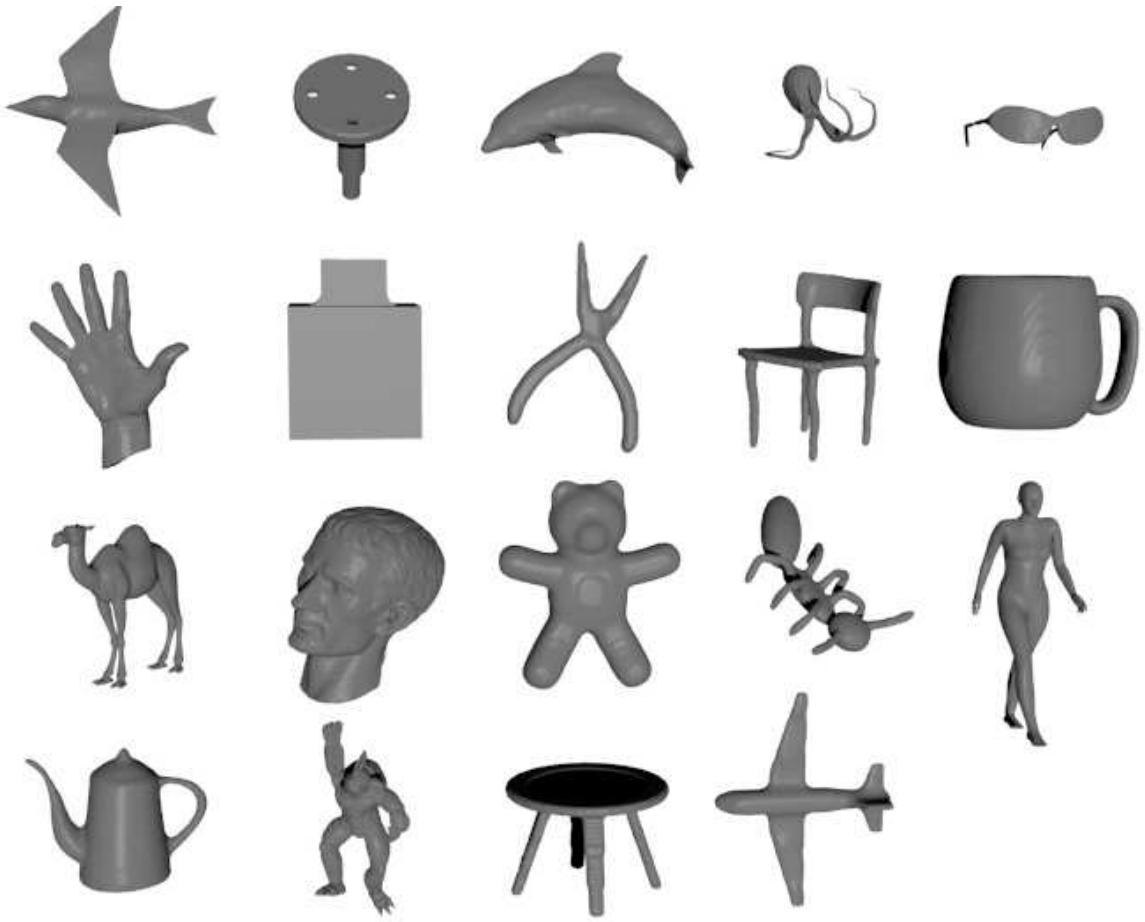


Figure 5.2: The object classes from the mesh database of Chen *et al*. [24].

database are: Human, Cup, Glasses, Airplane, Ant, Chair, Octopus, Table, Teddy, Hand, Plier, Fish, Bird, Armadillo, Bust, Mech, Bearing, Vase and Fourleg.

CHAPTER 6

APPLICATIONS OF THE BREADTH-FIRST GRAPH

In this chapter we discuss the results of using our breadth-first graph algorithm for two shape analysis applications, mesh segmentation and curve-skeleton extraction. We will show that even though we use mesh topology as the basis for our analysis, our results are commensurate with the more complex geometry-centered approaches. By establishing the quality of our results within the context of existing benchmarks, we demonstrate the power of our topological approach.

We first show our results for mesh segmentation using the breadth-first graph. We then show our results for skeleton extraction.

6.1 Segmentation

If we are to properly integrate part structure into shape alignment using our breadth-first algorithm, we need to know that the segmentations it produces are proper reflections of the mesh's part structure. In this section we will discuss our algorithm's performance against the benchmark for mesh segmentation of Chen *et al*. [24]. We first discuss the benchmark's structure and goals. We then describe the error metrics that provide the basis of evaluation. There were some issues with adapting our method to suit the input requirements of the benchmark that we then discuss. Finally, we give the performance results of our method.

Figure 6.1: AutoBFG segmentations. Breadth-first graph segmentations of object classes of Chen *et al*. [24] based on the AutoBFG algorithm. For classes with highly articulated part structures, the part structure is captured.

6.1.1 Benchmark of Chen *et al*.

In Chapter 5, we discussed the shape database of Chen *et al*. [24] as our experimental data set. They created that database for the sake of establishing a benchmark for mesh segmentation. That benchmark provides a set of evaluation metrics that can be used to evaluate how well computer-generated segmentations match human-generated ones. As a baseline, they compiled a set of human-generated segmentations. They compared the segmentation results of various existing algorithms against the

Figure 6.2: TopoBFG segmentations. Here the segmentations are based on the TopoBFG algorithm, which ignores mesh geometry entirely.

human baseline based on four different metrics. We add the results of our AutoBFG (Figure 6.1) and TopoBFG (Figure 6.2) algorithms to the list and compare them to the human benchmark.

6.1.2 Error metrics

Here we will discuss the four different error metrics that are used by Chen *et al*. [24] in determining the quality of segmentation. The first metric determines how

close segment boundaries are to the baseline boundaries, and the last three measure the consistency of segment interiors.

- **Cut Discrepancy**: Developed and used by Huang and Dom [41] for image segmentation, this metric sums the distances from points along the cuts in the computed segmentation to the closest cuts in the baseline. Its advantage is that it provides a straightforward measure of boundary alignment. Its disadvantage is that it is sensitive to the granularity of segmentation.

- **Hamming Distance**: Also used by Huang and Dom [41], this uses Hamming Distance to measure the total difference between the internal regions of two segmentations. It is essentially a metric of agreement between two segmentations based on shared area between individual segments. This gives good information when the baseline and automatic segmentations agree on the number of segments, but it will be noisy when they do not.

- **Rand Index**: First posed by Rand [62] for the evaluation of clustering methods, this measures the likelihood that a pair of faces are either in the same segment in two segmentations, or in different segments in both segmentations. The metric gives the proportion of face pairs that agree or disagree on segmentation identity. The main advantage is that it captures area overlaps without explicitly finding segment correspondences.

- **Consistency Error**: This metric, proposed by Martin *et al*. [54], is very similar to Hamming distance in that it measures the difference between segmentation regions. However, it does so while trying to account for the hierarchical structure

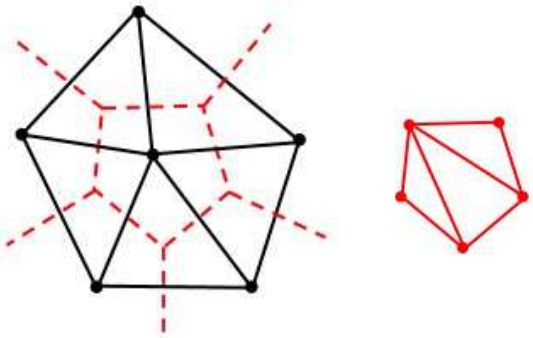
humans impose on objects. The disadvantage of this metric is that it provides better scores for segmentations with different numbers of segments. In this way, it is a good counter-balance to the *Hamming Distance* metric.

Chen *et al.* [24] tested seven well-known algorithms against the human benchmark under these error metrics. These algorithms are described in Appendix A. In addition to these seven algorithms, Chen *et al.* [24] evaluated the human-generated segmentations with respect to each other and used two other trivial mesh segmentations as a sanity check. The first trivial case is where all faces belong to the same segment, which they refer to as "None". The second trivial case is where each face belongs to its own segment, which they refer to as "Every".

6.1.3 Issues with testing the breadth-first graph

We wanted to test both our AutoBFG and TopoBFG algorithms' segmentation performance against the algorithms listed above, using the software provided by Chen (http://segeval.cs.princeton.edu). Unfortunately, this software evaluates only segmentations of mesh faces, and our algorithm segments mesh vertices. To use the software, we would have to modify our algorithm to produce a face segmentation.

In the end, we decided that such a modification was unnecessary. Since our algorithm requires only vertices and edges, instead of using the mesh M, we could use the dual mesh, M^*. If $\hat{G}_M$ is a segmentation of the vertices of M, $\hat{G}_{M^*}$ would be a segmentation of the faces of M, *i.e.* the vertices of M^*. The primary difficulty with using the dual mesh M^* is that it is not triangular. Our methods are designed



(a) Mesh and dual (b) Dual triangulation

Figure 6.3: (a) Given the mesh M (in black) and its dual M^* (in red), (b) we triangulate the non-triangular faces of M^*. This ensures that the dual will be a manifold triangle mesh, or simplicial 2-complex.

to operate on simplicial complexes (Section 3.2.3). For 2-manifolds, this can only be the case when they are composed of 2-simplices, *i.e.* they are triangular surface meshes. Our solution to this problem was to triangulate the non-triangular faces in M^* (Figure 6.3).

6.1.4 Results

Overall, their performance was better than expected, especially that of the purely topological version TopoBFG. Figures 6.4, 6.5 6.7, and 6.6 show the results of running both our AutoBFG and TopoBFG algorithms on the benchmark of Chen *et al.* [24].

Before analyzing the results, we introduce the issue of junction segments. The primary difference between the segmentations based on $\hat{G}$ and other methods is the existence of so-called *junction segments* (Section 4.2). These junction segments correspond to those mesh vertices encoded by junction vertices in $\hat{G}$. We kept these as

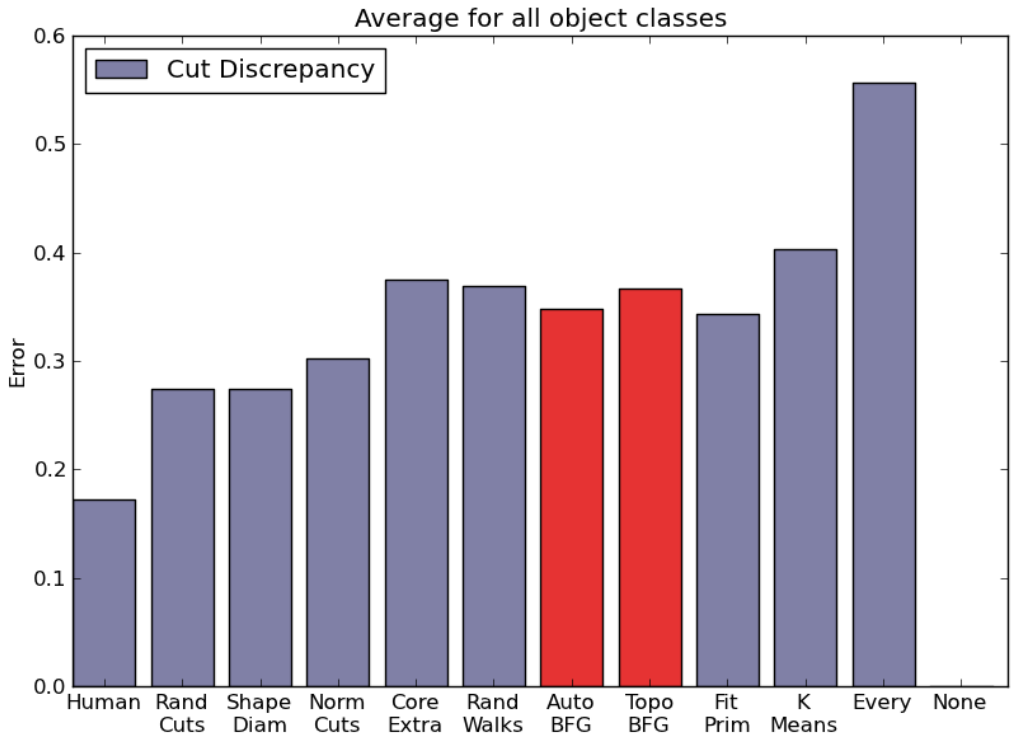


Figure 6.4: **Cut Discrepancy**: Comparison of segmentation algorithms on the shape database of Chen *et al*. [24]. Shown here are the Cut Discrepancy error values averaged over all object classes. This measures the overall disagreement between segment boundaries. The lower the value, the more the algorithm's boundaries agree with the baseline's.

separate segments rather than try to integrate their mesh vertices into surrounding pipe and cap segments for essentially two reasons. Primarily, we wanted to test our actual segmentation, since its dual relationship with the skeletonization of Section 4.3 will play an important role in our later alignment techniques. Second, because of the graph distillation process (Algorithm 2 of Section 4.4.3), junction segments can sometimes grow quite large, requiring potentially expensive extra computation to distribute their vertices. The downside is that these junction segments end up negatively influencing the error metrics of Chen *et al*. [24], as we will see.

Cut Discrepancy

Under this metric, both our AutoBFG and TopoBFG algorithms outperformed three different methods, Core Extraction, Random Walks and K-means clustering, and AutoBFG performed nearly as well as Fitting Primitives. We believe this is due to the fact that although Cut Discrepancy is sensitive to segmentation granularity, which is increased by junction segments, it is less sensitive when the extra segments are constrained to very long, thin segments near to existing baseline boundaries, which junction segments usually are. Note: the None method (where all faces belong to the same segment) has no boundaries, so no discrepancy error can be calculated.

Hamming Distance

Our algorithms only did better under this metric than one other method, K-Means, but they are still competitive. The issue here is the Hamming-Rf metric, or "false alarm" rate. This corresponds to the mismatch between the additional junction segments matching with much larger baseline segments.

Consistency Error

Under this metric, our algorithms performed better than two others, Fitting Primitives and K-Means. Because this metric accounts for nested, hierarchical differences in segmentations, junction segments likely played less of a role in decreasing performance. It is interesting to note that TopoBFG outperformed AutoBFG slightly,

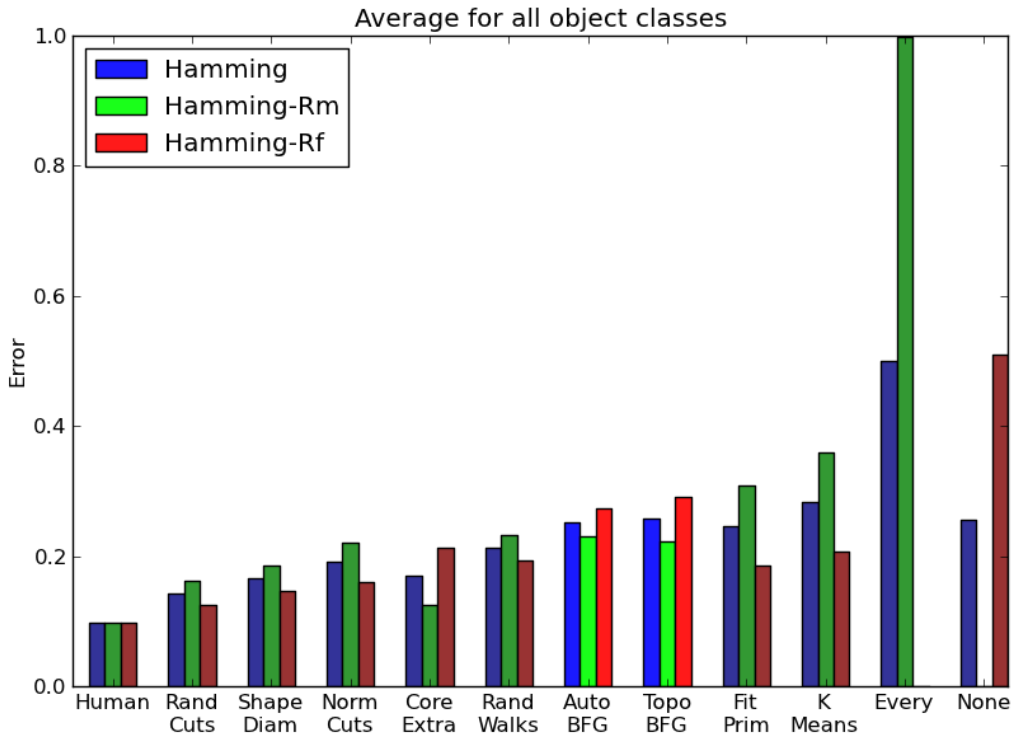


Figure 6.5: **Hamming Distance**: Comparison of segmentation algorithms on the shape database of Chen *et al*. [24]. Shown here are the Hamming distances averaged over all object classes. This measures the overall disagreement between segment regions. The lower the value, the more the algorithm's segment regions agree with the baseline. Hamming-Rm is the missing rate, Hamming-Rf is the false alarm rate, and Hamming is the average of the two.

though this is likely due to a few lucky choices of starting vertex leading to slightly better segmentations.

Rand Index

This is the metric under which our algorithms performed most poorly. The issue of extraneous junction segments is a significant factor, but it should be noted that this metric is particularly sensitive to the baseline segmentations. For example,

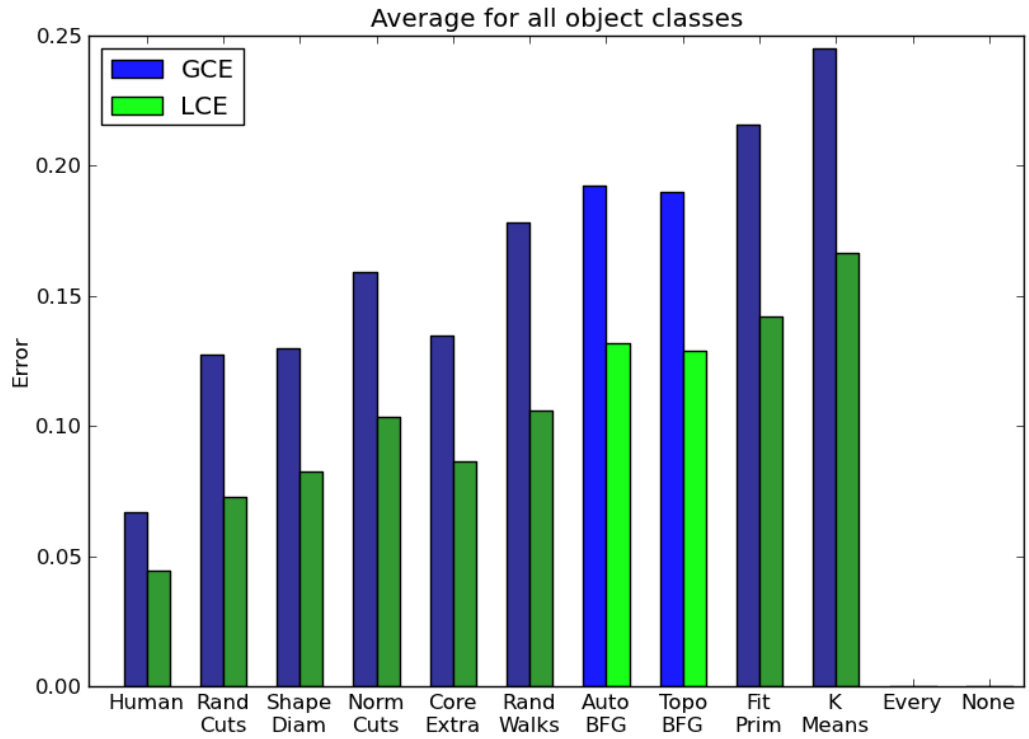


Figure 6.6: **Consistency Error**: Comparison of segmentation algorithms on the shape database of Chen *et al*. [24]. Shown here are the Consistency Error values averaged over all object classes. This measures the disagreement between segment regions in a way that does not penalize differences in hierarchies of part structure. It is a sum of per-face refinement error values that can be defined w.r.t. the baseline or the algorithm's segmentation. *Global Consistency Error* (GCE) forces all faces to be defined within a global error basis, and *Local Consistency Error* (LCE) is defined on a per-face basis.

in the case of the Cup object class where most human-made segmentations resulted in only a few segments, "None" outperformed nearly every algorithm. In the case of the "Human" object class where there was significant disagreement among human-man segmentations, "Every" performed nearly as well as the human benchmark.

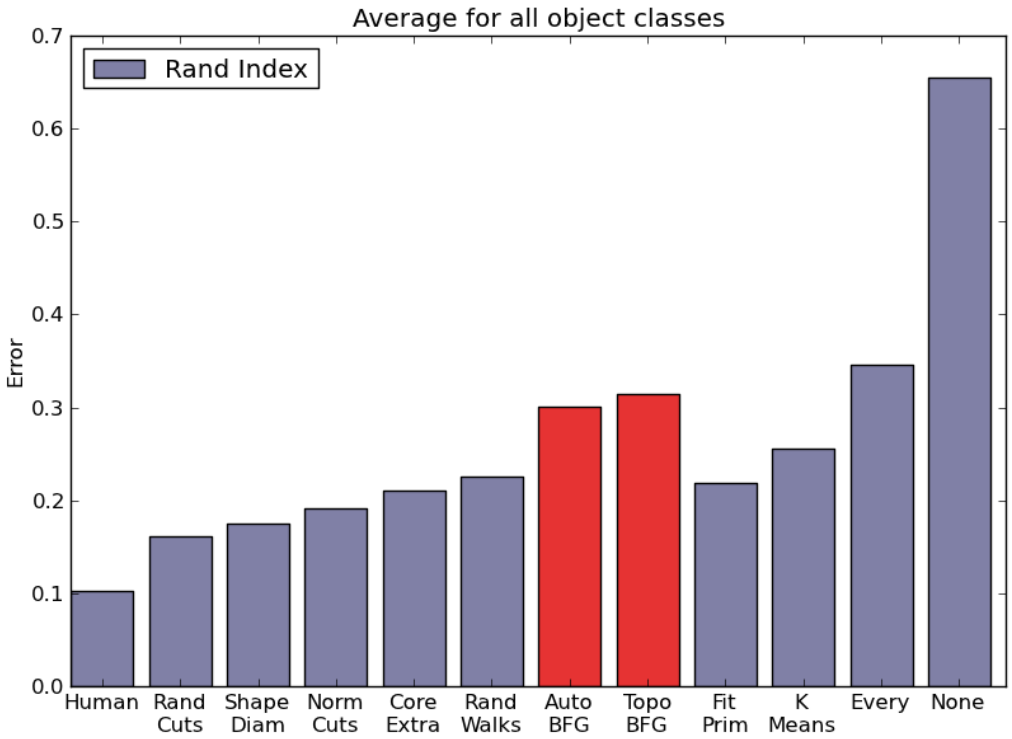
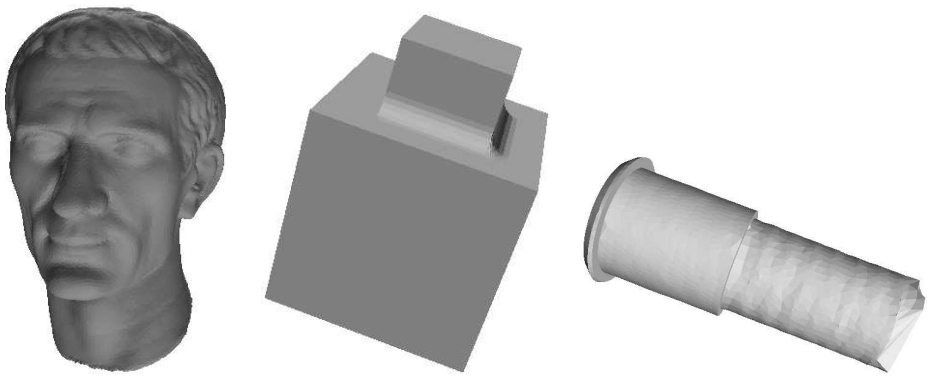


Figure 6.7: **Rand Index**: Comparison of segmentation algorithms on the shape database of Chen *et al*. [24]. Shown here are the Rand Index error values averaged over all object classes. This measures the unlikelihood that a pair of faces will agree on segment identity. The lower the number the more likely they will agree.

6.1.5 Discussion

There are a number of aspects in which our method is preferable to those tested by Chen. First, it is easy to implement and requires no geometric processing beyond simple centroid calculation. The other tested methods require an understanding of advanced geometrical concepts such as curvature or pre-processing of the mesh. Second, it requires no user input. Other than Core extraction and shape diameter function, all other methods require some level of user input to function, usually the number of desired segments. This aspect is the most relevant to our overall goal of



(a) Bust (b) Mechanical (c) Bearing

Figure 6.8: Three object classes from the Chen database that negatively affect our segmentation algorithm's performance. Objects with inherently static/subtle part distinctions (a) or little to no part structure (b,c) make poor candidates for breadth-first graph segmentation. They will tend not to be segmented at all (*i.e.* all vertices/faces will belong to the same segment).

automatic shape alignment. It would be overly burdensome for the user to have to manually tune the number of segments for each mesh prior to alignment.

Our methods were able to perform well on three of the four metrics given by Chen *et al.* [24]. Only in one metric, Rand Index, did our methods do significantly worse than the other seven algorithms. We believe that this is due to the particular way in which the $\hat{G}$ segments the mesh. That is, the presence of junction segments likely negatively affected performance.

Another issue affecting the performance of our $\hat{G}$ segmentation is its difficultly in dealing with objects with no pronounced parts or appendages. For object classes such as those in Figure 6.8, our algorithm essentially performs the same as "None", *i.e.* it places all faces in the same segment. Looking at these shapes from a topological perspective, they are almost identical, so it is no surprise that a topological approach to segmentation would do poorly. Also, our primary goal is the alignment of shapes

of the same object class based on their shared part structure, with a view towards dealing with significant part articulation. For largely static objects such as these, other alignment methods would certainly suffice.

Conclusions Overall, we have shown that our topological approach to segmentation performs well when compared to existing geometrically-based approaches. We applied the benchmark of Chen *et al.* [24] to our AutoBFG and TopoBFG segmentations. The performance of our methods on the benchmark shows that the breadth-first graph segments mesh shapes into semantic parts. The fact that ours is a fully automatic method increases its value as a tool for semantic shape analysis. This partially justifies its use in our larger problem of shape alignment. Full justification involves the curve-skeletons extracted using the breadth-first graph.

6.2 Skeleton Extraction

In order to know whether the skeletons produced by our breadth-first graph algorithm can be used fruitfully for shape alignment, we need some way of testing the overall quality of those skeletons. Cornea *et al.* [28] describe a set of desirable curve-skeleton properties. The properties were compiled from their analysis of the extraction literature and those applications in graphics and visualization that used skeletons. We will first describe those properties and establish which are the most relevant to shape alignment. We will then give the results and analysis of testing against the relevant properties.

Given some mesh M, we denote the skeleton of that mesh to be $Sk(M)$. The usefulness of that skeleton can be described as the extent to which it satisfies the following set of requirements/attributes.

- **Invariant to isometry**: Given an isometric (distance-preserving) transform T, the curve-skeleton of the transformed mesh $T(M)$ is the same as the transformed skeleton of the original mesh: $T(Sk(M)) = Sk(T(M))$.
- **Reconstruction**: The original mesh can be reconstructed from the information encoded in the skeleton.
- **Thin**: The skeleton $Sk(M)$ is a 1D curve, or is at most one voxel in thickness.
- **Centered**: The skeleton is centered within the object at all points.
- **Reliable**: Every point on the mesh surface is visible from at least one location on the skeleton. That is, for any surface point, there exists a straight-line path to the skeleton that does not intersect the surface.
- **Part preserving**: The skeleton distinguishes the different components of the mesh, reflecting the part structure.
- **Homotopic** (topology preserving): The skeleton $Sk(M)$ should have the same number of connected components (we include Cornea's "connectedness" property here), tunnels and cavities as the mesh M.
- **Robust**: The skeleton exhibits a low sensitivity to noise on the surface.

- **Smooth**: The variation of the curve tangent direction as we move along the curve-skeleton should be as small as possible.

- **Hierarchical**: The skeleton, as well as the process that produces it, reflects the natural hierarchy of the underlying object encoded by the mesh.

Our purpose in extracting curve-skeletons is to integrate the connectivity and relative distances of parts within the overall shape context into the process of alignment. As such, reconstruction, reliability and smoothness are largely irrelevant properties. Thinness of our skeleton is obvious, since our skeletons are straight-line graphs. The other six properties are of varying importance. We will now discuss, each in turn, the extent of their relevance to shape alignment and, if applicable, how well our algorithm satisfies each property.

6.2.1 Connectedness

Since our algorithm is based on a breadth-first traversal of mesh vertices, we are guaranteed to produce singly-connected skeletons for singly-connected meshes. However, for meshes with multiple connected components, we would have to make adjustments to our algorithm to take each component into account. In our actual implementation, we do allow for multiple components by selecting a random vertex from unseen components once the initial component (that which contains the centroid vertex, v_{cent}) has been traversed, but we lose isometry invariance. That is, the seed vertex of a breadth-first graph must be selected in an isometrically invariant way for

the graph to be invariant to isometry; random selection is not invariant. As such, when discussing the properties that follow, we are assuming a singly-connected mesh.

6.2.2 Isometry invariance

The BFG of a mesh is determined solely by the choice of seed vertices, not by the geometry of the mesh. To show invariance to isometry of the skeleton generated from the graph, it is sufficient to show that the choice of seed vertices is invariant to isometry. For the AutoBFG algorithm, this means showing $T(centroid(M)) = centroid(T(M))$, which we know is true in a distance-preserving isometry. For the TopoBFG algorithm, such invariance cannot be guaranteed, since the choice of initial seed vertex is random.

6.2.3 Centered

To demonstrate the performance of the BFG skeleton with respect to centeredness, we use a simplified approach of testing whether the skeleton is located within the surface and what percentage lies outside of the surface. We voxelize each mesh in the Chen dataset using a grid size of $128x128x128$ and the *binvox* software of Nooruddin *et al.* [60]. Each voxel of the grid is labeled internal or external. We then uniformly subsampled the BFG skeleton of each mesh and test whether each sample point is inside or outside the mesh according to the voxelization.

The BFG performs well under this condition for most classes of objects. Table 6.1 shows the performance on various object classes from the Chen database,

Class	# Samples	% Inside
Airplane	162.55	98.66
Armadillo	225.65	98.18
Pliers	162.45	98.08
Fish	101.25	97.71
Glasses	176.3	97.64
Hand	172.3	97.59
Ant	279.15	97.5
Human	210.15	97.0
Fourleg	185.7	96.85
Octopus	328.5	96.19
Teddy	137.25	95.84
Chair	333.5	95.5
Vase	109.25	93.68
Bearing	75.6	93.37
Bird	123.0	91.6
Bust	108.55	90.27
Table	230.0	87.83
Mech	82.25	85.21
Cup	106.65	39.58

Table 6.1: Performance of BFG skeletons for object classes in Chen database with respect to centeredness. For all but three classes, performance is better than 90%. Using a voxelization [60] of each mesh, we uniformly subsampled the BFG skeleton and tested each sample point's centeredness based on the voxelization. The second column shows the average number of skeleton sample points for each class, and the third column shows the average percentage of sample points labeled inside the mesh.

which is excellent ($> 90\%$) for 16 of the 19 classes. The cup class is the only class with poor performance. After visually inspecting the sample points that fall outside the mesh, we find, for most meshes, that these bad samples are located at the tips of parts and are mislabeled due to voxelization. That is, their faulty labeling is due to the resolution of the voxel grid, and not due to the skeleton actually falling outside of the surface. For the one object class with poor performance, the cup class, its topology is not well suited to constructing a centered curve-skeleton.

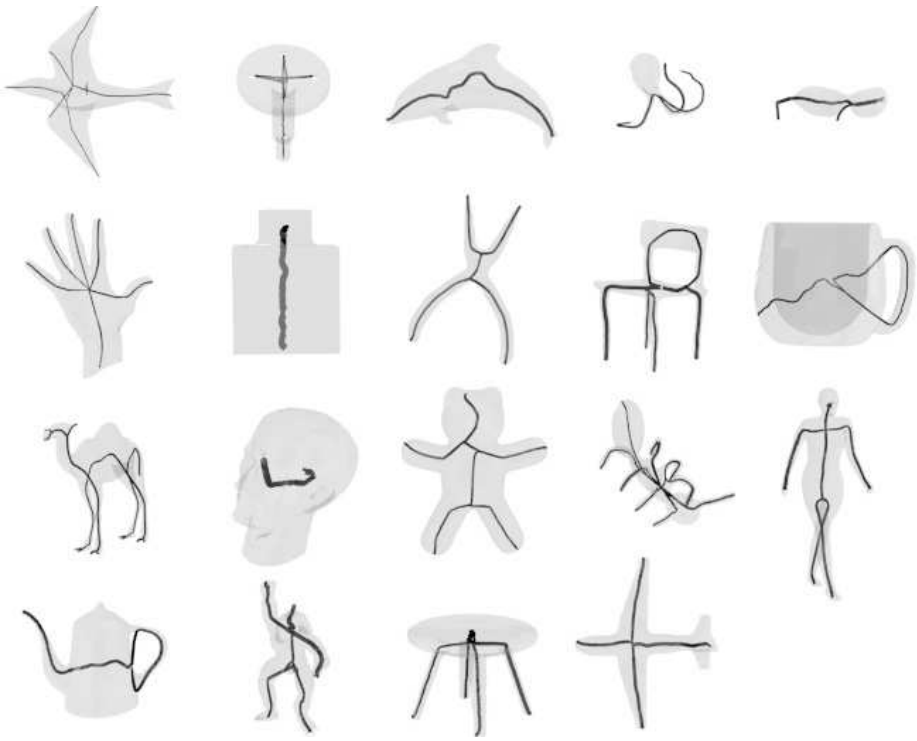


Figure 6.9: Various skeletons showing the part preserving nature of our algorithm. Notice that the skeletons reflect important topological characteristics. For example, those objects with surface genus greater than zero (cup, chair, teapot) have skeletons that contain the number of loops equal to their genus.

6.2.4 Part preserving

Part preservation of the BFG skeleton can be seen in Figure 6.9. These skeletons successfully capture the coarsest part structure of the articulated object classes. Important topological characteristics of the shapes, such as their surface genus, are also captured in these skeletons. For three classes (cup, chair and teapot), we can see that the number of loops in the skeleton is equal to the surface genus of these shapes.

Table 6.2 compares the number of junction and cap vertices extracted by the AutoBFG algorithm with the true number of junctions and caps in the underlying object. It shows that there is a high degree of consistency between the extracted and actual part structure. Avg_F is the average number of front vertices, *i.e.* junction

Class	Avg_F	Avg_J	$(\text{Avg}_J \pm 1)$	Avg_C	$(\text{Avg}_C \pm 1)$
Mech	81	0	100	2	100
Human	205	2	95	5	90
Glasses	176	0	95	2	90
Pliers	161	2	95	4	90
Fish	100	1	95	3	90
Ant	274	3	95	9	90
Bust	104	0	95	2	95
Cup	103	1	90	2	100
Armadillo	219	3	90	5	100
Vase	103	1	90	2	85
Airplane	159	2	90	6	50
Bearing	65	1	85	2	85
Table	225	2	85	6	65
Bird	119	1	85	4	60
Hand	169	3	80	5	95
Fourleg	181	4	75	7	80
Teddy	132	2	75	5	30
Octopus	324	3	65	8	90
Chair	331	4	55	5	90

Table 6.2: **AutoBFG part preservation**: For most object classes, the number of feature points extracted is highly consistent with the part structure. Avg_F, Avg_J and Avg_C are the average number of total front vertices (junctions + pipes + caps), average number of junction vertices, and average number of cap vertices, respectively, extracted by the AutoBFG algorithm. $(\text{Avg}_J \pm 1)$ is the percentage of meshes in the class whose number of junctions is within one of the average number of junction vertices. $(\text{Avg}_C \pm 1)$ is similar, except for caps and cap vertices vertices.

+ pipe + cap vertices. Avg_J and Avg_C are the average number of junction and cap vertices extracted by the AutoBFG algorithm, respectively. $(\text{Avg}_J \pm 1)$ and $(\text{Avg}_C \pm 1)$ are the percentage of meshes in the class whose number of junctions (or caps) is within one of these averages. These two measures make it clear that for most of the classes, the meshes in those classes have essentially the same number of junctions and caps. Since junctions and caps form the basis of part distinction, this demonstrates part preservation for our skeleton.

Class	Avg_F	Avg_J	$(\text{Avg}_J \pm 1)$	Avg_C	$(\text{Avg}_C \pm 1)$
Bust	101	0	100	2	100
Mech	79	0	100	2	100
Human	202	2	95	4	100
Fourleg	174	3	95	6	95
Ant	271	3	95	9	95
Pliers	162	2	95	4	90
Armadillo	214	2	95	4	90
Cup	101	1	90	2	100
Fish	101	1	90	3	90
Glasses	178	0	90	2	85
Bird	118	1	90	4	55
Bearing	65	1	85	2	95
Hand	168	3	85	5	90
Airplane	155	2	85	5	60
Table	235	2	85	6	45
Octopus	335	4	80	8	95
Chair	327	4	70	4	90
Vase	98	2	60	2	95
Teddy	133	2	55	4	40

Table 6.3: **Robustness of part structure**: Feature points from TopoBFG for each object class. When the initial priming seed vertex is chosen randomly (normally chosen to be the mesh vertex nearest the centroid of mesh vertices), the part structure within each object class is still preserved.

6.2.5 Robust

The robustness of the BFG to surface noise is immediately aparent when one remembers that surface geometry is only relevant to the choice of the initial priming seed, *i.e.* that mesh vertex which is closest to the centroid of mesh vertices. This robustness can be seen in Table 6.3. If the choice of initial priming seed were sensitive to noise, then we would see a marked drop in part preservation, yet the results in Table 6.3 show similar consistency rates as Table 6.2.

Of course, after the generation of the graph, surface noise does play a part in the generation of skeletal front vertices (as the centroid of the mesh vertices that

they encode). However, unless the noise level is severe, the noise transfered from the mesh surface to each skeletal front vertex is minimal. After Laplacian smoothing of the skeleton [79], all noise is essentially eliminated.

6.2.6 Homotopy

There are two aspects of homotopy to address, connectedness and surface genus. Connectedness has to do with the agreement between the number of connected components in the skeleton and its corresponding surface. The issue of surface genus involves the skeleton reflecting the number of tunnels or holes in the surface. For three object classes with genus greater than zero (cup, chair, teapot), the skeletons contain loops equal to the genus, seen in Figure 6.9.

6.2.7 Surface genus

Perfect reflection of the surface's genus in the skeleton would correspond to the number of loops in the skeleton being equivalent to the genus. For genus zero surfaces, it can be shown that our algorithm is guaranteed to produce skeletons that contain no loops, since it is based on breadth-first traversal. For surfaces of genus greater than zero, we cannot guarantee an equivalent number of loops, but we can achieve quality results. For example, in the case of the genus-3 Neptune mesh of Figure 4.3, the number of loops is exactly equivalent to the genus.

(a) (b) (c)

Figure 6.10: Capturing part hierarchy in $\hat{G}$ skeleton. Because our skeletons are based on the topology of the mesh, non-uniform meshing of the surface with respect to curvature (a) will lead to segmentations (b) and skeletons (c) that capture extended part heirarchies.

6.2.8 Hierarchical

Since the overarching goal of alignment is a minimal but semantically descriptive part structure, this property is not necessarily advantageous. As such, the methods presented here do not attempt to take into account multiple hierarchies of part structure. However, it is conceivable that being able to focus on a single part's substructure might be useful. A modification of our algorithm to further decompose individual parts will be left to possible future work.

In some cases, however, the part hierarchy is expressed directly within the topology of the mesh. When a shape has been non-uniformly meshed, the process that decides mesh density will inform the generation of breadth-first graph skeleton. Figure 6.10 shows an example of this. This camel mesh was generated such that it is

more densely sampled in areas of negative minimum curvature, *i.e.* seams and creases, that is the basis for the salient part distinctions mentioned in Section 2.2.1. Thus, that extra part information will be embedded in the breadth-first graph, *e.g.* the capturing of the individual toes and various facial features. In these areas, the mesh is much more densely sampled, affecting the topology, so our topological approach will be affected by that.

Conclusions We have shown in this section that the curve-skeletons generated by the AutoBFG and TopoBFG algorithms are suitable for analyzing the semantic structure of articulated shapes. They meet those properties of Cornea *et al.* [28] that are relevant to such analysis. They are topology preserving, invariant to isometry, centered, part preserving and robust. These qualities justify our use of those skeletons for shape alignment.

6.3 Conclusions

We have shown in this chapter that the breadth-first graph can be used for two major applications in shape analysis, mesh segmentation and curve-skeleton extraction. We have demonstrated that our topological method is able to perform as well as existing geometry-based methods on the benchmark for mesh segmentation of Chen *et al.* [24]. By adapting our method to suit the input requirements of the benchmark, we have shown its performance with respect to four different quality metrics against seven other well-established methods. We also have performed an analysis of the curve-skeletons generated by our breadth-first graph algorithm using the properties

developed by Cornea *et al*. [28]. We have shown that our skeletons are indeed usable for shape analysis. Finally, we provided qualitative and quantitative evidence that analysis of the semantic aspects of shape based on the breadth-first graph is justified. We can now address the problem of shape alignment.

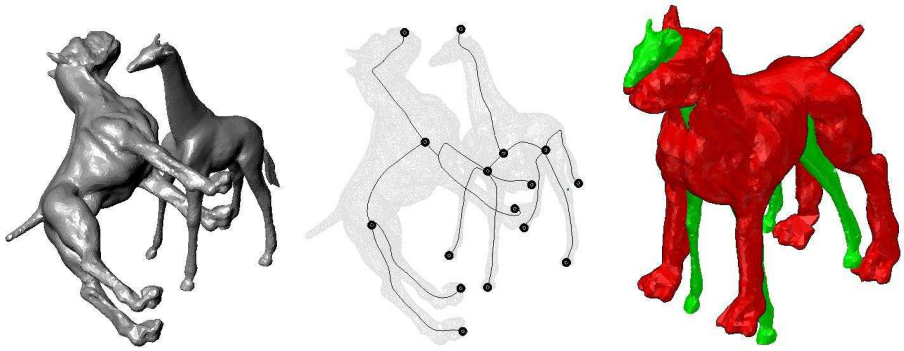
CHAPTER 7

AUTOMATIC SHAPE ALIGNMENT USING $\hat{G}$

We now turn our attention back to the original problem, shape alignment. In Chapter 4, we showed that the analysis of mesh topology was sufficient to extract a semantically meaningful part structure. We also provided a set of methods to produce a data structure, the breadth-first graph, that encodes that part structure in graph form. In this chapter, we will explain how to use this graph to align shapes of the same object class.

Our method can be classified as a global registration algorithm, one that leverages knowledge of the part structure of each input mesh to solve directly for the global minimum of the distance equation. The primary difference between our solution and other global registration algorithms is that ours is not based on correspondences between surface samples. Instead, we focus on building correspondences between relevant front vertices of the meshes' respective breadth-first graphs. Given those correspondences, we can solve for the minimizing transform in closed form (Figure 7.1).

The first section of this chapter describes how we use the breadth-first graph to solve the correspondence problem. The second presents our complete algorithm for automatic shape alignment. Finally, we discuss the process by which we tested our alignment algorithm and give the results of our experiments.



(a) Extract skeletons from mesh topology and feature points from those skeletons

(b) Correspond features and solve for rigid transform

Figure 7.1: Outline of our alignment algorithm based on the topological shape analysis introduced in Chapter 4. Given shapes P and Q, find their respective breadth-first graph skeletons, use those skeletons to find semantically relevant point correspondences, then use those point correspondences to solve for the rigid transform that minimizes their pairwise distance.

7.1 The Correspondence Problem

Recalling our discussion of surface registration in Chapter 2, we know that the problem of registration can be decomposed into two sub-problems: correspondence and alignment. Given some point correspondence, we must find the rigid transform that minimizes the Euclidean distance between corresponding point pairs. Since the minimizing transform can be solved in closed form (Arun [7]), the primary problem is finding the correct point correspondence. An optimal solution would require an exhaustive search of the total correspondence space, *i.e.* all possible point pairs of each input surface, which is prohibitively expensive. Instead, local surface registration techniques use a simple, closest-point correspondence solution that is refined iteratively, precluding the need for exhaustive search but leading to sensitivity to initial position. Global registration techniques select only a few points from each input

surface, based on their uniqueness within some feature space, and then exhaustively search the much smaller correspondence space.

In our case, because we have insight into the logical part structure of each mesh, we are able to limit the correspondence space even further to a relatively small number of points. These points are the most semantically dissimilar fronts of the breadth-first graph of each input mesh, *i.e.* those points that represent part boundaries (junction fronts) and those that represent part tips (cap fronts). We further limit correspondence space by precluding the correspondence of caps to junctions and *vice versa*. So, we limit our correspondence space to (junction, junction) and (cap, cap) pairs and solve for the optimal point correspondence within that space.

7.1.1 Correspondence error

One question remains: for a point correspondence within our limited correspondence space, how do we evaluate its quality so that we can compare it with other correspondences? This metric should of course reflect Euclidean distance after alignment. However, there are additional aspects of the correspondence that are relevant to determining overall quality, aspects that are available to us because the points in the correspondence are embedded within a graph structure. These aspects, which are associated with geodesic distances computed as distances along the curve-skeleton embedding of the breadth-first graph, should also be integrated into our metric.

Let P and Q be meshes with respective breadth-first graphs $\hat{G}_P$ and $\hat{G}_Q$. Let $\text{junc}(P)$ and $\text{junc}(Q)$ be the respective junction vertices of $\hat{G}_P$ and $\hat{G}_Q$. Let $\text{cap}(P)$

and cap(Q) be their respective cap vertices. The correspondence set between the breadth-first graphs of P and Q is:

$$\begin{aligned}\mathbf{PQ} = \{(\hat{p}_i, \hat{q}_i) :&(\hat{p}_i \in \text{junc}(P), \hat{q}_i \in \text{junc}(Q)) \ \vee \\ &(\hat{p}_i \in \text{cap}(P), \hat{q}_i \in \text{cap}(Q))\}\end{aligned} \tag{7.1}$$

Two common error metrics for point correspondences are the *correspondence root mean squared* (cRMS) and *distance root mean squared* (dRMS) error metrics. Adapting these metrics to our purposes, the cRMS error of a particular correspondence set $\mathbf{PQ} = \{(\hat{p}_i, \hat{q}_i)\}$ between meshes P and Q is given by:

$$\text{cRMS}^2(\mathbf{PQ}) = \min_{\mathbf{R},\mathbf{t}} \frac{1}{n} \sum_{i=1}^{n} ||(\mathbf{R}\hat{p}_i + \mathbf{t}) - \hat{q}_i||^2 \tag{7.2}$$

where $\mathbf{R}$ and $\mathbf{t}$ are a rotation and translation, respectively. This metric requires that the minimizing rotation $\mathbf{R}$ and translation $\mathbf{t}$ be computed prior to error calculation, which is not ideal. Also, those aspects associated with graph distance, such as geodesic distance, are unaffected by rigid transformation, so cRMS error cannot take them into account.

The dRMS error of correspondence $\mathbf{PQ}$ is computed by summing all internal pairwise distances of the two point sets:

$$\text{dRMS}^2(\mathbf{PQ}) = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} (\text{dist}(\hat{p}_i, \hat{p}_j) - \text{dist}(\hat{q}_i, \hat{q}_j))^2 \tag{7.3}$$

The dist$(\hat{p}_i, \hat{p}_j)$ function is usually defined as the Euclidean distance between points $\hat{p}_i$ and $\hat{p}_j$, but it need not be so. It is for this reason that we chose the dRMS error metric. Because this function can be arbitrarily defined, we can integrate the aspects associated with the breadth-first graph along with Euclidean distance.

Let $\hat{v}_i$ and $\hat{v}_j$ be front vertices, $\mathbf{v}_i$, $\mathbf{v}_j$ the centroids of their corresponding mesh vertices, gd$(\hat{v}_i, \hat{v}_j)$ the distance between them along the skeleton graph (described in Section 4.3), $agd(\hat{v})$ the average graph distance (*i.e.* the average distance of $\hat{v}$ to all other junction and cap vertices in the graph), and agdDiff$(\hat{v}_i, \hat{v}_j) = |agd(\hat{v}_i) - agd(\hat{v}_j)|$ the difference between their respective average graph distances (*i.e.* the average distance of each front to all other junction and cap points in the graph). We then define the distance between two front vertices $\hat{v}_i$ and $\hat{v}_j$ as follows:

$$\text{dist}(\hat{v}_i, \hat{v}_j) = ||\mathbf{v}_i - \mathbf{v}_j|| + \text{gd}(\hat{v}_i, \hat{v}_j) + \text{agdDiff}(\hat{v}_i, \hat{v}_j) \tag{7.4}$$

7.1.2 Correspondence algorithm

Using the error metric of Equations 7.3 and 7.4, we can determine the best correspondence from some set of correspondences. The final question then is how to generate a set of plausible correspondences of (junction, junction) and (cap, cap) pairs. Of course, this could be done exhaustively by searching the entirety of correspondence space, and such an approach is manageable for shapes with non-trivial but relatively simple part structures. However, for shapes with more complex part structures, *e.g.* octopuses having eight legs or those meshes mentioned in Section 6.2.8 having extra part information embedded in the topology (Figure 6.10), the $O(N!)$ cost of

an exhaustive search can become expensive. Instead, we would like to have a greedy algorithm that achieves usable results without the cost of an exhaustive search.

We now present and algorithm to find the best correspondence **PQ** between two mesh shapes P and Q from their corresponding breadth-first graphs $\hat{G}_P$ and $\hat{G}_Q$ (Algorithm 5). It can be best understood in this way: imagine that the skeletons based on $\hat{G}_P$ and $\hat{G}_Q$ are made of string, with junctions and caps as knots in the string, cap knots colored blue and junction knots colored red. Select a knot in $\hat{G}_P$ and select a knot in $\hat{G}_Q$ of the same color. Hold one knot in each hand and let the rest of the graph hang limp. For every knot $\hat{l}_P$ in $\hat{G}_P$'s skeleton, we look for the knot in $\hat{G}_Q$'s skeleton that agrees in color, and whose distance from the selected knot in $\hat{G}_Q$ best matches the distance of $\hat{l}_P$ from the selected knot in $\hat{G}_P$, using the distance metric (Eq. 7.4). These pairs then define the correspondence set **PQ**. So for every junction and cap $\hat{v}_i \in \text{junc}(P) \cup \text{cap}(P)$, there will be a potential correspondence set $\mathbf{PQ_i}$. We then choose the $\mathbf{PQ}_i$ that minimizes Equation 7.3.

The only caveat to this method has to do with degenerate skeletons, *i.e.* skeletons that contain no junctions. These could be skeletons of surfaces that are strongly toroidal (leading to a circular loop skeleton) or spherical (leading to a line segment skeleton). In the latter case of a line segment, we add a third knot color, for pipes, and add a knot for the pipe vertex that is in the topological "middle" of the segment, *i.e.* has index equal to 1/2 the total segment size. In the case of circular skeletons, we arbitrarily chose a vertex and one of its neighbors as caps and treat it as a line segment skeleton above.

Algorithm 5 BuildCorrespondence($\hat{G}_P, \hat{G}_Q$)

Require: BFGs $\hat{G}_P = \{\hat{V}_P, \hat{E}_P\}$ and $\hat{G}_Q = \{\hat{V}_Q, \hat{E}_Q\}$
  $\hat{F}_P = \text{junc}(P) \cup \text{cap}(P)$
  $\hat{F}_Q = \text{junc}(Q) \cup \text{cap}(Q)$
  $\mathbf{PQ}_{all} = \emptyset$
  for all $\hat{v}_P \in \hat{F}_P$
    Let $\hat{L}_P$ be $\hat{F}_P - \{\hat{v}_P\}$
    for all $\hat{v}_Q \in \hat{F}_Q$
      Let $\hat{L}_Q$ be $\hat{F}_Q - \{\hat{v}_Q\}$
      Let $\mathbf{PQ} = \{(\hat{v}_P, \hat{v}_Q)\}$
      for all $\hat{l}_P \in \hat{L}_p$
        Find $\min_{\hat{l}_Q \in \hat{L}_Q} |\text{dist}(\hat{v}_P, \hat{l}_P) - \text{dist}(\hat{v}_Q, \hat{l}_Q)|$, such that $\hat{l}_P$ and $\hat{l}_Q$ are same front type and dist is the distance function of (7.4)
        if $\hat{l}_Q$ exists **then**
          $\mathbf{PQ} = \mathbf{PQ} \cup (\hat{l}_P, \hat{l}_Q)$

      $\mathbf{PQ}_{all} = \mathbf{PQ}_{all} \cup \mathbf{PQ}$

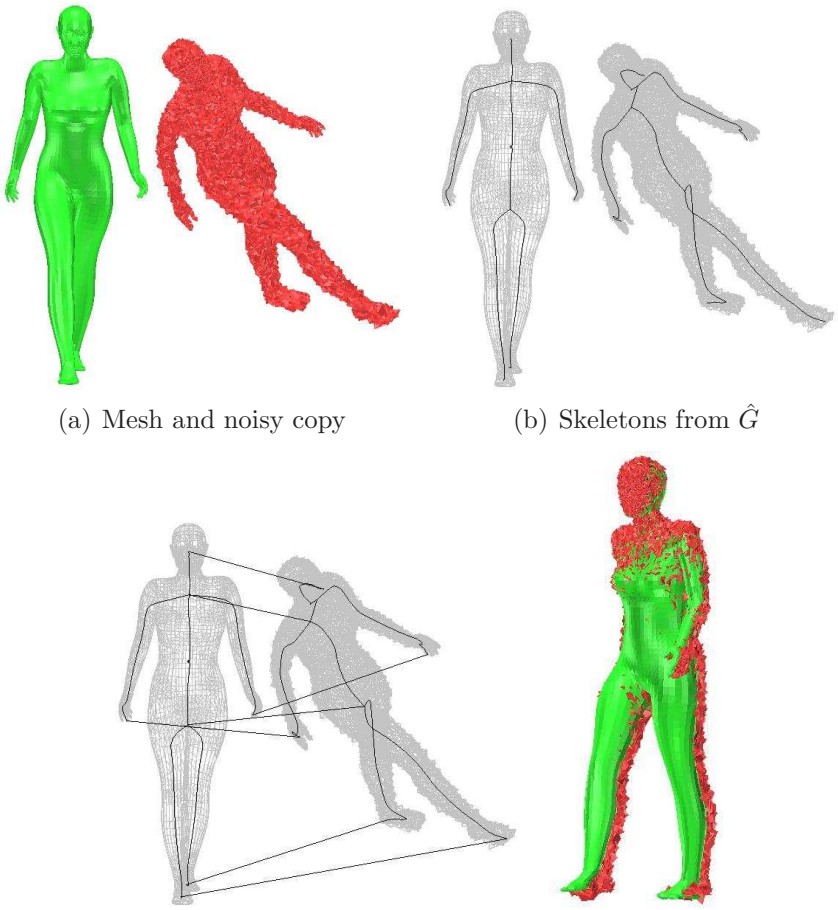
  $\mathbf{PQ}_{final} = \min_{\mathbf{PQ} \in \mathbf{PQ}_{all}} \text{dRMS}(\mathbf{PQ})$

7.2 Automatic Alignment Algorithm

Given the above means of finding point correspondences, we now present our algorithm for automatic mesh alignment. Let mesh P and mesh Q have corresponding breadth-first graphs $\hat{G}_P$ and $\hat{G}_Q$. We shall find the rotation R and translation T that minimizes the least squares distance between the junctions and caps of $\hat{G}_P$ and $\hat{G}_Q$, $\hat{F}_P = \text{junc}(P) \cup \text{cap}(P)$ and $\hat{F}_Q = \text{junc}(Q) \cup \text{cap}(Q)$, respectively.

1. Build a correspondence **PQ** between the junctions and caps of $\hat{G}_P$ ($\hat{F}_Q$) and those of $\hat{G}_Q$ ($\hat{F}_Q$) using Algorithm 5.

2. Solve for R and T in closed form using Arun [7]. That is, prepare for ICP by approximately aligning P and Q.



(a) Mesh and noisy copy (b) Skeletons from $\hat{G}$

(c) Correspondence (d) Alignment

Figure 7.2: Alignment process with mesh and copy of itself with noise introduced. Given a mesh and noisy copy (a), the breadth-first graph $\hat{G}$ and corresponding skeleton of each mesh is generated (b). Using the junctions and caps of $\hat{G}$, a correspondence space is built (c), where junctions must correspond to junctions and caps to caps. After sorting correspondence space by dRMS error, we solve for rotation and translation in closed form based on the correspondence with smallest dRMS error. ICP is then used to refine the alignment.

3. Run ICP on $\hat{F}_P$ and $\hat{F}_Q$ until convergence.

Figure 7.2 demonstrates the alignment process for a mesh and a noisy copy of itself.

Our alignments are determined from correspondences of front vertices in the breadth-first graphs $\hat{G}_P$ and $\hat{G}_Q$ given as input. As such, we could use either our AutoBFG or TopoBFG algorithms as the basis of our alignment. We will refer to align-

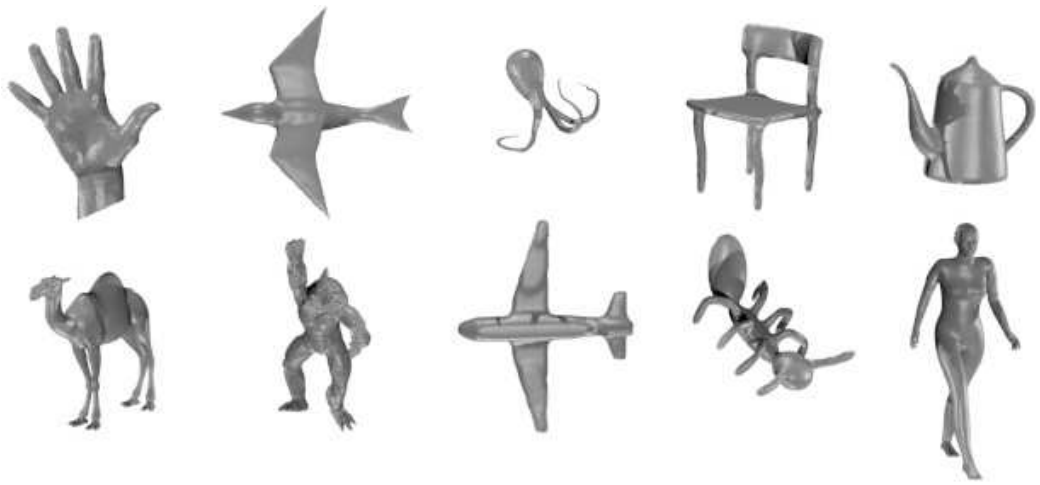
ments based on the AutoBFG and TopoBFG algorithms as *AutoBFG* and *TopoBFG alignments*, respectively. In Section 7.3 below, we test both of these versions of our alignment algorithm.

7.3 Results

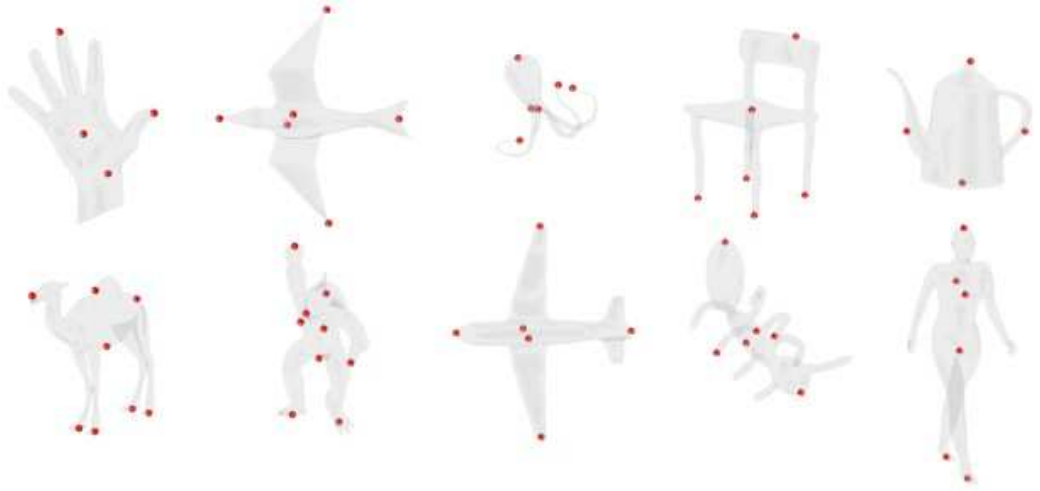
In this section, we will show that our algorithm was able to successfully align shapes within the same object class and out-perform existing surface registration software on object classes with articulated part structures. We first discuss how we prepared our dataset for testing by manually adding semantic landmarks and how those landmarks are used to establish a quantitative expression of alignment quality. We then discuss the various input conditions that any shape alignment system should be expected to handle and how we modified the test data to take those conditions into account. Finally, we provide the results of testing under those conditions.

7.3.1 Landmarks and landmark error

From the Chen dataset discussed in Chapter 5, we chose 10 object classes with 20 meshes in each class for a total of 200 meshes (Figure 7.3(a)). Our test procedure involved performing pairwise alignments for all objects within each class. In order to provide a quantitative measure of alignment quality we decided to use surface landmarks (vertices) and record the pairwise distance of those landmarks after alignment.



(a) Classes selected from Chen database



(b) Semantic landmarks for object classes

Figure 7.3: Object classes from Chen *et al*. [24] used in our alignment experimentation and testing. In order: hand, bird, octopus, chair, vase, fourleg, armadillo, airplane, ant and human. For each mesh (a) in these classes, we selected landmark vertices (b) corresponding to semantically relevant areas of the mesh (tops of heads, tips of fingers, *etc*.). We then ran our alignment algorithm and recorded the mean distance of all pairs of corresponding landmarks.

Landmark selection

We selected a number of semantically relevant landmark points for each class (*e.g*. tops of heads, tips of wings, *etc*.) and manually recorded the vertices in each mesh that corresponded to these landmarks (Figure 7.3(b)). These landmarks represented

both the relatively stable core regions as well as those regions that could vary widely due to changes in pose. We then used those landmarks as the basis for evaluating the quality of alignment of two mesh shapes.

Landmark error

For some mesh pair (M_P, M_Q), whose landmark vertices are $L_P \subset V_P$ and $L_Q \subset V_Q$, respectively ($|L_P| = |L_Q| = N$), and some aligning transform α_{PQ}, the error associated with α_{PQ} is:

$$E(M_P, M_Q, \alpha_{PQ}) = \frac{1}{N} \sum_{i=1}^{N} ||\alpha_{PQ}(\mathbf{p}_i) - \mathbf{q}_i||, \mathbf{p}_i \in L_P, \mathbf{q}_i \in L_Q \tag{7.5}$$

That is, we defined the error of an alignment to be the mean of the pairwise distances of corresponding mesh landmarks, after alignment.

It should be noted that our alignment algorithm is not symmetric. That is, given M_P and M_Q, α_{PQ} (*i.e.* the transform found by our algorithm to align M_P to M_Q) is not necessarily the same as α_{QP}. We found in practice that $\alpha_{PQ} \approx \alpha_{QP}$ when $M_P \approx M_Q$ (*i.e.* differ only by an isometry and/or per-vertex noise). However, when M_P and M_Q are different shapes, it is often the case that $\alpha_{PQ} \neq \alpha_{QP}$. Since the graphs associated with M_P and M_Q are different, the optimal correspondences **PQ** and **QP** can also be different, which leads to different minimizing transforms. If the two transforms are different, then their respective landmark errors will be different, so we used the average of their respective landmark errors. Therefore, we define the

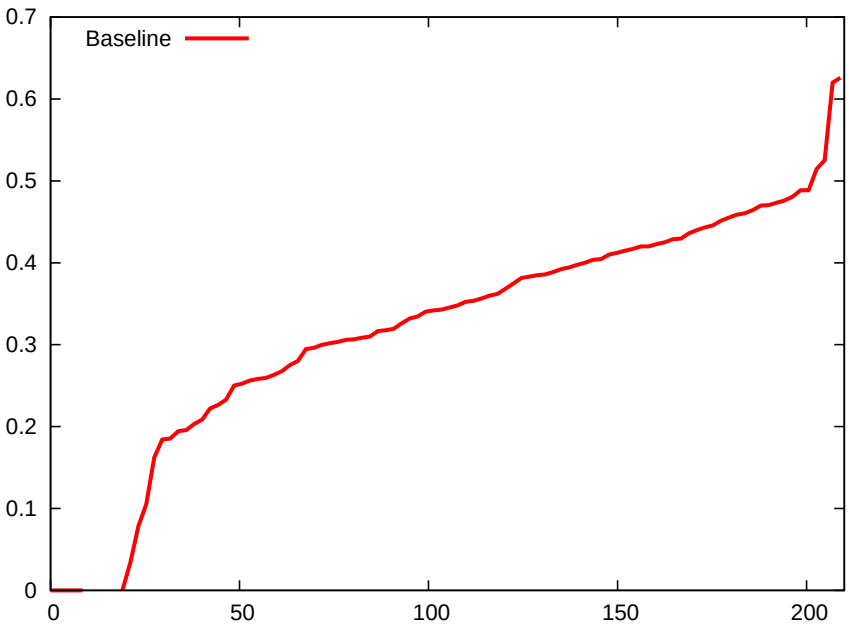


Figure 7.4: Performance curve representing the baseline (inherent) alignment error of the Armadillo object class. This curve is generated by first calculating the 210 error values (Equation 7.6) of alignments based on the known landmark correspondences, sorting those values and plotting them as a line graph. The total baseline error for the class, then, is the area under this curve.

total error associated with aligning M_P and M_Q to be:

$$\mathcal{E}(M_P, M_Q) = \begin{cases} \frac{1}{2}\left(E(M_P, M_Q, \alpha_{PQ}) + E(M_Q, M_P, \alpha_{QP})\right) & \text{if } M_P \neq M_Q \\ E(M_P, M_Q, \alpha_{PQ}) & \text{if } M_P \approx M_Q \end{cases} \tag{7.6}$$

where $M_P \approx M_Q$ means that they are the same mesh, possibly modified with per-vertex noise.

Performance curves For each object class (20 meshes), there will be 210 total error values for any particular alignment algorithm. These 210 values are composed of $\binom{20}{2}$ error values for alignments where $M_P \neq M_Q$ plus 20 error values where $M_P \approx M_Q$. If we were to sort these values and plot them as a line graph, we would

have what we will refer to as a *performance curve* of the algorithm. We then can define the overall performance of an algorithm on an object class as the area underneath the performance curve. Figure 7.4 shows the performance curve of the baseline algorithm (discussed below) of the Armadillo object class.

Caveat about performance curves: It should be noted that when comparing multiple performance curves for different alignment algorithms, the x coordinate of a curve represents simply the index in its sorted list of error values. Given some x, the y values of two curves are *not* necessarily the error of the same shape pair. These performance curves are given as a clean representation of overall performance of an algorithm for an object class, not as a means to compare how they performed on a particular mesh pair in that class.

Baseline error and algorithm

In the case where $M_P \neq M_Q$, especially when their parts are in different poses, a perfect alignment of landmarks (*i.e.* one where the error of Equation 7.6 is zero) is unlikely, even with prior knowledge of landmark correspondences. If we use the known correspondences as input to Arun's least-squares method, it will produce the distance minimizing transform of that correspondence. We will refer to this hereafter as the *baseline algorithm*. This can be used to calculate a performance curve (*e.g.* Figure 7.4) for the baseline (or inherent) error of that class. In later Figures (7.5, 7.6, 7.8, and 7.9), this baseline algorithm will be presented with the other alignment algorithms as a hard lower bound.

7.3.2 Input conditions to test

Given object shapes **P** and **Q** and their mesh representations M_P and M_Q, all algorithms must be tested under the following input conditions:

(i) Identical meshes ($M_P = M_Q$). Algorithm must align M_P to a rigidly transformed version of itself.

Testing: We generated two random rigid transforms $X_0 = (R_0, T_0)$ and $X_1 = (R_1, T_1)$ and gave $X_0(M_P)$ and $X_1(M_Q)$ as input.

(ii) Identical meshes with noise ($M_P \approx M_Q$). Algorithm must align M_P to a rigidly transformed version of itself whose vertices have been modified with random noise.

Testing: We generated a noisy version $\bar{M}$ of each mesh M in each object class. For each vertex in M, we generated a random unit vector (*i.e.* point on the unit sphere) and scaled it by 1/100th of the length of the mesh's bounding-box diagonal. We then translated each vertex by its corresponding noise vector. Using the random transforms (X_0, X_1), we gave $X_0(M_P)$ and $X_1(\bar{M}_Q)$ as input.

(iii) Similar shapes of the same class ($M_P \neq M_Q$). Algorithm must align different meshes that are of the same object class. This implies significant surface variation as well as pose variation.

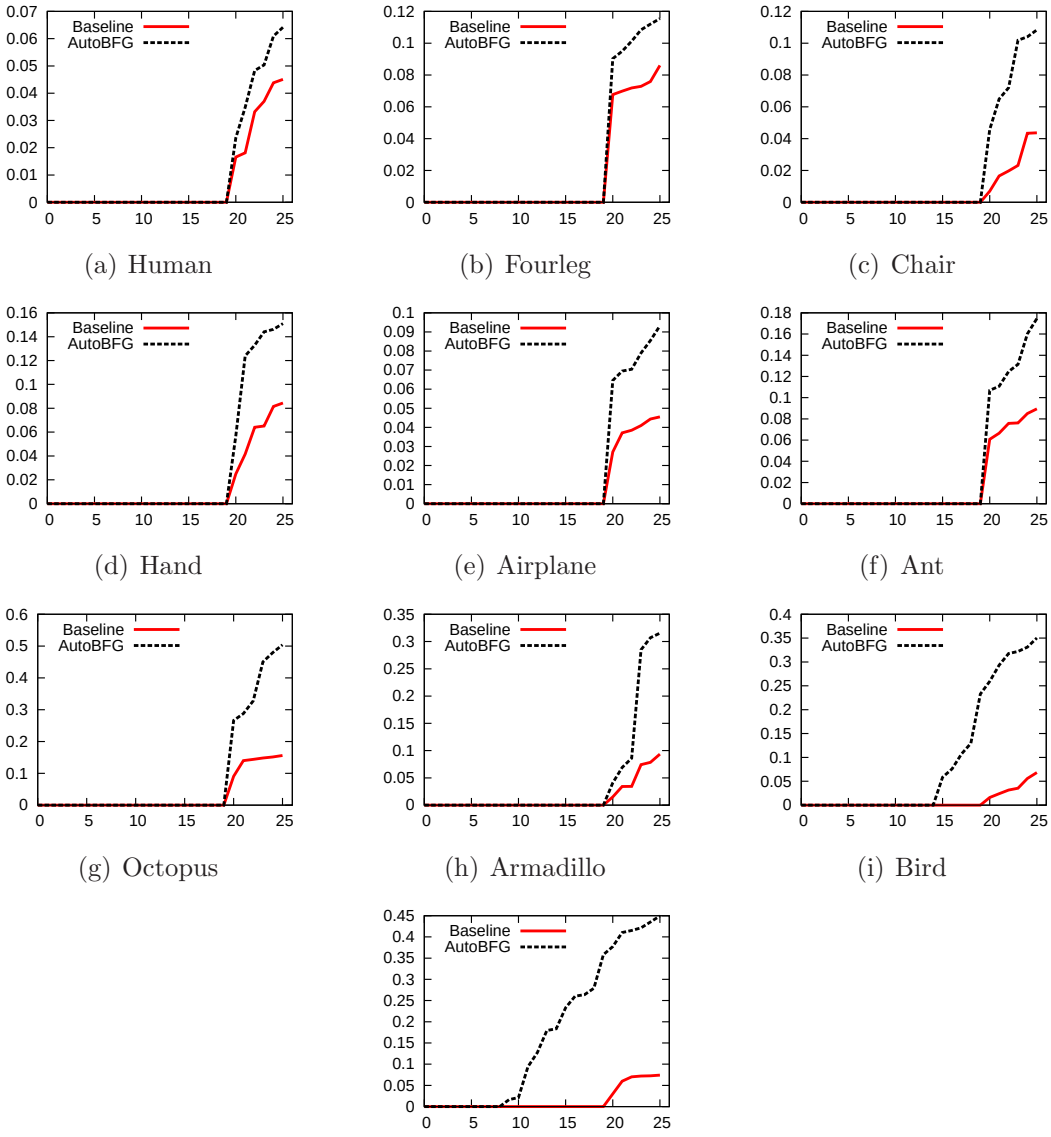
Testing: Using the random transforms (X_0, X_1) we give $X_0(M_P)$ and $X_1(M_Q)$ as input. We tested this under noise conditions as well.

7.3.3 Testing results

For each of the conditions mentioned above, we prepared the data appropriately and tested the two versions of our alignment algorithm (AutoBFG and TopoBFG alignments) on that data. We expected our AutoBFG alignment to perform near-perfectly under conditions (i) and (ii), due to its invariance to isometry. For condition (iii) we tested our AutoBFG method on the data with the expectation of good performance. We did not know what to expect from TopoBFG on conditions (iii), but given its commendable performance with respect to segmentation (Section 6.1), we thought it had the potential to perform well. As the lower bound for performance, we used the baseline algorithm mentioned above in Section 7.3.1. As an upper bound, we used an implementation of Generalized ICP of Segal *et al.* [66] discussed in Section 3.4.

(i) Alignment of identical meshes

When aligning a mesh M_P to itself, our AutoBFG algorithm was able to align the landmarks perfectly, within machine tolerance, for 8 of the 10 object classes, *i.e.* $E(M_P, M_P, \alpha_{PP}) = 0$. Figure 7.5 shows the performance curves of the AutoBFG and baseline algorithms for their 25 best mesh pairs. We can see from the performance curves of 8 of the 10 classes that both algorithms have an error of zero for the first 20 mesh pairs, corresponding to those pairs where $M_P = M_Q$. Intuitively, since our breadth-first graph algorithm is guaranteed to produce the same graph for the same mesh given the same seed vertices, and since properties such as the centroid of a mesh and straight-line distance are invariant to rigid transformation, we are guaranteed to



(a) Human (b) Fourleg (c) Chair

(d) Hand (e) Airplane (f) Ant

(g) Octopus (h) Armadillo (i) Bird

(j) Vase

Figure 7.5: Performance curves AutoBFG and baseline alignments. In the first 8 classes (a-h), the first 20 error values ($x \in [0, 19]$) correspond to the error of aligning each mesh in the class with a rigidly transformed version of itself.

find the same starting seed vertex (that nearest the centroid) for identical meshes, thus producing identical graphs.

However, in two classes, bird and vase, it did not perform perfectly. We believe this is a result of our AutoBFG producing straight line skeletons (*i.e.* skeletons with

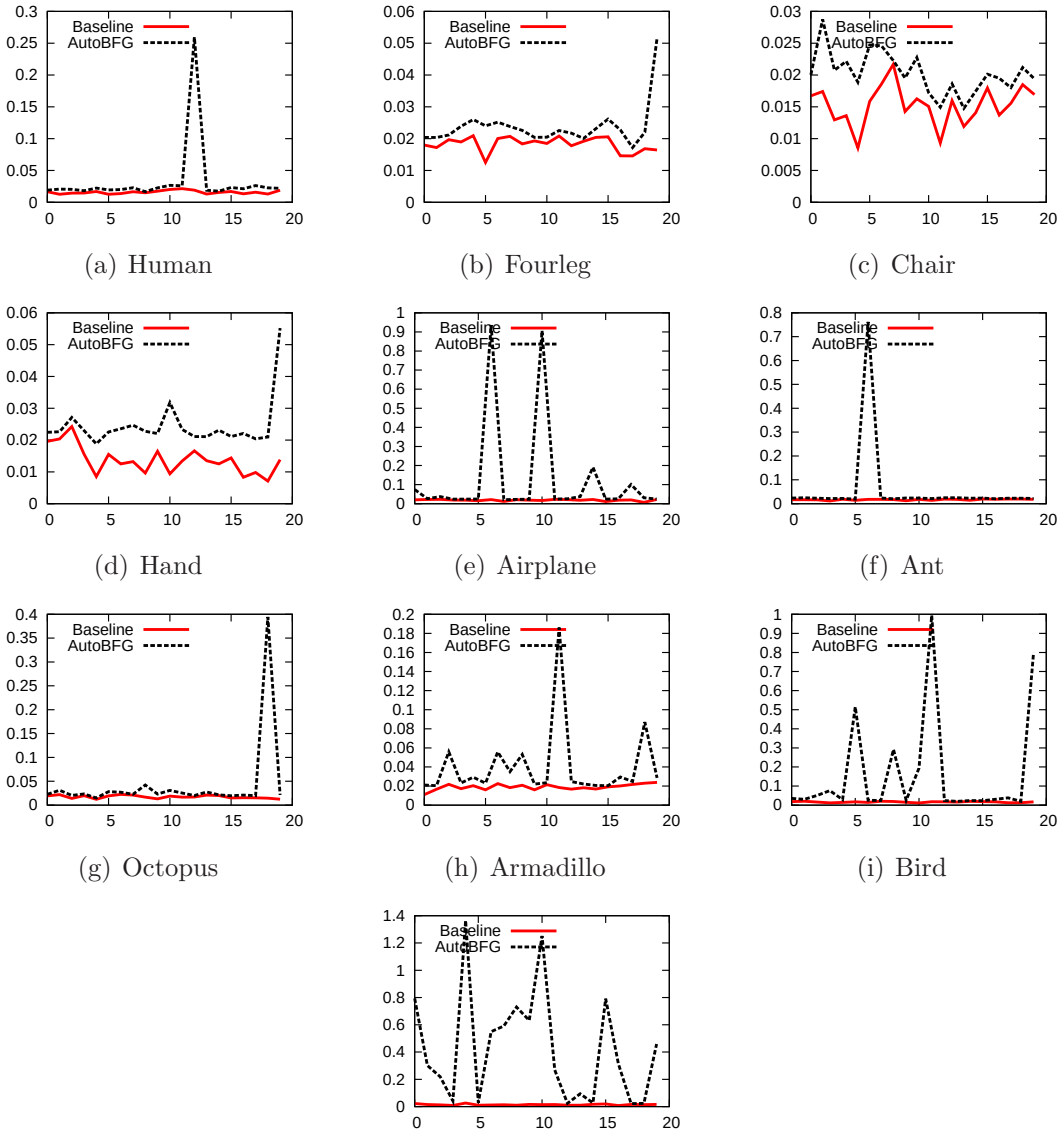
no junctions) for some meshes in the class. For the vase class, this is understandable. For the bird class, this has to do with the fact that in some cases the wings were so large in relation to the head and tail that the head and tail sections were distilled out of the graph. If we recall back to our discussion in Section 7.1.2, we handled the case of having only two points in a straight line segment (the cap vertices) by adding a third point (a pipe vertex) in between them. Unfortunately, if that third point is sufficiently collinear to the others, this leads to an underconstrained least-squares solution, with unpredictable results. In other words, even if the point correspondences are correct, the rigid transform output by Arun will likely be incorrect.

Alignment of identical meshes with noise

In the face of noise, our method was able to achieve results very similar to those results without noise. Figure 7.5 shows a 1-to-1 comparison between the AutoBFG and baseline algorithms for aligning the 20 meshes of each object class to a noisy version of themselves. In three cases (Figures 7.6(b), 7.6(c), and 7.6(d)), the AutoBFG algorithm had almost exactly the baseline's performance, while in two more (Figures 7.6(a) and 7.6(f)) it misaligned only one mesh.

Alignment of similar shapes in same object class

Both our AutoBFG and TopoBFG alignment algorithms were able to perform as well as Generalized ICP on all 10 object classes, outperforming it on 8 of the 10. Figure 7.7 demonstrates visually a subset of the results from three of the best-



(a) Human (b) Fourleg (c) Chair

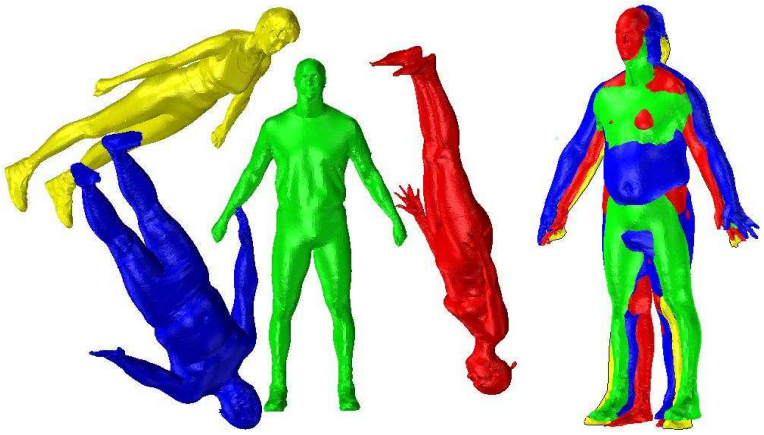
(d) Hand (e) Airplane (f) Ant

(g) Octopus (h) Armadillo (i) Bird

(j) Vase

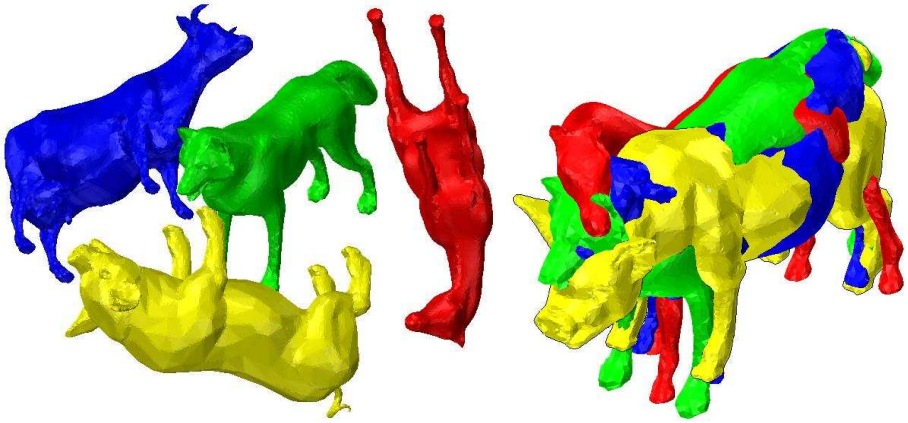
Figure 7.6: 1-to-1 comparison of alignment error for identical meshes under noisy conditions. Our AutoBFG alignment was able to perform very near to baseline in 6 of the ten classes (a, b, c, d, f, g).

performing classes, Human, Hand and Fourleg, to better explain what it means to align similar shapes of the same object class. Figure 7.8 shows the performance curves for AutoBFG, TopoBFG, Generalized ICP and baseline.



(a) Human meshes aligned to man in center

(b) Hands of various shapes, aligned to center mesh



(c) Several four-legged animals aligned to a wolf

Figure 7.7: Alignment results for three different object classes. In each case, the peripheral meshes were aligned to the center mesh. The original orientation of each mesh was chosen randomly.

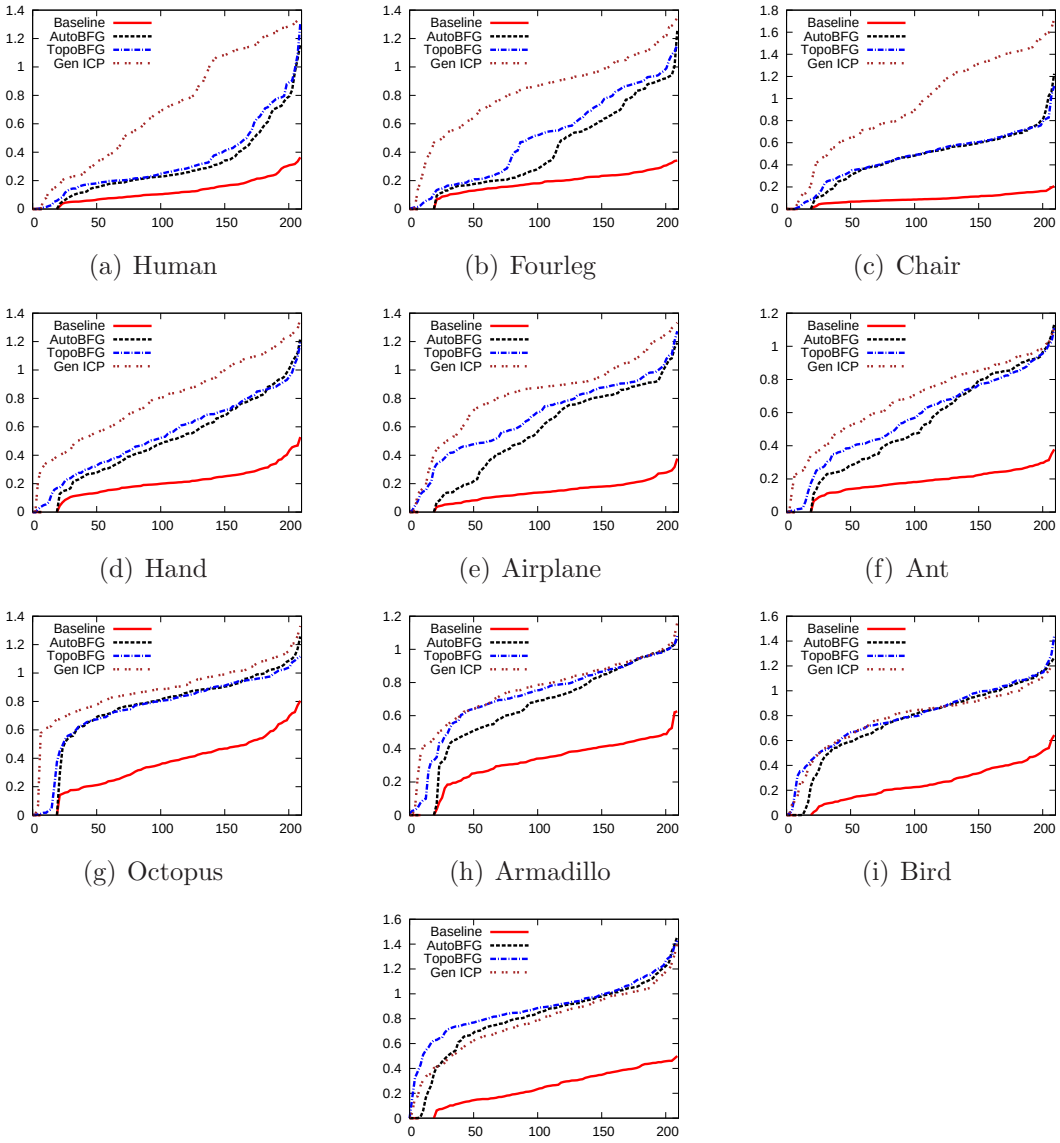
Notice that our algorithms perform better when the objects have highly articulated part structures. The two classes with performance curves nearest the baseline, Human and Fourleg, show that both the AutoBFG and TopoBFG algorithms perform best when articulation is highest but pose variation is moderate. Performance on the Chair and Hand classes, which are highly symmetric and asymmetric, respectively, indicate that symmetry is not a defining attribute. The Airplane class demonstrates the most disparity between the two BFG algorithms. We believe that the relatively poor performance in the Ant, Octopus and Armadillo classes is due to the extreme pose variation in those classes.

Figure 7.9 shows the performance curves of the AutoBFG, TopoBFG, Generalized ICP and baseline algorithms on alignments of M_P to a noisy version of M_Q. In this case, our alignment algorithms were able to outperform Generalized ICP on 7/10 object classes. This demonstrates our alignment algorithms' robustness to noise.

Running time analysis

For large databases, our alignment algorithm can be optimized in a simple way to significantly decrease total alignment running time. This can be seen in the graph of Figure 7.10. In this graph, each object class has four bars. The first bar (blue) represents the total average time per alignment for Generalized ICP. The second bar (yellow) represents the total average time per alignment for our AutoBFG-based alignment, and it is composed of these steps:

1) building the BFG for each input shape,



(a) Human (b) Fourleg (c) Chair

(d) Hand (e) Airplane (f) Ant

(g) Octopus (h) Armadillo (i) Bird

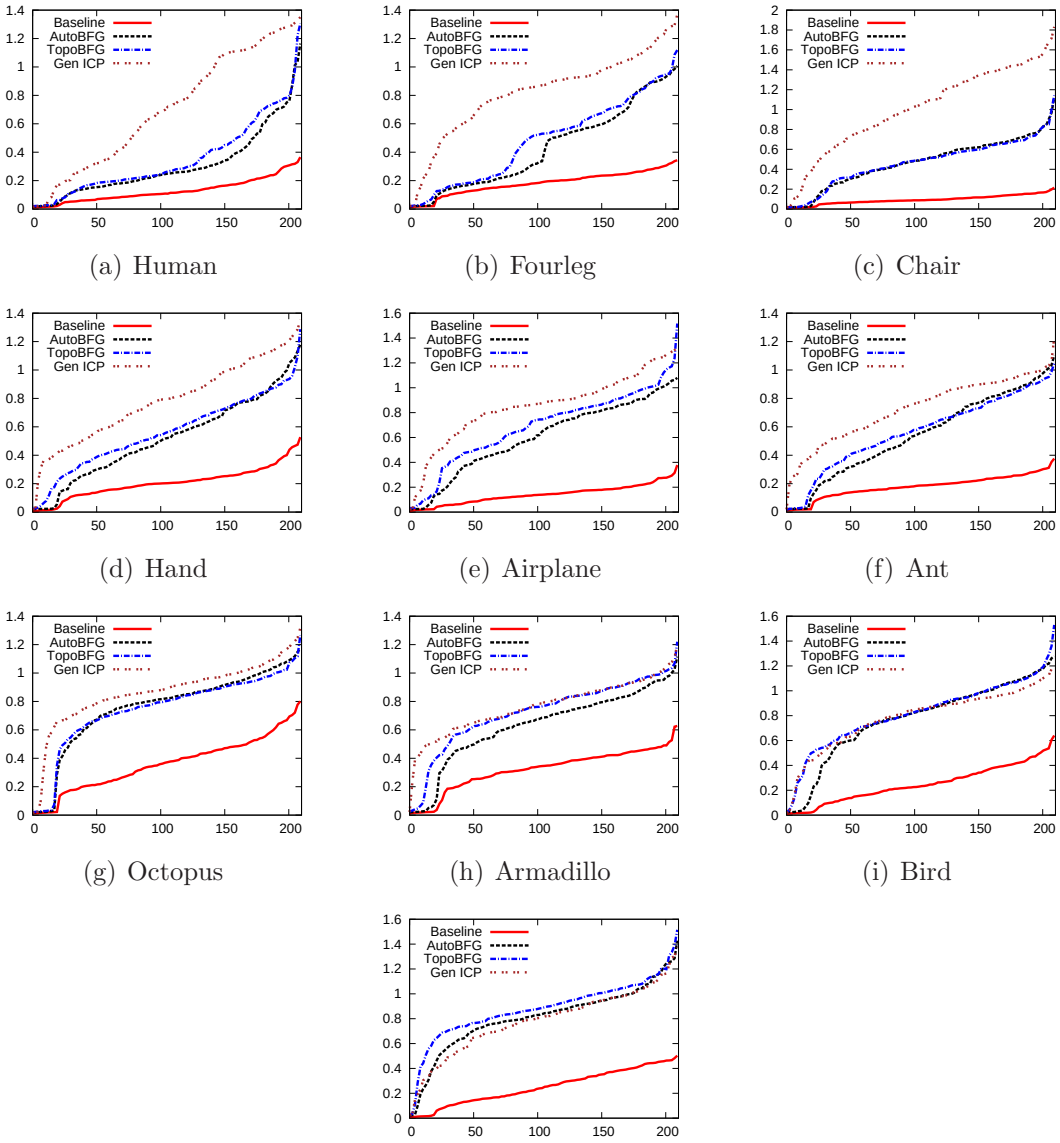
(j) Vase

Figure 7.8: Performance curves for AutoBFG, TopoBFG, Generalized ICP and the baseline alighnments. For 8 of the 10 classes, both our algorithms outperformed Generalized ICP. Only Bird and Vase are again anomalous (although comparable) because of their lack of well-defined topology.

2) finding correspondences,

3) performing the initial alignment (prepping for ICP), and

4) refining the alignment via ICP.



(a) Human (b) Fourleg (c) Chair

(d) Hand (e) Airplane (f) Ant

(g) Octopus (h) Armadillo (i) Bird

(j) Vase

Figure 7.9: Performance curves for AutoBFG, TopoBFG, Generalized ICP and the baseline alighnments when aligning each mesh M_P with a noisy version of M_Q. Again, in 8 of the 10 classes, both the AutoBFG and TopoBFG algorithms outperformed Generalized ICP.

The third bar (red) represents the total average time per alignment for building the breadth-first graph for each input shape (step 1). The fourth bar (green) corresponds

to the average time per alignment for steps 2, 3 and 4. The vast majority of the time per alignment is spent building the breadth-fist graphs of input shapes.

If these graphs were generated once per mesh prior to alignment, total running time can be decreased dramatically. In our implementation, this approach was used. The difference in total alignment time for all 10 tested classes was more than an order of magnitude. As an aside, the timing results for our TopoBFG-based alignment were similar enough to these that our analysis still applies.

It should also be noted that every aspect of our algorithm was implemented in the Python scripting language with very little optimization. Generalized ICP was implemented in C/C++ with a fair amount of optimization. This leads us to believe that a C/C++ implementation of our algorithm would provide even more gains in speed.

Differences in topology

One difficulty in extracting part structure from mesh topology is that the topology must somehow reflect the part structure. For example, if the arms of a mesh are completely fused to the body, then topology alone is not enough to find them. This is demonstrated in Figure 7.11. The two women are in roughly the same pose, but the arms of the woman on the left are entirely connected to her body, whereas the arms of the other woman are disconnected from the body. Figure 7.11(b) shows how this leads to significantly different $\hat{G}$ skeletons for both. However, we can see from Figure 7.11(e) that our method is still able to align these meshes. This

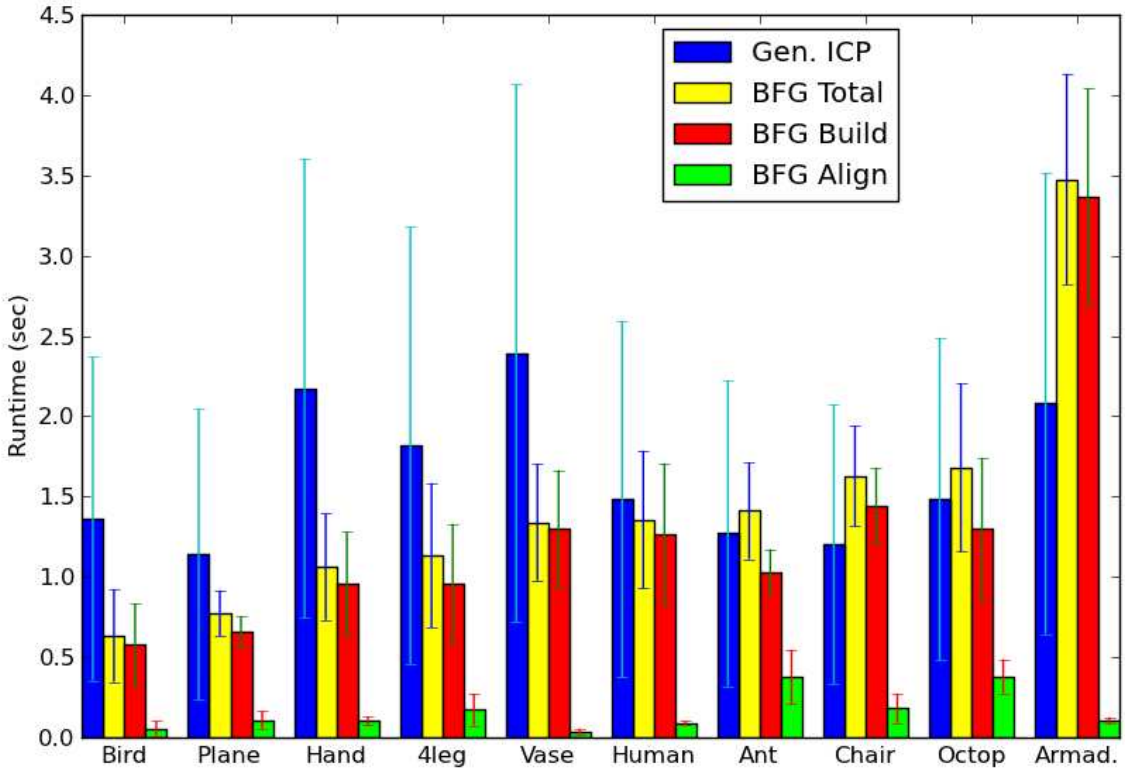


Figure 7.10: Running time results for Generalized ICP and BFG alignment. Each bar represents average time per alignment in seconds, and the T-bars represent the standard deviation. The blue and yellow bars are the total time per complete alignment for Generalized ICP and our AutoBFG-based algorithm, respectively. Timing results for our algorithm (yellow bar) take into account building the BFG for each input shape every time. The red bar represents the average time per alignment spent on building the BFG for each input shape. The green bar is the average time per alignment for correspondence, initial alignment, and refinement via ICP. By pre-processing the BFGs for input shapes, running time can be significantly increased. Timing results for our TopoBFG-based alignment algorithm are essentially the same.

is due to the fact that even though the graph for the woman on the right contains more junction vertices (four more where the arms leave and rejoin the body), only the junction at the legs will have the internal pairwise graph distances that best minimize dRMS error. We can see this correspondence relationship in Figure 7.11(d).

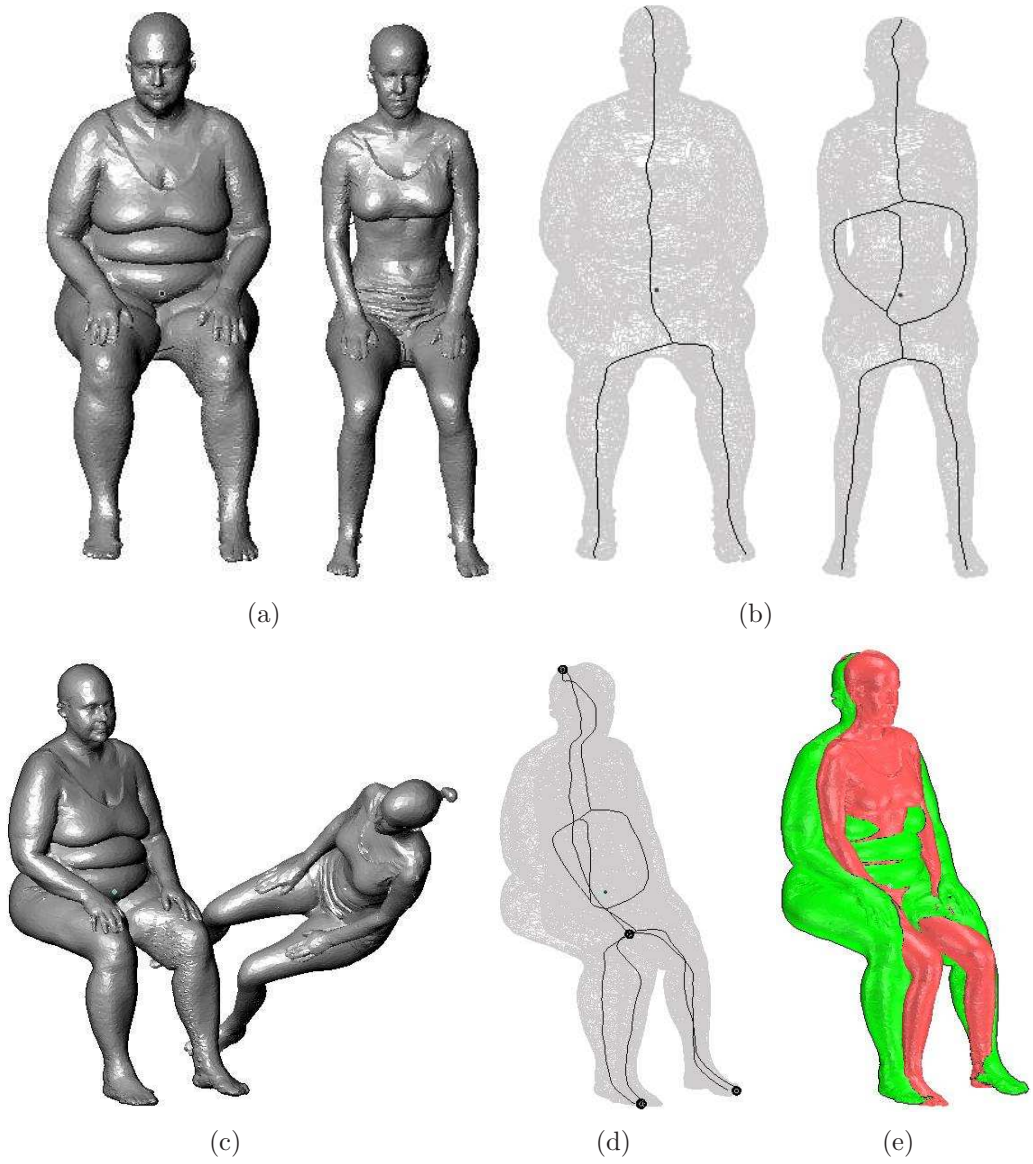


Figure 7.11: Even if there are significant topological differences between input meshes (a), our method can still find the aligning transform for (c). The breadth-first graphs are different (b), but correspondence is based upon similarity of internal graph distance, as well as Euclidean distance (d). Even if there is extra topological information in one mesh, such as the arms of the woman on the right, the correct correspondence is still found (d-e).

7.4 Conclusions

We have shown in this chapter that our topological approach to shape analysis, the breadth-first graph, can be used fruitfully to perform alignments on mesh shapes

of the same object class. We have shown that these algorithms are robust to surface variation due to noise and changes in pose. We have provided extensive testing and quantitative evidence that mesh topology is a valuable resource when trying to align shapes.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

8.1 Validity of Hypotheses

In our introductory chapter, we introduced two hypotheses. The first stated that the integration of the semantic aspects of shape (*i.e.* the part structure) would significantly improve its robustness to surface variation. The second stated that it was possible to use an analysis of mesh topology to understand the semantic aspects of shape critical to the first hypothesis. In this work we have shown that these hypotheses are valid and worthy of further study. We will discuss the validity of the second hypothesis first. Then we will discuss the validity of the first.

8.1.1 Hypothesis: Shape analysis through mesh topology

In this work, we have shown that the shape analysis has a basis in combinatoric topology. The triangle mesh can be viewed as a topological space, specifically as a simplicial 2-complex. We can apply primitive operations on simplicial complexes, *i.e.* closure, star and link, to create a scalar function defined on a simplicial 2-complex. This function can be used to analyze the shape within the context of Morse theory as a Discrete Reeb Graph.

We have shown that this manner of analysis on mesh topology provides a significant amount of semantic part information. To extract that part information, we

have provided a novel approach based on a modified breadth-first traversal of mesh vertices. We have shown how to encode the part information into a graph structure and how that graph can be used for shape analysis tasks such as segmentation and skeleton extraction. We have shown how to overcome the breadth-first graph's dependence on user input by using an initial priming traversal of the mesh. We use the result of that priming run to construct a final graph in an automated manner.

We have shown that our method garners valuable and useful information about the semantic part structure. We have tested and justified the ability of this analysis method to extract part information by using an established benchmark for mesh segmentation. We have also shown that the curve-skeletons produced by our method are valid and usable for our goal of shape alignment according to established evaluation criteria.

8.1.2 Hypothesis: Improving shape alignment through semantic analysis

We have shown that our breadth-first graph can be used to perform alignments on mesh shapes of the same object class. We limit the scope of correspondence space to only those vertices in these graphs that were most semantically dissimilar, junctions and caps. By employing a greedy graph correspondence solution to limited correspondence space, we are able to solve for the rigid transform that minimizes the distance between (junction, junction) and (cap, cap) pairs. We have tested our alignment algorithms against a well-establish alignment solution, the Generalized ICP.

Our tests show that our algorithms outperform Generalized ICP over most object classes and are robust to surface variation due to noise and changes in pose.

8.2 Future Work

In future, there are other application areas to which we would like to apply our topological approach to shape analysis.

- **Shape reconstruction from BFG**: We believe that our method can be used to reconstruct the basic shape characteristics of a mesh based on the pure topology. This would be very similar to the techniques of Isenburg *et al.* [42] in their work on connectivity shapes. Given the edge information of a mesh, we would use the TopoBFG algorithm to produce an abstract (*i.e.* non-geometric) skeleton of mesh that we could embed in $\mathbb{R}^3$ randomly. We then could apply multi-dimensional scaling to "unwind" the randomized skeleton to a canonical form. Finally, we could rebuild the vertex geometry locally at each front based on the connectivity between fronts.

- **Applications to graph visualization**: One thing that has not been explored in this work is the fact that our breadth-first graph algorithm only requires vertices and edges. It does not require that the input be a triangle mesh or a surface of any type. We limited ourselves to simplicial 2-complexes for the sake of establishing our shape analysis within a discrete version of Morse theory and to apply it to the specific application area of shape alignment. We could also apply it to discover and visualize clusters within densely connected abstract

graphs. Essentially, given some graph $G = \{V, E\}$ junctions within $\hat{G}(G)$ whose degree is above some threshold β could be labeled as cluster sites, reducing the overall visualization load.

- **Integration into Blender3D**: The functionality of the methods described in this work, everything from the generation of the breadth-first graph of a mesh to colorization of a mesh based on part decomposition to skeletonization to alignment, are available as a Blender 2.5 addon. We would like to augment this functionality to include *auto-rigging* of meshes (constructing a skeleton of bones which are used within Blender's inverse kinematics engine for animation) and possibly even a rudimentary shape retrieval system for ad-hoc shape databases.

LIST OF REFERENCES

[1] Agathos, A., Pratikakis, I., Papadakis, P., Perantonis, S., Azariadis, P., and Sapidis, N. 3D articulated object retrieval using a graph-based representation. *The Visual Computer* (2010), 1–19.

[2] Agathos, A., Pratikakis, I., Perantonis, S., and Sapidis, N. Protrusion-oriented 3D mesh segmentation. *The Visual Computer 26*, 1 (2010), 63–81.

[3] Agathos, A., Pratikakis, I., Perantonis, S., Sapidis, N., and Azariadis, P. 3D mesh segmentation methodologies for CAD applications. *Computer-Aided Design and Applications 4*, 6 (2007), 827–841.

[4] Aiger, D., Niloy, M., and Cohen-Or, D. 4-points congruent sets for robust pairwise surface registration. *ACM Transactions on Graphics 27*, 3 (2008), 85–85.

[5] Amenta, N., Choi, S., and Kolluri, R. K. The power crust. In *SMA '01: ACM Symposium on Solid Modeling and Applications* (New York, NY, USA, 2001), ACM, pp. 249–266.

[6] Anguelov, D., Srinivasan, P., Pang, H., Koller, D., Thrun, S., and Davis, J. The correlated correspondence algorithm for unsupervised registration of nonrigid surfaces. In *Advances in Neural Information Processing Systems* (2005), The MIT Press, p. 33.

[7] Arun, K., Huang, T., and Blostein, S. Least-squares fitting of two 3-D point sets. *IEEE Trans. Pattern Anal. Mach. Intell. 9*, 5 (1987), 698–700.

[8] Attene, M., Falcidieno, B., and Spagnuolo, M. Hierarchical mesh segmentation based on fitting primitives. *Vis. Comput. 22*, 3 (2006), 181–193.

[9] Attene, M., Katz, S., Mortara, M., Patane, G., Spagnuolo, M., and Tal, A. Mesh segmentation - a comparative study. In *SMI '06: Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006* (Washington, DC, USA, 2006), IEEE Computer Society, p. 7.

[10] Berretti, S., Del Bimbo, A., and Pala, P. 3D mesh decomposition using Reeb graphs. *Image and Vision Computing 27*, 10 (2009), 1540–1554.

[11] Besl, P. J., and McKay, N. D. A method for registration of 3-D shapes. *IEEE Trans. Pattern Anal. Mach. Intell. 14*, 2 (1992), 239–256.

[12] Biasotti, S., Attali, D., Boissonnat, J.-D., Edelsbrunner, H., Elber, G., Mortara, M., di Baja, G. S., Spagnuolo, M., Tanase, M., and Veltkamp, R. Shape analysis and structuring. *Mathematics and Visualization* (2008), 145–183.

[13] Biasotti, S., Marini, S., Mortara, M., and Patané, G. An overview on properties and efficacy of topological skeletons in shape modelling. In *SMI '03: Proceedings of the Shape Modeling International 2003* (Washington, DC, USA, 2003), IEEE Computer Society, p. 245.

[14] Biasotti, S., Marini, S., Spagnuolo, M., and Falcidieno, B. Sub-part correspondence by structural descriptors of 3d shapes. *Computer-Aided Design 38*, 9 (2006), 1002–1019.

[15] Blanz, V., Tarr, M., Bülthoff, H., and Vetter, T. What object attributes determine canonical views? *Perception 28* (1999), 575–600.

[16] Blender. Blender 3D. In *http://www.blender.org*.

[17] Blender. Blender Model Repository. In *http://e2-productions.com/repository/*.

[18] Blum, H. A transformation for extracting new descriptors of shape. *Models for the Perception of Speech and Visual Form 19*, 5 (1967), 362–380.

[19] Bouaynaya, N., Charif-Chefchaouni, M., and Schonfeld, D. Spatially variant morphological restoration and skeleton representation. *Image Processing, IEEE Transactions on 15*, 11 (2006), 3579–3591.

[20] Bronstein, A., Bronstein, M., and Kimmel, R. Topology-invariant similarity of nonrigid shapes. *International journal of computer vision 81*, 3 (2009), 281–301.

[21] Chang, W., and Zwicker, M. Automatic registration for articulated shapes. In *Computer Graphics Forum* (2008), vol. 27, Wiley Online Library, pp. 1459–1468.

[22] Chaouch, M., and Verroust-Blondet, A. A novel method for alignment of 3d models. In *Shape Modeling and Applications, 2008. SMI 2008. IEEE International Conference on* (june 2008), pp. 187 –195.

[23] Chaouch, M., and Verroust-Blondet, A. Alignment of 3d models. *Graphical Models 71*, 2 (2009), 63–76.

[24] Chen, X., Golovinskiy, A., and Funkhouser, T. A benchmark for 3D mesh segmentation. *ACM Transactions on Graphics (SIGGRAPH) 28*, 3 (2009).

[25] Chen, Y., and Medioni, G. Object modeling by registration of multiple range images. In *IEEE International Conference on Robotics and Automation* (1991), pp. 2724–2729.

[26] Chuang, J., Tsai, C., and Ko, M. Skeletonisation of three-dimensional object using generalized potential field. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 22*, 11 (2000), 1241–1251.

[27] Cornea, N., Demirci, M., Silver, D., Shokoufandeh, A., Dickinson, S., and Kantor, P. 3d object retrieval using many-to-many matching of curve skeletons.

[28] Cornea, N., Silver, D., and Min, P. Curve-skeleton properties, applications, and algorithms. *IEEE Transactions on Visualization and Computer Graphics 13*, 3 (2007), 530–548.

[29] de Goes, F., Goldenstein, S., and Velho, L. A hierarchical segmentation of articulated bodies. In *Computer Graphics Forum* (2008), vol. 27, Blackwell Science Ltd, Osney Mead, Oxford, OX 2 0 EL, UK,, pp. 1349–1356.

[30] Dey, T., and Sun, J. Defining and computing curve-skeletons with medial geodesic function. In *Symposium on Geometry Processing* (2006), Eurographics Association, p. 152.

[31] Edelsbrunner, H., and Harer, J. *Computational topology: an introduction*. Amer Mathematical Society, 2010.

[32] Field, D. Laplacian smoothing and delaunay triangulations. *Communications in Applied Numerical Methods 4*, 6 (1988), 709–712.

[33] Fu, H., Cohen-Or, D., Dror, G., and Sheffer, A. Upright orientation of man-made objects. In *SIGGRAPH 2008* (New York, NY, USA, 2008), ACM, pp. 1–7.

[34] Gagvani, N., and Silver, D. Parameter-controlled volume thinning. *Graphical Models and Image Processing 61*, 3 (1999), 149–164.

[35] Gelfand, N., Mitra, N. J., Guibas, L. J., and Pottmann, H. Robust global registration. In *SGP 2005: Symposium on Geometry Processing* (Aire-la-Ville, Switzerland, Switzerland, 2005), Eurographics Association, p. 197.

[36] Giorgi, D., Biasotti, S., and Paraboschi, L. Shape retrieval contest 2007: Watertight models track. *SHREC competition* (2007).

[37] Golovinskiy, A., and Funkhouser, T. Randomized cuts for 3D mesh analysis. *ACM Transactions on Graphcs (SIGGRAPH ASIA) 27*, 3 (Dec. 2008).

[38] Grimson, W. E. L. *Object recognition by computer: the role of geometric constraints*. The MIT Press, 1991.

[39] Hart, J. Morse theory for implicit surface modeling. *Springer-Verlag, Berlin* (1998), 257–268.

[40] Hoffman, D. D., and Singh, M. Salience of visual parts. *Cognition 63* (1997), 29–78.

[41] Huang, Q., and Dom, B. Quantitative methods of evaluating image segmentation. In *Image Processing, 1995. Proceedings.*, vol. 3, IEEE, pp. 53–56.

[42] Isenburg, M., Gumhold, S., and Gotsman, C. Connectivity shapes. In *VIS '01: Visualization* (Washington, DC, USA, 2001), IEEE Computer Society, pp. 135–142.

[43] Katz, S., Leifman, G., and Tal, A. Mesh segmentation using feature point and core extraction. *The Visual Computer (Pacific Graphics) 21*, 8-10 (October 2005), 649–658.

[44] Katz, S., and Tal, A. Hierarchical mesh decomposition using fuzzy clustering and cuts. In *SIGGRAPH 2003* (New York, NY, USA, 2003), ACM, pp. 954–961.

[45] Kazhdan, M. An approximate and efficient method for optimal rotation alignment of 3d models. *IEEE Trans. Pattern Anal. Mach. Intell.* (2007), 1221–1229.

[46] Kresch, R., and Malah, D. Skeleton-based morphological coding of binary images. *Image Processing, IEEE Transactions on 7*, 10 (1998), 1387–1399.

[47] Lai, Y., Hu, S., Martin, R., and Rosin, P. Rapid and effective segmentation of 3D models using random walks. *Computer Aided Geometric Design 26*, 6 (2009), 665–679.

[48] Lee, Y., Lee, S., Shamir, A., Cohen-Or, D., and Seidel, H.-P. Mesh scissoring with minima rule and part salience. *Comput. Aided Geom. Des. 22*, 5 (2005), 444–465.

[49] Levoy, M., Pulli, K., Curless, B., Rusinkiewicz, S., Koller, D., Pereira, L., Ginzton, M., Anderson, S., Davis, J., Ginsberg, J., Shade, J., and Fulk, D. The digital Michelangelo project: 3D scanning of large statues. In *SIGGRAPH 2000* (New York, NY, USA, 2000), pp. 131–144.

[50] Lian, Z., Godil, A., Bustos, B., Daoudi, M., Hermans, J., Kawamura, S., Kurita, Y., Lavoué, G., Nguyen, H., Ohbuchi, R., et al. Shrec11 track: Shape retrieval on non-rigid 3d watertight meshes.

[51] Lien, J.-M., Keyser, J., and Amato, N. M. Simultaneous shape decomposition and skeletonization. In *SPM '06: Proceedings of the 2006 ACM symposium on Solid and physical modeling* (New York, NY, USA, 2006), ACM, pp. 219–228.

[52] Makadia, A., and Daniilidis, K. Direct 3d-rotation estimation from spherical images via a generalized shift theorem.

[53] Maragos, P., and Schafer, R. Morphological skeleton representation and coding of binary images. *Acoustics, Speech and Signal Processing, IEEE Transactions on 34*, 5 (1986), 1228–1244.

[54] Martin, D., Fowlkes, C., Tal, D., and Malik, J. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics.

[55] Martinek, M., and Grosso, R. Optimal rotation alignment of 3D objects using a gpu-based similarity function. *Computers & Graphics 33*, 3 (2009), 291–298.

[56] Mohlenkamp, M. A fast transform for spherical harmonics. *Journal of Fourier analysis and applications 5*, 2 (1999), 159–184.

[57] Mortara, M., and Patané, G. Affine-invariant skeleton of 3d shapes. In *smi* (2002), Published by the IEEE Computer Society, p. 245.

[58] Mortara, M., Patané, G., Spagnuolo, M., Falcidieno, B., and Rossignac, J. Blowing bubbles for multi-scale analysis and decomposition of triangle meshes. *Algorithmica 38*, 1 (2003), 227–248.

[59] Munkres, J. *Elements of algebraic topology*, vol. 2. Addison-Wesley Reading, MA, 1984.

[60] Nooruddin, F., and Turk, G. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics* (2003), 191–205.

[61] Podolak, J., Shilane, P., Golovinskiy, A., Rusinkiewicz, S., and Funkhouser, T. A planar-reflective symmetry transform for 3D shapes. *ACM Transactions on Graphics (TOG) 25*, 3 (2006), 549–559.

[62] Rand, W. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association 66*, 336 (1971), 846–850.

[63] Reeb, G. Sur les points singuliers dune forme de pfaff completement intégrable ou dune fonction numérique. *Comptes Rendus de LAcadémie ses Séances, Paris 222* (1946), 847–849.

[64] Rusinkiewicz, S., and Levoy, M. Efficient variants of the ICP algorithm. In *Proceedings of 3DIM* (2001), pp. 145–152.

[65] Saleem, W., Wang, D., Belyaev, A., and Seidel, H. Automatic 2d shape orientation by example.

[66] Segal, A., Haehnel, D., and Thrun, S. Generalized-icp. In *Proc. of Robotics: Science and Systems (RSS)* (2009), Citeseer.

[67] Shamir, A. A formulation of boundary mesh segmentation. In *3DPVT 2004* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 82–89.

[68] Shamir, A. A survey on mesh segmentation techniques. In *Computer graphics forum* (2008), vol. 27, pp. 1539–1556.

[69] Shapira, L., Shamir, A., and Cohen-Or, D. Consistent mesh partitioning and skeletonisation using the shape diameter function. *The Visual Computer 24*, 4 (2008), 249–259.

[70] Shilane, P., Min, P., Kazhdan, M., and Funkhouser, T. The princeton shape benchmark. In *Shape modeling international* (2004), vol. 105, Citeseer, p. 179.

[71] Shlafman, S., Tal, A., and Katz, S. Metamorphosis of polyhedral surfaces using decomposition. *Eurographics 2002* (2002), 219–228.

[72] Siddiqi, K., and Pizer, S. *Medial representations: mathematics, algorithms and applications*, vol. 37. Springer Verlag, 2008.

[73] Siddiqi, K., Zhang, J., Macrini, D., Shokoufandeh, A., Bouix, S., and Dickinson, S. Retrieving articulated 3-D models using medial surfaces. *Machine Vision and Applications 19*, 4 (2008), 261–275.

[74] Sorkine, O., and Cohen-Or, D. Least-squares meshes. In *SMI '04: Proceedings of the Shape Modeling International 2004* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 191–199.

[75] Sorkine, O., Irony, D., and Toledo, S. Geometry-aware bases for shape approximation. *IEEE Transactions on Visualization and Computer Graphics 11*, 2 (2005), 171–180. Member-Cohen-Or, Daniel.

[76] Sundar, H., Silver, D., Gagvani, N., and Dickinson, S. Skeleton based shape matching and retrieval. In *Shape Modeling International* (2003), vol. 130, Citeseer.

[77] Svensson, S., Nystr
"om, I., and Sanniti di Baja, G. Curve skeletonization of surface-like objects in 3D images guided by voxel classification. *Pattern Recognition Letters 23*, 12 (2002), 1419–1426.

[78] Tagliasacchi, A., Zhang, H., and Cohen-Or, D. Curve skeleton extraction from incomplete point cloud. In *ACM SIGGRAPH 2009 papers* (2009), ACM, pp. 1–9.

[79] Taubin, G. Geometric signal processing on polygonal meshes. *Eurographics (State of The Art Report)* (2000).

[80] Tierny, J., Vandeborre, J.-P., and Daoudi, M. 3D mesh skeleton extraction using topological and geometrical analyses. *Pacific Graphics 2006* (2006), 85–94.

[81] Tierny, J., Vandeborre, J.-P., and Daoudi, M. Topology driven 3D mesh hierarchical segmentation. In *SMI '07: IEEE International Conference on Shape Modeling and Applications* (Washington, DC, USA, 2007), pp. 215–220.

[82] Vranic, D., Saupe, D., and Richter, J. Tools for 3d-object retrieval: Karhunen-loeve transform and spherical harmonics. In *Multimedia Signal Processing, 2001 IEEE Fourth Workshop on* (2001), IEEE, pp. 293–298.

[83] Wolfson, H., and Rigoutsos, I. Geometric hashing: An overview. *IEEE Computational Science & Engineering 4*, 4 (1997), 10–21.

[84] Xu, K., Stewart, J., and Fiume, E. Constraint-based automatic placement for scene composition. In *Graphics Interface* (2002), pp. 25–34.

[85] Yamauchi, H., Saleem, W., Yoshizawa, S., Karni, Z., Belyaev, A., and Seidel, H. Towards stable and salient multi-view representation of 3d shapes. In *Shape Modeling and Applications, 2006. SMI 2006. IEEE International Conference on* (2006), IEEE, pp. 40–40.

[86] Zaharia, T., and Prêteux, F. 3d versus 2d/3d shape descriptors: A comparative study. In *SPIE Conf. on Image Processing: Algorithms and Systems*, vol. 2004, Citeseer.

[87] Zheng, Q., Sharf, A., Tagliasacchi, A., Chen, B., Zhang, H., Sheffer, A., and Cohen-Or, D. Consensus skeleton for non-rigid space-time registration. In *Computer Graphics Forum* (2010), vol. 29, Wiley Online Library, pp. 635–644.

APPENDIX A

ALGORITHMS TESTED BY CHEN *et al.*

Chen *et al.* used these four metrics to evaluate and compare the results of seven different automatic and semi-automatic segmentation algorithms against their human-generated segmentations. Here we will give a short description of these algorithms.

- **K-Means**: Shlafman *et al.* [71] demonstrated how a K-means clustering of faces can be used to segment a mesh. Given some user-specified k number of segments, the algorithm iteratively refines clusters of faces based on central "seed" faces. It continues until the cluster assignment converges.

- **Random walks**: Lai *et al.* [47] provide a two-phase iterative segmentation process. First, each face is assigned to the segment most likely to reach it via a random walk on the dual graph of the mesh. Second, segments are merged according to the relative lengths of the intersections of perimeters of adjacent segments. This terminates when a user-specified number of segments is reached.

- **Fitting Primitives**: Attene *et al.* [8] use salient features for fitting primitives (planes, cylinders and spheres) that approximate mesh parts. Starting with each face in its own segment, the algorithm builds bottom-up, combining adjacent segments until some user-specified number of segments is reached.

- **Normalized cuts**: Golovinsky *et al.* [37] hierarchically segment the mesh in a bottom-up fashion. First, each segment is placed in its own segment. These segments are then merged according to the area-normalized cut cost (the sum of each segment's perimeter divided by its area) until a user-defined number of segments is reached.

- **Randomized cuts**: Golovinsky *et al.* [37] also propose a hierarchical segmentation method that starts with a decimated version of the mesh in a single segment and uses randomized minimum cuts to recursively split the mesh into binary segments.

- **Core extraction**: Katz *et al.* [43] decompose the mesh by first extracting its core and a set of appendages defined by feature points. Segmentation continues until segments no longer contain any feature points or some threshold proportion of mesh vertices are contained on the segment's convex hull.

- **Shape Diameter Function**: Shapira *et al.* [69] use the "Shape Diameter Function", which approximates the diameter of the object's volume in the neighborhood of a point on the surface, to apply energy minimization techniques to cluster faces in a way that maximizes boundary smoothness and location along concave seams.