The Symmetry Transform and its Applications

Joshua Podolak

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
By the Department of
Computer Science

Advisor: Szymon Rusinkiewicz

November 2008

Abstract

Recent improvements in methods for acquiring and generating 3D shape data over the last few decades have motivated the need for ever complex tools to analyze and edit such information. As a result, in the last few years, numerous methods have been introduced to improve the ways in which computers understand shape. These methods vary in the type of data they measure, ranging from from low-level, local geometry information to high-level semantic information.

Symmetry has long been known as an important cue for non-local human recognition of shape, and as such has been a key component in Vision and Graphics applications. A second important feature of symmetry is that it is prevalent in real-world models. Pipe and gear models contain rotational symmetries, buildings and square objects contain 4-point symmetry, and nearly every natural or man-made object will contain some reflectional symmetry. This prevalence is important because this means that applications relying on symmetry information will work on a large range of models.

In this thesis, we introduce the notion of a *Symmetry Transform*, a measure of "the amount of" or "the degree of" symmetry present in a 3D shape, under some class of transformations. This can be the set of all plane-reflectional symmetries, the set of all point reflections or some arbitrary set of transformations. We concentrate on point and reflection symmetry transforms, providing some theoretical reasoning for the transforms. We show efficient means for computing them, discuss storing them, and analyze some of their properties, including noise resistance, continuous variation under deformation, and stability with missing data.

While computing the transform of an entire model can be useful for a general understanding of shape, strong symmetries (such as the center of an ellipse or the main reflection planes of a rectangle), are easily identified by humans and represent an impor-

tant subset of the transform. In this thesis we provide a method for efficiently extracting these “principal symmetries”, and discuss some of their properties.

Finally, the symmetry transform provides mid-level information about the nature of a 3D object that can be utilized in a wide range of applications. We explore how symmetry may be used to improve alignment, matching, viewpoint selection, remeshing, and segmentation.

Acknowledgments

Psalm 3.14

My advisor is my Shepherd I shall not want.
He maketh me overturn green pastures, He leadeth me to question sill waters.
Yea, though I walk in the valley of the shadow of a paper submission, I will fear no reviewers for thou art with me; Thy rebuke and thy praise they comfort me.
Thou preparest a defense before me in the presence of my thesis committee, thou anointest my dissertation with suggestions, my references runneth over.
Surely publications and awards shall follow me all the days of my life and I will graduate from the university of my advisor forever.

I would like to thank my advisor, Szymon Rusinkiewicz, for his mentorship and continuing support. Besides contributing in many ways to the completion of this dissertation, he has also been a steady source of reassurance and inspiration to me. I always felt more confident about the universe after every conversation we had. I would also like to thank the other graphics professors, Thomas Funkhouser and Adam Finkelstein for their ever welcome feedback and guidance. Without their collaboration this dissertation would not be possible.

I have had an incredible time bouncing ideas off of people during my time at Princeton University, while I can't possibly begin to acknowledge every person I've had interactions with, I'd especially like to thank my office-mates Christopher DeCoro, and Michael Burns as well as the rest of the graphics lab: Benedict Brown, Paul Calamia, Forrester Cole,

Aleksey Golovinskiy, Misha Kazhdan, Diego Nehab, and Phil Shilane. Life at Princeton would have been much duller and a lot less fulfilling without you.

I would like to thank Princeton University for supporting my work. This work was also funded by Air Force Research Lab grant FA8650-04-1-1718 and NSF grants CCF-0347427 and IIS-0121446.

Of course, graduate school is more than “staring-at-a-computer-screen” research, and I’d like to single out Sam Cohen, Andy Plaks and Toby and Elaine Robison for creating a very welcome atmosphere for grad students a long way from home.

Finally I would like to thank my parents, my family and especially my wife Dahlia for putting up with long hours, frustrated moods and overlong descriptions of my work. None of this would have been possible without your support. Next time, things will be easier. I promise. ;-)

Contents

List of Figures

Chapter 1

Introduction

Accurate portrayal of 3D geometry has always been an important aspect of computer graphics, and so the creation of 3D content has long been an important requisite of the graphics community. Traditionally, this geometry was constructed by hand: either on a point by point basis, or using basic shapes such as cubes and cylinders to create progressively complex forms. More recently, a wide range of 3D scanning technologies have become common, enabling the easy acquisition of increasingly detailed representations of real-world objects. Finally, with the proliferation of 3D content on the Internet, an easy method of creating 3D composition is to find and combine existing models found online. This method, combined with the increasing number of 3D models available raises the demand for ever more complex 3D processing tools to help search for and edit existing models, with the goal of creating new content.

3D geometry analysis processing algorithms have long been known to the graphics, geometry and vision communities, and can be split into three major categories.

Local descriptors concentrate on local surface information. That is, algorithms have been developed to describe properties such as curvature, smoothness, or sharpness, all

qualities that can be described as functions of the *local* geometry. These properties are often used in tools that analyze or edit multiple points on a model, such as locating salient features or noise removal.

Global descriptors use a single set of numbers to describe an entire model. Classical examples of such numbers are the center of mass, the surface area and the volume of the model. Another type of global descriptor is a statistical analysis of local functions. Examples of such a descriptor would be the average smoothness of a model, or a histogram of the curvature. Global descriptors are useful in a range of applications, and are particularly noted for their use in shape matching.

Finally, mid-level descriptors are used to describe a portion of the model. A simple example of such a descriptor would be to segment a model into parts and run a global descriptor on each.

In our thesis we describe a method for measuring symmetry, a non-local descriptor, and show how symmetry information can be used to solve a wide range of problems.

1.1 Symmetry

Symmetry has long been known to provide visual cues for human vision [14], be an important developmental attribute in Evolutionary Biology [13] and a physical necessity in various manufactured objects. As such, over the last few decades there have been numerous methods proposed for detecting and measuring various forms of symmetry in 2D and 3D.

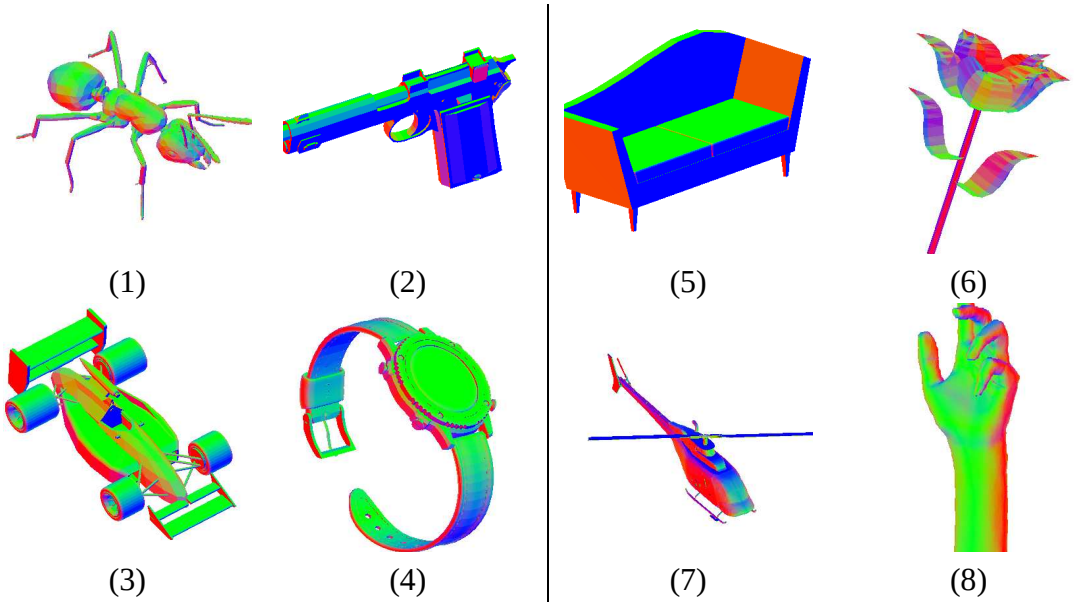


Figure 1.1: In this example we show a set of models classified using a simple left-right symmetry criterion. Models on the left are left-right symmetric, while models on the right are not.

1.1.1 Symmetry As a Descriptor

Traditionally, when a person claims that an object looks symmetric, they mean that certain areas of that object look similar to other areas. More accurately, we can say that an object contains symmetry if it appears unchanged after applying a transformation to it (e.g. rotating it by 90 degrees or reflecting it through some plane). Thus, by classifying objects based on their symmetric properties, we are using symmetry as a shape descriptor.

The simplest way to use symmetry as a descriptor is to chose a transformation (e.g. left-right reflection) and compute for every model whether it is symmetric or not. This would generate a very simple (and not particularly useful) global shape descriptor. We see an example of this in Figure 1.1, where objects are classified based on whether they have left-right symmetry or not.

1.1.2 Symmetry Measure

Rather than describing symmetry as a discrete property (a model can be 'symmetric' or 'not symmetric' with respect to some transformation), we can describe the degree of symmetry a model has. A fully symmetric object would have 100% symmetry, while non-symmetric objects would have 0% symmetry. Using *continuous symmetry* for shape description is more useful than discrete symmetry, as many natural objects contain strong symmetries, while not being perfectly symmetric. Moreover, even perfectly symmetric objects may result in non-symmetric scanned 3D models, because of noise or scanner miscalibration. A good example of a strong symmetry that is not perfect can be shown in Figure 1.1(7). While the helicopter model does not contain *perfect* left-right symmetry, the body of the helicopter is symmetric. Thus we can claim that the helicopter model has at least 90% left-right symmetry.

1.1.3 Symmetry Transform

Using the notion of continuous symmetry, we can define a *Symmetry Transform*, a function over an entire space of symmetries (e.g. all reflections) describing how symmetric the model is with respect to every symmetry in the space. This gives us a much richer description of the symmetries of the model. In the example shown in Figure 1.2 we see a few examples of a 2D reflection symmetry transform. Note that while the strongest symmetries are most noticeable, there are many other symmetries in the model that are captured in the transform. In all the 2D reflection symmetry examples shown, points are colored using the symmetry measure of the plane with the largest symmetry passing through that point, with darker lines representing stronger symmetries.

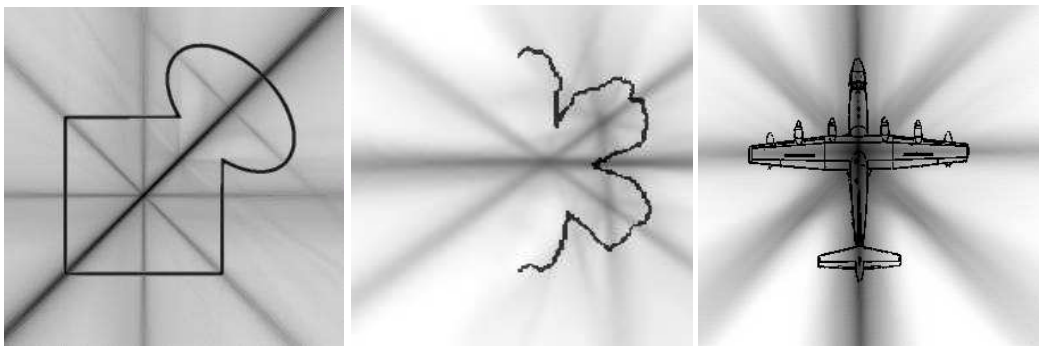


Figure 1.2: Examples of reflective symmetry transforms. Unless otherwise noted, points are colored by the symmetry measure of the plane with the largest symmetry passing through them, with darker lines representing stronger symmetries. Note how the symmetry of the flower remains strong despite being incomplete.

In this thesis we describe methods to efficiently compute symmetry transforms for 2D and 3D models and explore their properties. Additionally, we demonstrate methods to utilize the symmetry information in the transform to improve various geometry processing applications.

1.1.4 Principal Symmetries

Since the symmetry transform is a function that measures a class of symmetry over an entire object, it can be very informative to analyze the local maxima of this function. These maxima form a subset of strong symmetries that correspond to various perfect, local and partial symmetries of the the whole object. Due to their nature, these *Principal Symmetries* are important for many applications. In the example shown in Figure 1.3 we show a 2D shape with its principal symmetries highlighted. Note that how both perfect and non-perfect symmetries are found.

In this thesis we describe a method to find interesting local maxima in the transform and compute their exact position.

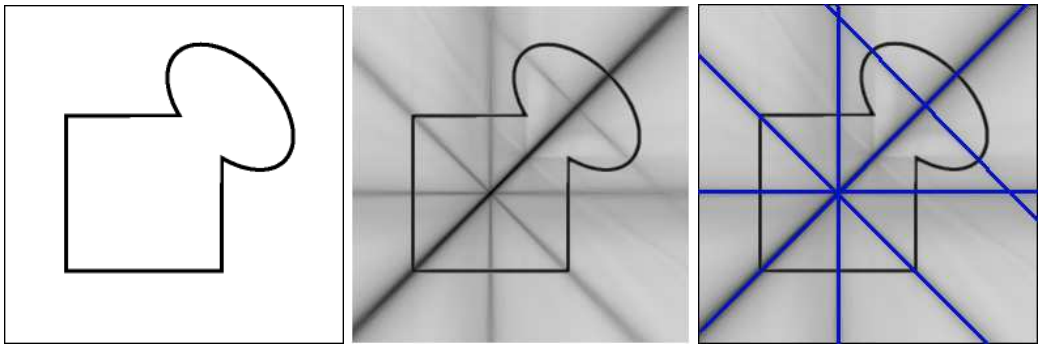


Figure 1.3: Example of principal symmetries. Starting with a 2D shape (left), we compute a reflective symmetry transform (center), and mark the local maxima (right). Note that the perfect symmetry is found as well as a number of local symmetries. Principal symmetries are marked in blue.

1.2 3D Analysis Applications

Whenever we try to fit two puzzle pieces together, we are performing shape analysis on those pieces. Any investigation of a 3D model, such as identifying the purpose of a piece of furniture or finding similarly shaped proteins, requires some understanding of the shape of that object. Shape analysis tools endeavor to supply that understanding by quantifying various properties of the object analyzed.

In this work we will show how to use symmetry information extracted from the symmetry transform to improve a few analysis applications

1.2.1 Alignment

Perhaps the most fundamental analysis problem in computer graphics is to find the correct orientation of a model. While humans have no difficulty in positioning objects (e.g. on shelf), computers have no obvious way of telling which way is “up”. One simple method of approaching the problem is to note that many objects have strong left-right symmetry. Thus, finding the largest symmetry of the model and using it as the left-right axis is a

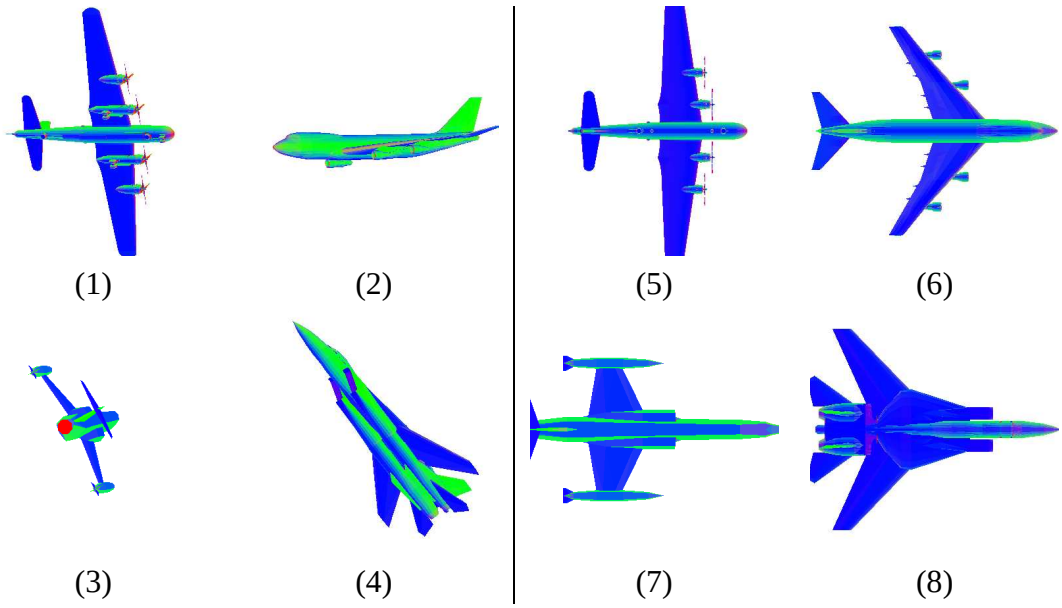


Figure 1.4: In this example we demonstrate the problem of alignment. At left, the airplane models are not aligned. At left, we have aligned them using symmetry considerations. Note how all the airplanes have perfect left-right symmetry and relatively strong up-down symmetry.

good start toward solving the problem. An example of this method can be shown in Figure 1.4, where the airplane models are automatically aligned using their perfect left-right symmetry. In section 4.1 we discuss a more robust way to align models using symmetry.

1.2.2 Matching

A key component of retrieving 3D content from a database or repository is the ability to efficiently search for a desired model. To this end, many search methods have been proposed that enable querying for a desired 3D shape. While text-based queries may be available if the database is tagged or classified, a more general method of searching uses a 3D input model as a query, and attempts to retrieve similar shapes from the database. An example of such a query can be found in Figure 1.5. The user inputs a car model as

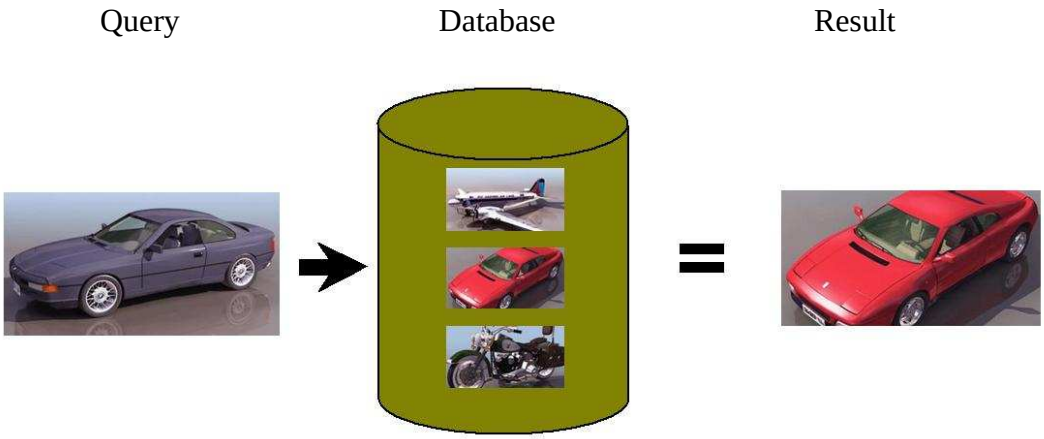


Figure 1.5: In this figure we show an example of shape matching. At left, the user inputs a 3D model of a car as a query. The system then searches through a database of 3D models for a similar shape (center), yielding another car model (right).

a query (left), the system then compares the model to all models in the database (center). Finally, the system returns the model that is most similar (right). In Section 4.2 we describe a method that uses symmetry information to improve matching.

1.2.3 Viewpoint Selection

Finally, an interesting analysis problem is to find the best direction for displaying a 3D model. Suppose that you wanted to make a catalog of 100 objects. From which direction would you take the photograph for each of those objects? In Section 4.3 we show how to use symmetry information to find the best direction automatically.

1.3 3D Editing Applications

Shape analysis leaves the object untouched, but many applications require that editing operations be preformed on the object, such as smoothing, slicing, or distorting. The importance of symmetry to the utility of an object (e.g. airplanes need to be left-right symmetric) means that we frequently wish to retain the symmetry characteristics of the object through the editing process, and so the addition of symmetry information to the editing algorithm will automatically improve the results of the operation.

In this work we show how to use symmetry information to improve two editing applications: Remeshing and Segmentation.

1.3.1 Remeshing

A typical problem in 3D modeling is that the data-sets can get very large. This is a big concern of applications that require rapid computation on 3D models, such as interactive rendering, editing, or physical simulation. One solution often employed by such applications is to first *remesh* those models, simplifying them while closely approximating their geometry, curvature or other properties. An example of such remeshing can be found in Figure 1.6, where the model of the hippo is remeshed to have fewer polygons while still preserving the features of the original shape. Since symmetry can be an important feature that we would like to preserve, we show a method for explicitly doing so in Section 5.1

1.3.2 Segmentation

Any serious editing tool must have some way to let a user split an object into parts, even if it's only by allowing the removal of individual triangles. Smart segmentation algorithms

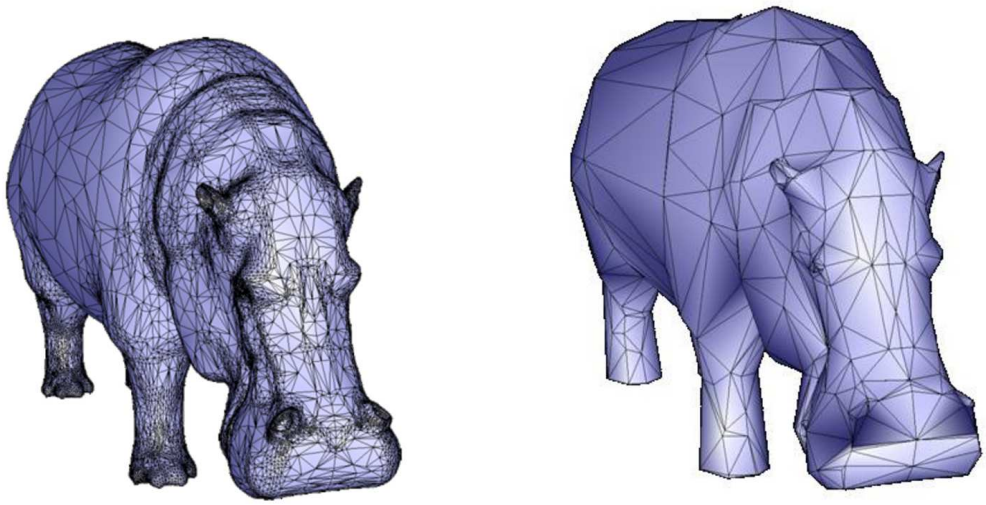


Figure 1.6: In this figure we show an example of remeshing. At left we show the original model of a hippo. At right we show the simplified model. Note how simplified the model retains the left-right symmetry of the original.

automatically find optimal cuts, splitting the model into intuitive components. Section 5.2 demonstrates a method using symmetry to segment models automatically.

The remainder of the thesis will be organized as follows: We will begin by discussing previous and current work done with symmetry in chapter 2, we will describe the Symmetry Transform in detail in chapter 3, and discuss methods for extracting useful information from it. In chapters 4 and 5 we will describe improvements to a number of analysis and editing applications using the symmetry transforms, and we will conclude in chapter 6 with some discussion and future work.

Chapter 2

Symmetry

Symmetry is an important feature of almost all shapes: nearly every man-made object contains at least one plane of perfect symmetry, and most natural objects exhibit near-perfect symmetries (look around your office and count the symmetric objects). This prevalence combined with the importance of symmetry to the human visual system [14], has made it an important avenue of research. As such, over the last few decades there have been numerous methods proposed for detecting and measuring various forms of symmetry in 2D and 3D.

2.1 Perfect Symmetries

The simplest form of symmetry is perfect symmetry or whole-object symmetry. For objects that contain perfect symmetry, applying the symmetry transformation causes one half of the model to be exactly mapped to the other half. For obvious reasons, perfect point symmetry can only happen about the center of mass, perfect rotational symmetry

can only occur about an axis going through the center and perfect reflectional symmetry can only happen with a plane passing through the center of mass.

Because of this constraint, perfect symmetries are the easiest to search for, and many efficient algorithms have been described for finding whole-object symmetry.

The earliest works on symmetry detection [3, 50, 55] reduced the problem to the detection of symmetry in circular strings and then used sub-string matching algorithms to detect the symmetries by exhaustively searching for instances of a string t within the concatenation of t with itself - tt. While this method is robust, searching through all possibilities can be prohibitive for large models. Rather than using an exhaustive search, [34] use an octree representation to efficiently search for possible candidates of the symmetry plane. Other methods include using the extended Gaussian image [46] and the singular value decomposition of the points of the model [42] in order to efficiently find symmetry planes.

More recently, Martinet et al. [33] have introduced a method based on generalized moments to detect perfect symmetry in 3D shapes accurately. Their approach combines the property that even order moments contain the same symmetries as the model with an efficient method for calculating moment coefficients using a spherical harmonic decomposition.

While perfect symmetry is an important feature of 3D models, it is limited in how well it can describe a model. In the first place, most models will contain only a single plane of perfect symmetry. Having two planes of symmetry usually means that the model is rotationally symmetric, and only very simple shapes like cubes have three or more perfect planes of symmetry that do not all intersect along the same axis. Furthermore, in order to have perfect symmetry there must be no noise in the model. Depending on the algorithm used, small amounts of noise or tessellation difference in the model can

prevent detection of otherwise perfect symmetry. Finally, a large class of models are not symmetric themselves (i.e. do not have perfect symmetry), but are composed of parts that are symmetric. For these types of models we require local symmetry.

2.2 Local Symmetry

Local symmetry occurs when an entire object is not symmetric, but contains some portion of it that is perfectly symmetric. A good example of model containing local symmetry is a motor. The motor itself is not symmetric, but it contains many parts such as axles, cylinders and gears that are all perfectly symmetric.

As a recent example, Thrun and Wegbreit [47] detect perfect symmetries in scanned models by explicitly searching ever-growing sets of points while maintaining a list of possible rotational and reflectional perfect symmetries. In this way, scans of models composed of symmetrical parts may be completed by extending the measured symmetries to the entire model

Further methods are available for describing local symmetries with a respect to a point — e.g., the medial axis [6]. However, since these methods consider only perfect symmetries, they are unstable with added noise or missing data and fail to recognize the potentially important cues of imperfect symmetries.

2.3 Approximate Symmetry

Rather than describing symmetry as a discrete property (a model can be 'symmetric' or 'not-symmetric'), In the last decade, methods have been provided for measuring imperfect symmetries. Given a 3D model it is possible to describe for a given transform the

degree of symmetry a model has. A fully symmetric object would have 100% symmetry, while non-symmetric objects would have 0% symmetry for that transform. For example, Zabrodsky et al. defined the *symmetry distance* of a shape with respect to a transformation as the distance from the given shape to the closest shape that is perfectly symmetric with respect to that transformation [54, 53]. They provide an algorithm to find the symmetry distance for a set of connected points for any given reflective or rotational transformation. Their method however, considers symmetry with respect to only one point or plane at a time, and thus not efficient for general cases where approximate symmetry is required for a large set of transformations.

Mitra et al.[35] search for approximate reflectional symmetry in 3D polygonal models by sampling pairs of points on the model. Since for each pair there is exactly one plane that reflects one point onto the other, each point pair constitutes a "vote" for that plane. They then cluster these votes to find the strongest planes of symmetry and show how to use this information for segmentation and compression.

2.4 Symmetry Measure

Kazhdan et al. used the same continuous measure of imperfect symmetries to define a *shape descriptor* that represents the symmetries of an object with respect to all planes and rotations through its center of mass [24, 26]. Their method begins by defining the *symmetry distance* of a vector v with respect to symmetry transform γ as the distance to the nearest vector u that is symmetric with respect to γ.

$$SD(v,\gamma) = \min_{u|\gamma(u)=u} \|v-u\|.$$

Using the fact that vectors symmetric with respect to γ are a subspace, they show that the nearest symmetric vector is the projection of v onto γ. Since the space of reflectional symmetries is orthogonal, this means that the projection of v onto γ is the average of v and $\gamma(v)$. Thus

$$SD(v,\gamma) = \left\| v - \frac{v+\gamma(v)}{2} \right\| = \frac{\|v-\gamma(v)\|}{2}.$$

Since we want the symmetry distance to be no greater than 1, this distance is normalized.

$$SD(v,\gamma) = \frac{\|v-\gamma(v)\|}{\|v\|}.$$

Finally, in order to obtain the symmetry measure, they complement the distance.

$$SM(v,\gamma) = 1 - SD(v,\gamma) =$$

$$\frac{SD^2(v,\gamma)}{\|v\|^2} = 1 - \frac{\|v-\gamma(v)\|^2}{4\|v\|^2} =$$

$$1 - \frac{\|v\|^2 - 2v\cdot\gamma(v) + \|\gamma(v)\|^2}{4\|v\|^2}.$$

Obtaining a function that is 1 when v is itself symmetric with respect to γ and 0 when v is anti-symmetric.

In the next section we will show how we use this formulation a the basis for computing transforms for capturing the symmetry around every point (PST) and every plane (PRST) in a 3D model.

Chapter 3

Symmetry Transform

Having precise definitions for continuous symmetry allows us to describe a specific partial or approximate symmetry in a model. However, most models contain multiple symmetries, and many applications often require a holistic understanding of a model, encompassing most or all of those symmetries to work well. In order to capture an entire such range of symmetries in a model, we define the notion of a symmetry *transform*, a function that measures a class of symmetry over an entire model.

In the following sections, we explore transforms for two symmetries: The *Point Symmetry Transform* and the *Planar Reflective Symmetry Transform*. Portions of this work were performed in collaboration with Philip Shilane, Aleksey Golovinskiy, Szymon Rusinkiewicz and Thomas Funkhouser, and were previously published as [39].

3.1 Point Symmetry Transform (PST)

In this section we describe the Point Symmetry Transform, show how to compute it efficiently, and discuss it's possibilities and limitations.

3.1.1 Defining the PST

The Point Symmetry Transform is a mapping from a scalar-valued function f defined over a d-dimensional space of points to another scalar-valued function PST over the same space, such that the scalar value associated with each point is a measure of f's symmetry with respect to that point.

Following the work mentioned in the previous chapter by [54, 24, 26] on symmetry, assuming we have rasterized our surface to a volumetric function f, we define the Point Symmetry Distance $PSD(f,p)$ of f with respect to a point p as the L_2 distance between f and the nearest function that is invariant to the reflection through that point:

$$PSD(f,p) = \min_{g|P(g)=g} \|f-g\|.$$

Where P is the reflection through the point p.

Since the symmetry distance lies between 0 and $\|f\|$ and provides a measure of the "anti-symmetry" of the shape with respect to p, we complement it and divide by the magnitude of f, to produce a normalized symmetry measure for our PST, such that

$$PST^2(f,p) = 1 - \frac{PSD^2(f,p)}{\|f\|^2}$$

As before, the nearest symmetric function to f is simply the average of f and $p(f)$

$$PSD(f,P) = \left\| f - \frac{f+P(f)}{2} \right\| = \frac{\|f - \|(f)\|}{2}.$$

Combining the two equations, yields

$$PST^2(f,p) = 1 - \frac{PSD^2(f,p)}{\|f\|^2} = 1 - \frac{\|f-P(f)\|^2}{4\|f\|^2} =$$

$$1 - \frac{\|f\|^2 - 2f \cdot P(f) + \|P(f)\|^2}{4\|f\|^2}.$$

If f is normalized, then $\|P(f)\| = \|f\| = 1$ (since norms are preserved by reflection), and we obtain

$$PST^2(f,p) = 1 - \frac{1 - 2f \cdot P(f) + 1}{4} = \frac{1 + f \cdot P(f)}{2}.$$

Thus, we obtain the required result of $PST(f,p) = 1$ if f is perfectly symmetric with respect to p, 0 is f is perfectly anti-symmetric with respect to p, and an intermediate value for partial symmetries.

3.1.2 Computation

As we have shown in the previous subsection, the calculation of the symmetry measure for a single point reduces to the calculation of a dot product between f and its reflection:

$$D(f,p) = f \cdot P(f).$$

Since we have already rasterized the model to a volumetric grid, a simple algorithm for computing the PST would be for every point, sum the product of each pair of corresponding elements.

Thus, to compute the PST(f) we would do the following:

for each point p in f:

  for each point x:

    $x' \leftarrow P(x)$

    $PST^2(f,p)$ += $f(x) \cdot f(x')$

Note that for 2D this method takes $O(n^2)$ to compute the PST for each point on an *nxn* grid ($O(n^3)$ in 3D). To obtain the full Point Symmetry Transform, this must be computed for $O(n^2)$ points ($O(n^3)$ in 3D), yielding a total algorithmic complexity of $O(n^4)$ ($O(n^6)$ in 3D). Obviously, this is prohibitive for large sized grids.

Instead, we use the following method to efficiently compute the PST. Note that for point symmetry, $P(x) = 2 \times (p - x) + x = 2p - x$. We find that computing the PST for a given p through all x at once

$$PST(f, p) = \sum (2p - x)x$$

Thus computing the PST for all p at once may be given by

$$PST(f) = \sum_{p} (2p - x)x$$

Or a convolution of f with itself. This immediately suggests a more efficient method for computing the PST: Instead of computing the transform for every point separately, we use an FFT to convert f to the frequency domain and do a dot product there. Running an inverse FFT on the result yields the complete PST. The total computation for this method is $O(n^2 log(n))$ for 2D and $O(n^3 log(n))$ for 3D.

Computation time:

By computing the PST over the entire volume at once, we are able to obtain the PST of models efficiently. For example, for a 64^3 grid resolution, computing the PST takes a 4 seconds in Matlab on a 3 GHz processor

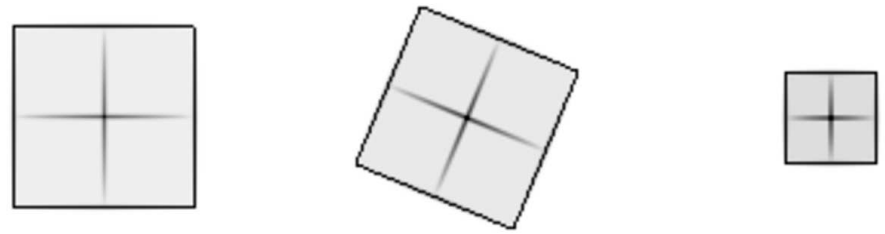
3.1.3 Properties of the PST

In this subsection, we present experimental results aimed at evaluating the stability of the PST in practice. We performed a set of experiments with simple shapes perturbed by several common transformations, such as rotation, scale, adding noise, adding a small feature, and deformation.

Figure 3.1 provides empirical evidence for the stability of the PST. Looking at the top row of images (Figure 3.1a), we see that it trivially translates, scales, and rotates with the input. While our algorithm could potentially introduce small sampling artifacts, since it rasterizes the input contour into a regular axis-aligned grid and stores the PST for a discrete set of points at regular angular intervals, we do not find this to be a significant problem in practice.

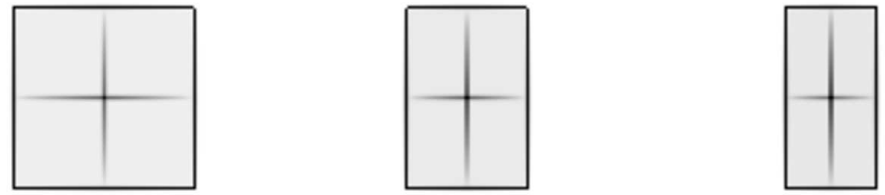
Looking at the second row of images, we took a shape and applied small perturbations to the boundary. Note this result is shown empirically for a simple 2D example in Figure 3.1b, but since we compute a GEDT on the surface, the contribution of any small change is minimized.

Finally, the bottom rows, we see how PST varies with large surface deformations. In (Figure 3.1c), the change to the PST is gradual when a square deforms into a rectangle. This continuity property is guaranteed by the fact that symmetry measure $PST(f,p)$ is defined as an integral over the entire function f for every point (Equation 3.1.1). Therefore, the impact of any perturbation to f on $PST(f,p)$ is limited by the magnitude of the perturbation. In (Figure 3.1d) we see a limitation of the PST in the sense that point symmetry measures only rotation of 180 degrees around a point. In the example shown, even though all the polygons have strong symmetry in their center, polygons that have an even number of faces contain strong symmetry along the axis, while polygons that do not



(a) Similarity transformation

(b) Small perturbations of the boundary



(c) Large deformations (changing aspect ratio)

(d) Increasingly fine tessellation

Figure 3.1: Visualization of the PST for several simple 2D shapes. These examples show the variation of the PST for several common boundary perturbations.

have an even number of faces will not contain symmetry along the axis, but instead have a gradual increase in symmetry as you get closer to the center.

The method used to visualize these PST is the same as shown in Figure 1.2 – the original 2D shape is shown in black, and each point of the PST is drawn shaded according to its symmetry measure. A dark colored point thus represents a near-perfect symmetry, while lighter, more fuzzy regions represent points with partial symmetry. In order to highlight the most significant symmetries, this visualization uses a nonlinear mapping of symmetry to intensity corresponding to a power law with an exponent of 4.

3.2 Planar Reflective Symmetry Transform (PRST)

Unlike the Point symmetry transform, which is a mapping from one function over a d-dimensional space to another function over the same space, the Planar Reflective Symmetry Transform is a mapping from a scalar-valued function f defined over a d-dimensional space of points to a scalar-valued function $PRST(f,\gamma)$ defined over the d-dimensional space of (hyper-)planes. This difference causes additional complexities in computing the PRST, which we will discuss in the following subsections.

3.2.1 Defining the PRST

Once again, we assume that we are given a 3D model that is rasterized to a volumetric function f. We define the Planar Reflective Symmetry Distance $PRSD(f,\gamma)$ of f with respect to a plane reflection γ as the L_2 distance between f and the nearest function that is invariant to that reflection:

$$PRSD(f,\gamma) = \min_{g|\gamma(g)=g} \|f-g\|.$$

Since the nearest symmetric function g to f is again the average of f and $\gamma(f)$, we can follow the same logic we used for the PST(Equation 3.1.1), obtaining a final result of:

$$PRST^2(f,\gamma) = \frac{1 + f \cdot \gamma(f)}{2}.$$

As with the PST, $PRST(f,\gamma) = 1$ if f is perfectly symmetric with respect to γ, 0 if f is perfectly anti-symmetric with respect to γ, and an intermediate value for partial symmetries.

3.2.2 Storage

Given a scalar function f over a d-dimensional space, our goal is to record symmetry values for all planes γ in the d-dimensional space of planes. To reach this goal, we need to discretize the space of planes. In general we want our parameterization to match the resolution of the input function; finer sampling will yield no additional information on f, which is band-limited. Moreover, we would like our parameterization to be as uniform as possible, in the space of planes.

To satisfy the above criteria, we discretize the space of planes by their normals and by their distance from the origin, which we chose to be the center of mass. Thus, when working in 2D, for an n-by-n grid, we use a uniform parameterization of the set of lines by their angles $\theta \in [0,\pi]$ and distance from the center of mass $r \in [-r_{max}, r_{max}]$ with n values in each dimension. We made the slightly unusual choice of angles on a semicircle with both positive and negative radii because using only positive radii would create a singularity at the origin. The origin is important because all perfect symmetries must pass through the origin and we wish to capture them accurately. Similarly, we parameterize planes in 3D by the spherical coordinates of their normals $\theta \in [0,\pi/2], \phi \in [0,2\pi]$ and

distance from the origin $r \in [-r_{max}, r_{max}]$, using n values in each dimension. While in 2D we could achieve a perfectly uniform discretization, in 3D the "buckets" of planes are not of uniform size, shrinking towards the poles as $\sin\theta$. We will take this into account when we show how to compute the transform in the next subsection.

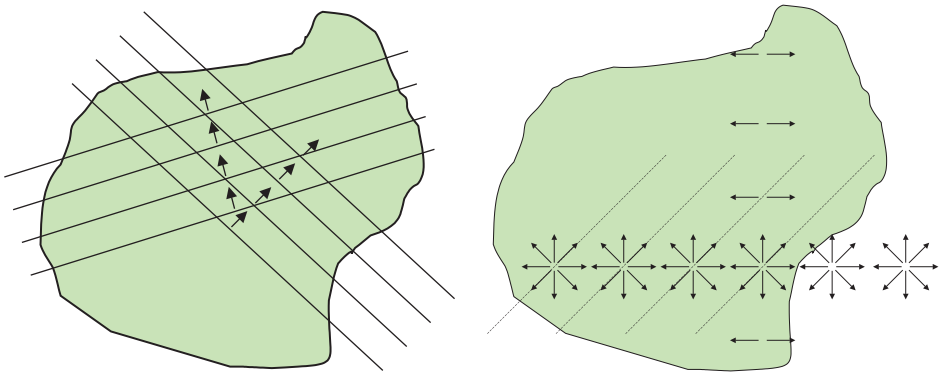
3.2.3 Computation

As we have shown above, computation of the symmetry measure for a single plane reduces to the computation of a dot product between f and its reflection:

$$D(f,\gamma) = f \cdot \gamma(f).$$

As with the PST, a naive algorithm for computing the transform would be to evaluate $PRST^2(f,\gamma)$ for every possible plane of reflection γ separately. Since there are $O(n^3)$ possible planes through a $n \times n \times n$ grid, and the evaluation of a dot product over the grid for each plane requires $O(n^3)$, the total complexity of this brute force algorithm is $O(n^6)$, which is prohibitively expensive for most applications.

Convolution Rather than evaluating the PRST for each plane separately, we note that PRST values for planes with the same orientation require dot products of functions successively shifted at regular intervals with respect to one another (Figure 3.2a), and so we can compute them all at the same time with a single convolution. Since in 3D, the convolution for a single direction takes $O(n^3 \log n)$, and there are $O(n^2)$ possible directions through the grid, the total running time of this algorithm is $O(n^5 \log n)$. Equivalently, we can consider convolutions over rotations at a discrete set of points (Figure 3.2b). In this case, we use the frequency domain algorithm described in [24] to compute the PRST for all planes through $O(n)$ points (Figure 3.2b). Since each invocation of Kazhdan et al.'s



(a) Sampling orientations (b) Sampling translations

Figure 3.2: Computing the PRST for many planes at once by: (a) sampling orientations and convolving over plane translations, or (b) sampling positions and convolving over plane rotations.

algorithm takes $(n^4 \log n)$, the total running time is again $O(n^5 \log n)$. A multiresolution approximation is possible in $O(n^4 \log n)$ ([26]).

Monte Carlo While the algorithms discussed so far are equally efficient for all functions, rasterized surfaces and point sets naturally lead to sparsity over the volume. In this section, we describe a Monte Carlo algorithm for computing the PRST that takes advantage of this sparsity to increase efficiency.

Our discussion of the algorithm begins with the brute-force approach presented above:

for each plane γ:

for each point x:

$x' \leftarrow \gamma(x)$

$PRST^2(f,\gamma)$ += $f(x) \cdot f(x')$

We observe that for sparse functions this is inefficient, since it performs useless computation whenever either $f(x)$ or $f(x')$ is near zero. Instead, we interchange the order of computations and perform *importance sampling* in a Monte Carlo framework:

for sampled points x:

  for sampled points x':

    $\gamma \leftarrow$ reflection plane(x, x')

    $PRST^2(f, \gamma)$ += $w(x, x', \gamma) \cdot f(x) \cdot f(x')$

Intuitively, this algorithm repeatedly picks a pair of points and "votes" for the plane between them. The sampling of x and x' is performed according to the energy in the function f, allowing us to focus effort on computations that will contribute to the final answer. For a typical 3D surface, non-negligible values appear in only $O(n^2)$ voxels, and thus this algorithm requires only $O(n^4)$ operations to compute the entire PRST.

Weighting: In order for the above algorithm to compute the PRST correctly, it is necessary to weight the contribution of each "vote" appropriately. This is the role of the function $w(x, x', \gamma)$, which consists of two terms. The first term accounts for the importance sampling that we perform, and is simply the reciprocal of the probability of having selected x and x':

$$w_{samp}(x, x', \gamma) \; = \; \frac{1}{f(x) \cdot f(x')}.$$

The second term represents a *change-of-variables*, accounting for the two different ways we have of sampling the space of planes: as a pair of points (x, x') and with our discretized bins over (r, θ, ϕ). While part of this change-of-variables term is intuitive (accounting for the $\sin\theta$ decrease in bin size), another part accounts for the fact that bins of planes will receive more votes if x and x' are far apart than if they are nearby. For example, both parts of Figure 3.3 consider the same single bucket of planes. However, for a fixed x, it is clear that more points x' will vote for that bin if the points are further apart, so we should weight the contribution of such pairs lower than votes by nearby points.

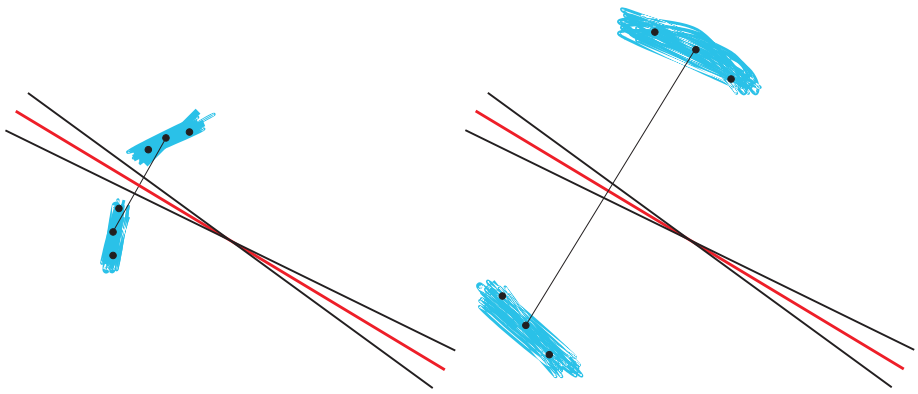
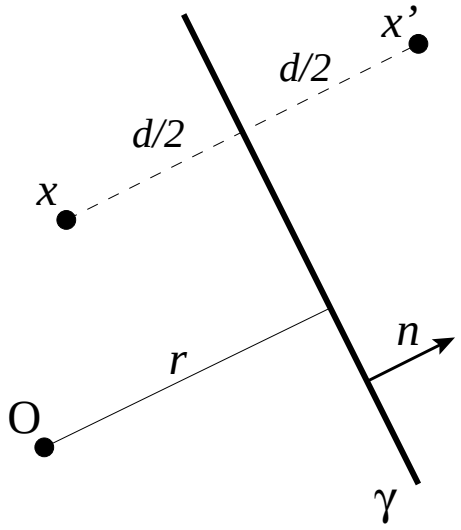


Figure 3.3: The efficient Monte Carlo algorithm selects a pair of points and votes for the plane between them. The vote must be weighted, accounting for the fact that as a point is father away from the plane of reflection, the chance of finding a reflection point is increased (the size of the blue area is larger when the points are farther apart).

To derive the change-of-variables weight, we simply compute the determinant of the Jacobian of the transformation between the parameterization of the planes of reflection and the reflected points themselves. If we let

$$\hat{n} = \begin{pmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta \end{pmatrix}$$

be the normal of the plane of reflection, then we can write



$$
\begin{aligned}
d &= \|x - x'\| \\
&= 2\,(r - \hat{n}\cdot x) \\
x' &= x + d\hat{n} \\
&= x + 2r\hat{n} - 2\,(\hat{n}\cdot x)\hat{n} \\
J &= \begin{pmatrix} \uparrow & \uparrow & \uparrow \\ (\frac{\partial x'}{\partial r}) & (\frac{\partial x'}{\partial \theta}) & (\frac{\partial x'}{\partial \phi}) \\ \downarrow & \downarrow & \downarrow \end{pmatrix}
\end{aligned}
$$

and solve for the determinant:

$$w_{\textit{change-of-variables}} = |J| = 2\,d^2 \sin\theta.$$

Therefore, we have

$$
\begin{aligned}
w(x, x', \gamma) &= w_{samp} \cdot w_{\textit{change-of-variables}}, \\
&= \frac{1}{f(x)\, f(x')\, 2\,d^2 \sin\theta}.
\end{aligned}
$$

So, overall, our Monte Carlo estimator is:

$$D(f,\gamma) = \frac{1}{N_{samp}} \sum_{i=1}^{N_{samp}} \frac{1}{2\,d^2 \sin\theta}$$

Analysis: Although the worst-case behavior of our algorithm is $O(n^6)$ if the input function f is represented on an $n \times n \times n$ grid, it is asymptotically faster for the sparse functions typical of 3D models. Good convergence can be achieved if the number of Monte Carlo samples is at least on the order of the number of non-negligible values in the grid. So, for

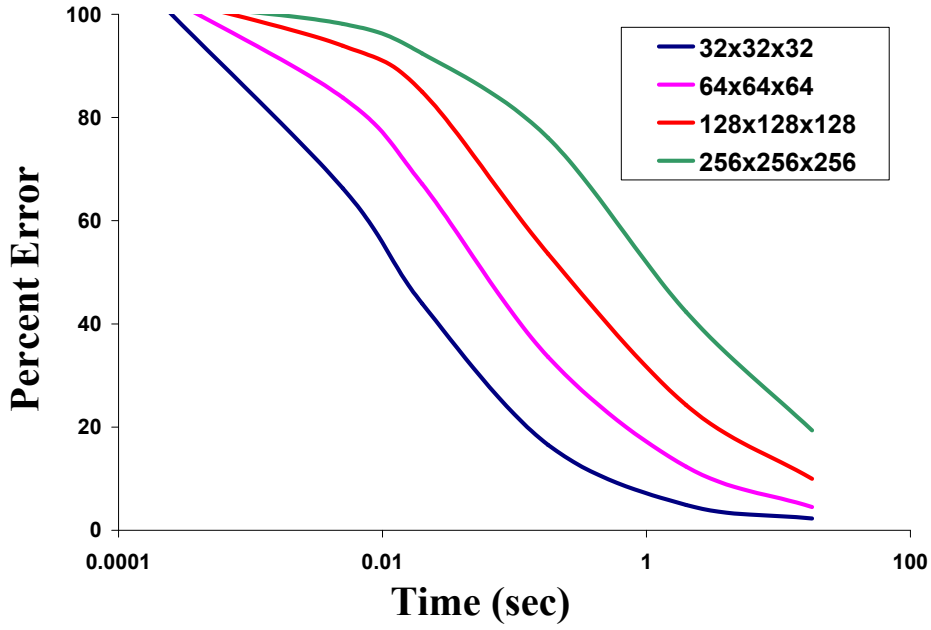


Figure 3.4: Comparison of error in the Monte Carlo approximation to the PRST, as a function of time. For typical grid sizes, such as the 64^3 used in the applications in following sections, computing the PRST takes only a few seconds.

a typical 3D surface that has non-negligible values in only $O(n^2)$ voxels, $O(n^4)$ operations (samples) are necessary to compute the PRST robustly, which significantly faster than the $O(n^6)$ brute force algorithm. Similarly, only $O(n^2)$ operations are required to compute the PRST for typical curves. These speedups make computing the PRST practical for all but the most complex 3D surfaces.

Computation time: By exploiting sparsity in the volume, the Monte Carlo algorithm is able to compute the PRST of 3D surfaces efficiently. As with all randomized algorithms, noise in the final approximation decreases with additional samples, but as shown in

Figure 3.4, the algorithm converges quite quickly. For example, for the 64^3 grid resolution used throughout this paper, computing the PRST to 1% noise takes an average of 8 seconds on a 3 GHz processor, corresponding to two million sampled point pairs. These results are typical - there is little variation in computation time, except for very large models (for which the rasterization time can begin to dominate). For 2D, computing the PRST for a 128^2 grid resolution takes less than a second in Matlab on a 3 GHz processor

3.2.4 Properties of the PRST

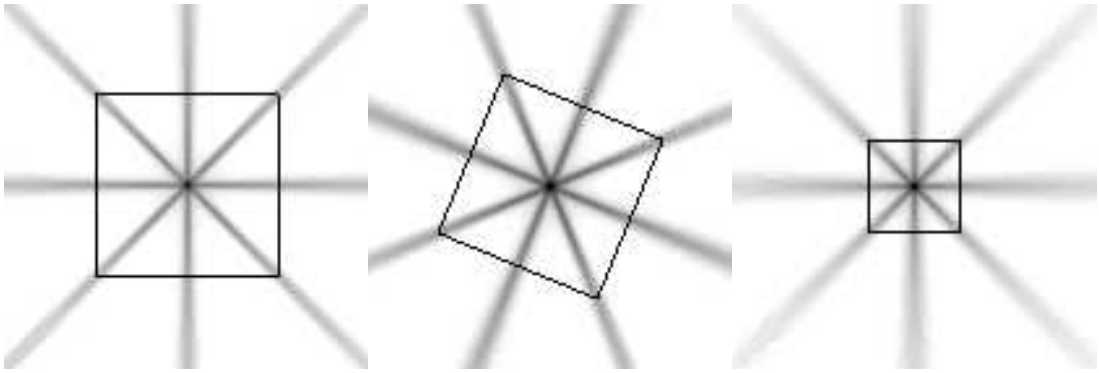
In this subsection, we present experimental results aimed at evaluating the stability of the PRST. We performed the same set of experiments as with the PST, perturbing simple shapes by common transformations.

As with the PST, the top row of Figure 3.5 provides empirical evidence for the stability of the PRST with respect to translation, scale, and rotation.

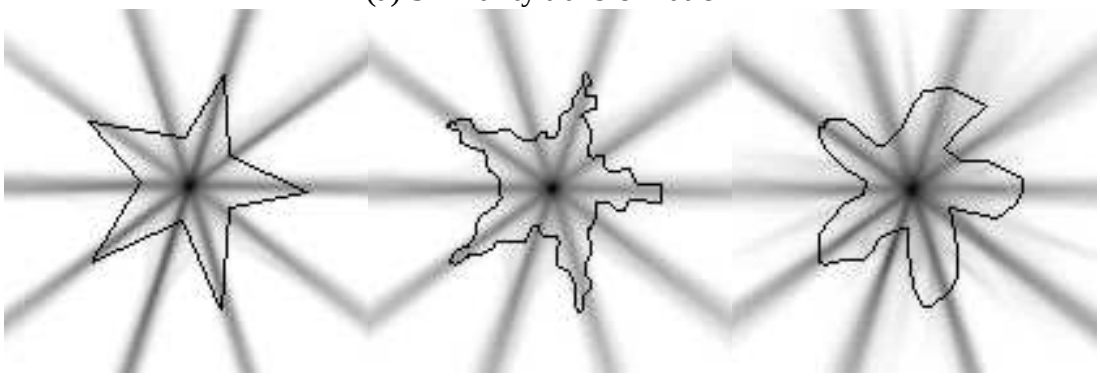
In the second and third rows of images, we took a simple shape and applied small perturbations to the boundary. As with the PST, the contribution of any small change is minimized by computing a GEDT on the surface after rasterization.

Finally, in the bottom row, we see an advantage of the PRST over the PST. In Figure 3.5(d), increasing the tessellation of the polygon causes the PRST to behave in the expected manner (i.e. increasing the number of symmetry planes), while in Figure 3.1(d), the lack of 180 symmetry in odd-numbered polygons causes the PST to fluctuate between identifying a strong symmetric center and and having uniform symmetry throughout the model.

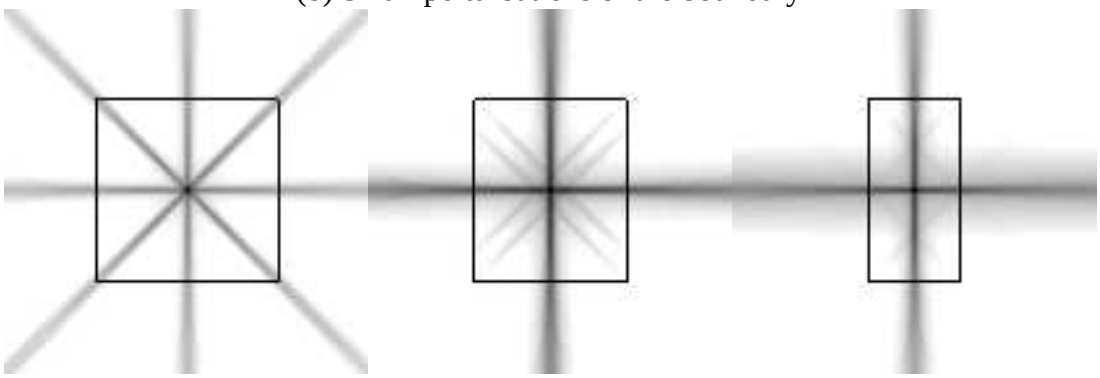
While the PRST is defined over the space of planes, in order to give an intuitive sense for the information provided by the PRST, and to compare to the results obtained for the PST, the visualizations for shown above are shown in the original 2D space. In



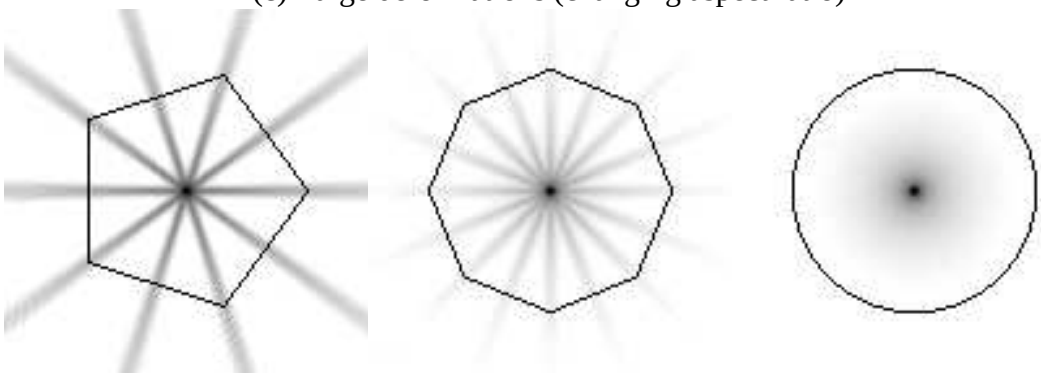
(a) Similarity transformation



(b) Small perturbations of the boundary



(c) Large deformations (changing aspect ratio)



(d) Increasingly fine tessellation

Figure 3.5: Visualization of the PRST for several simple 2D shapes. These examples show the variation of the PRST for several common boundary perturbations.

these visualizations, symmetry is measured for every plane, and the darkness of every point represents the maximum of PRST values over all planes passing through the point (darker values represent larger symmetries). As before, the original 2D shape is overlaid in black and we use the same nonlinear mapping of symmetry to intensity.

3.3 Principal Symmetries

While the PRST taken as a whole characterizes all of an object's symmetries, its *local maxima* form an important and intuitive subset. They may correspond to the principal symmetries of the whole object, weaker or partial local symmetries, or perfect symmetries of parts. Due to their intuitive nature, they are important for several applications, including several of those described in Chapterss 4 and 5. We now discuss an approach that builds upon the algorithm presented in the previous section to find local symmetry maxima precisely: we extract candidates for local maxima from the discrete PRST, then refine their locations using an iterative local optimization algorithm. This algorithm is able to find local maxima of the PRST with arbitrary precision.

Given the full 3D symmetry transform, tabulated at a moderate resolution, we first look for cells with a higher symmetry value than all their immediate neighbors. This yields a large number of candidates, so we apply a number of thresholds to extract only the strongest symmetries. First, we apply a threshold on the strength of the symmetry at that cell. While we could use a single fixed threshold, we have observed that portions of the model away from the center naturally have lower symmetry values (since there is less of the model that could potentially map onto itself under those reflections), so it is more natural to use a *lower* threshold near the edges of the model than near the center. In particular, we use a threshold proportional to $1 - {}^{r}/_{R}$, where R is the radius of the object

and r is the distance of the candidate plane from the center of mass. This corresponds to the symmetry score of a plane passing at a distance of r from the center of a sphere of uniform thickness with radius R. On top of the symmetry threshold, we also discard shallow maxima, which are potentially subject to noise: we impose a threshold on the discrete Laplacian (sum of second partial derivatives) of the PRST. The thresholds are set automatically to $^1/_{10}$ of the values at the strongest local symmetry.

Once we have a list of candidate local maxima, we refine them to find the planes of symmetry with high precision. This approach, of finding maxima of a function by first tabulating it then locally refining candidate maxima, is commonly used for numerical maximization in general, and also resembles the local optimization performed by Martinet et al. [33]. Our refinement method is inspired by the Iterative Closest Points algorithm [4], commonly used to perform pairwise alignment of meshes, but solves for a plane of reflection rather than a rigid-body transformation.

Our "Iterative Symmetric Points" or ISP algorithm begins by randomly sampling points from the mesh (we typically use around 10,000 points per iteration), then reflecting them across the candidate plane. We match each reflected point to the closest point on the mesh, then solve for the three parameters of the reflection plane that minimizes the sum of distances (weighted to account for the Gaussian Euclidean Distance Transform) between corresponding points (note that the minimal sum of weighted distances provides a maximum for Equation 3.2.1 when f is the GEDT of the surface). The process is iterated until it converges to a local maximum of the PRST. Figure 3.6, left, shows an iteration of ISP, with source points in red, the candidate plane in gray, and reflected points in green. The support of the final maximum is shown at right.

If the iteration causes the reflection plane to leave its cell (in the discrete PRST), the candidate is determined to be an unstable local maximum and discarded. Of course, this

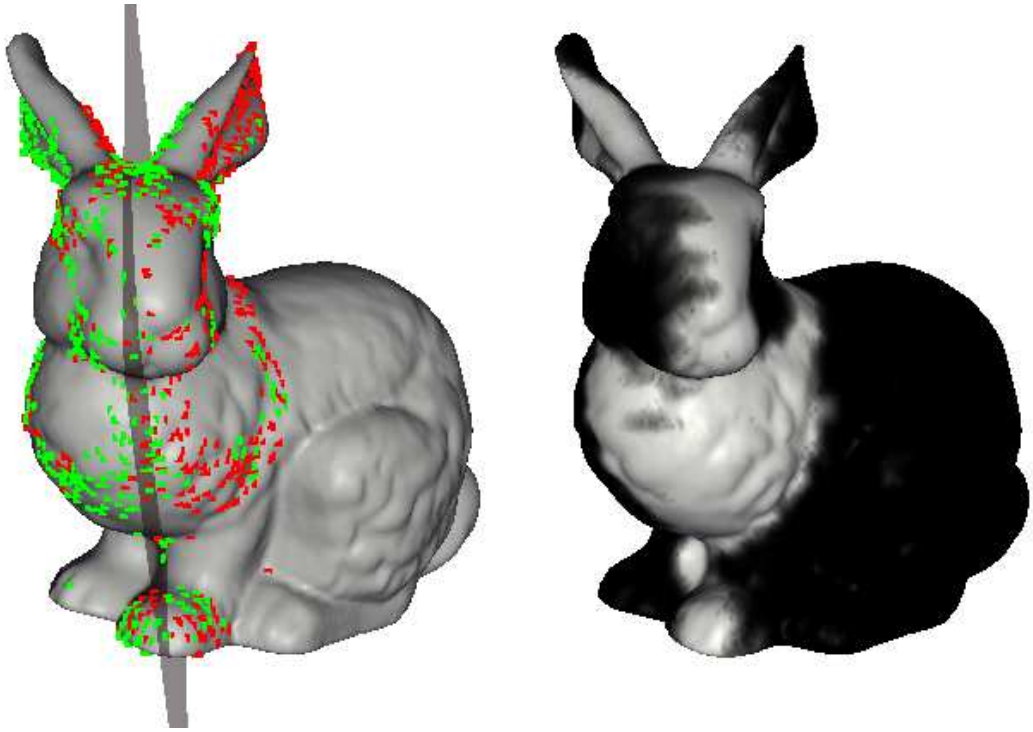


Figure 3.6: At left, we show an iteration of *ISP*. We select random points (red), reflect them through the candidate plane of symmetry (gray), and find closest points on the surface (green). We then update the plane of reflection to optimize the sum of Gaussian distances between corresponding point pairs (samples with low weight have been culled in this visualization). At right, we show the support of the final local symmetry maximum, as indicated by the gray level.

should not happen if the function f is sufficiently band-limited by the GEDT. However, we have found this check is necessary since different point sampling strategies are used by the discrete and iterative algorithms.

In our experiments, this two-stage process of first tabulating the PRST then refining candidate local maxima has proven both robust and efficient. The local refinement converges in a few seconds for each plane, and we typically find 10–20 strong local maxima of symmetry for models of moderate complexity. Figure 3.7 shows the four strongest local maxima for a bull model, together with the surface support of each plane reflection (white regions of the surface reflect onto each other across the chosen plane). Note

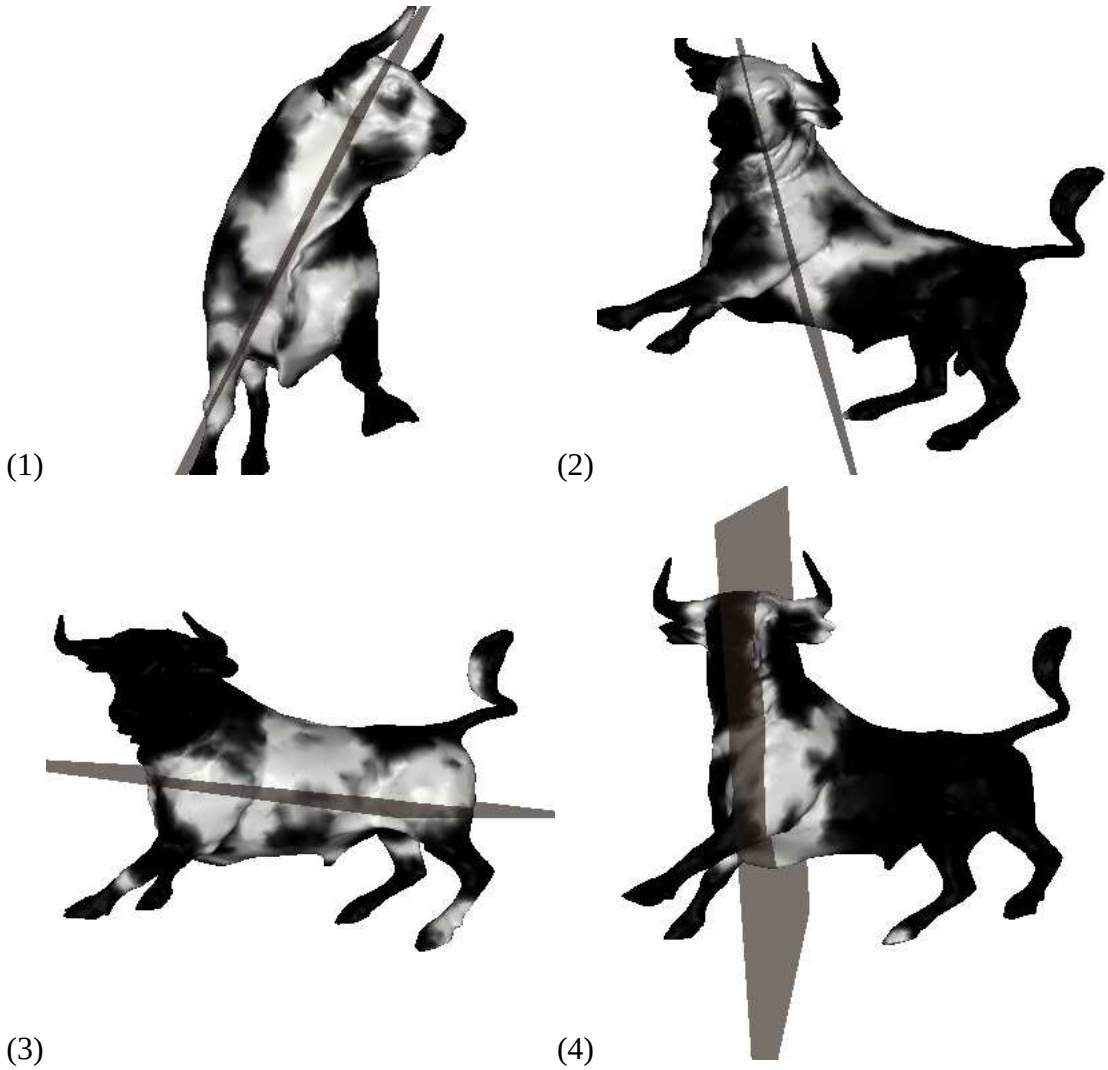


Figure 3.7: In this visualization, the triangles of the bull are colored to show how symmetric they are with respect to the plane of symmetry displayed, with black meaning no support of the plane reflection. Planes representing the four strongest local maxima of the PRST are shown here. Note how points supporting reflections across planes (2), (3), and (4) tend to cluster into regions corresponding to the neck, body, and head, respectively.

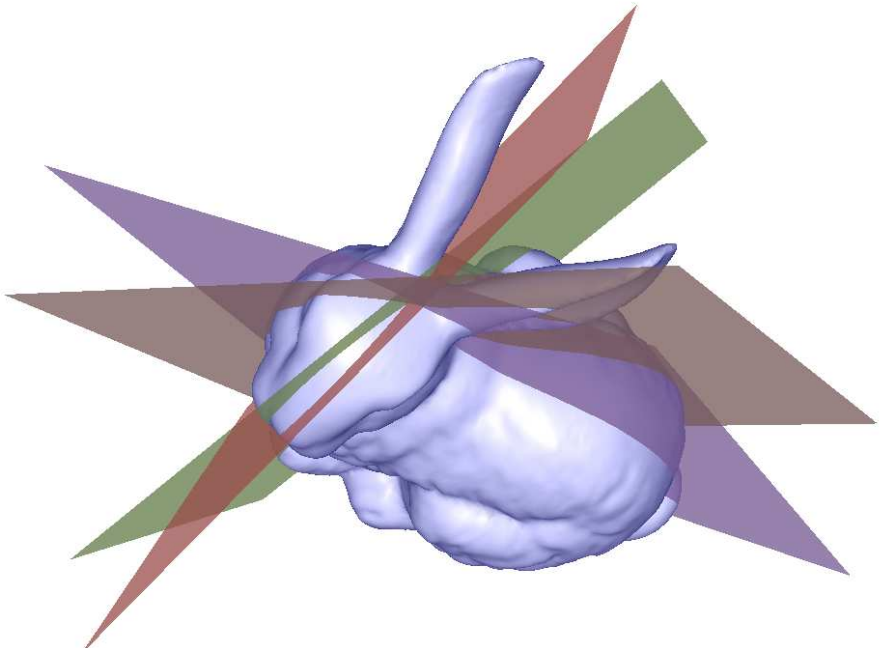


Figure 3.8: An example of principal symmetries of a 3D model, extracted from the 3D symmetry transform. Note that these capture the partial and approximate symmetries of the ears, head, body and legs of the bunny.

that we find planes capturing the global symmetries of the bull (1), as well as separate local maxima capturing symmetries of the neck (2), body (3), and head (4). Another interesting example of local maxima can be found in Figure 3.8. While it is intuitively obvious that the bunny has two major planes of symmetry (one for the head and one for the body), running the PRST on the model shows that there are in fact *four* strong symmetries, capturing the separate symmetries of the ears, body, head and legs of the bunny respectively.

Chapter 4

Analysis Applications

With the ever increasing number of 3D models available, the ability to sift through a large number of models and extract relevant information about them is of significant importance. In this chapter we will discuss the ways in which symmetry information can improve results for the applications of alignment, matching and viewpoint selection.

4.1 Alignment

Assume you have models of two chairs, and you wish to know which one is more top-heavy. A naive way to do this is to compare the centers of mass of the two chairs and see which one is higher. In a case in which the models were both generated using a similar scanning or modeling system this is probably correct. Unfortunately, in the general case, the models were generated independently, so there is no way of telling if the scale of the two chairs is the same, what the "up" direction is, and if it is consistent. Instead, we need a preprocess in which we align the scale, orientation and position of the two chairs. This general problem of alignment is an important step for a variety of tasks, including

visualization, studying the variation of models across different classes, composition of scenes, and indexing of 3D model databases.

In the example described above, one solution might be to align one chair to the other. This is known as *pairwise alignment*. A more general case is when we want to compare the characteristics of many models and pairwise alignment may be prohibitive. Instead, we automatically align all models to some canonical coordinate frame. In this section, we will concentrate on selecting a global point of reference for the origin of 3D models (*translation alignment*) and a set of axes to determine their orientation (*rotation alignment*). We do not address the scale alignment problem: we assume this is preformed using some other technique such as PCA [12].

4.1.1 Previous Work

Historically, canonical alignments are computed with principal component analysis (PCA): the center of mass is of the object chosen as the origin, and the principal axes are used to determine the orientation [12]. However, PCA does not always produce compatible alignments for objects in the same class. Specifically, PCA gives the direction of maximal variance in an object. Since variance is a function of the distance squared, the orientation produced by PCA tends to be overly influenced by small areas of the surface far from the center. As an example, consider the drawings of the mugs shown in Figures 4.1. Even though the mugs look identical to each other with the exception of the shape of their handles, the principal axes (shown in green in Figure 4.1) are widely different, producing inconsistent alignments undesirable for most applications. Note that the center of mass is similarly affected, though to a lesser extent.

A second weakness of using the center of mass + PCA is that it rarely produces alignments similar to what a human would select. Again, consider the mugs shown in

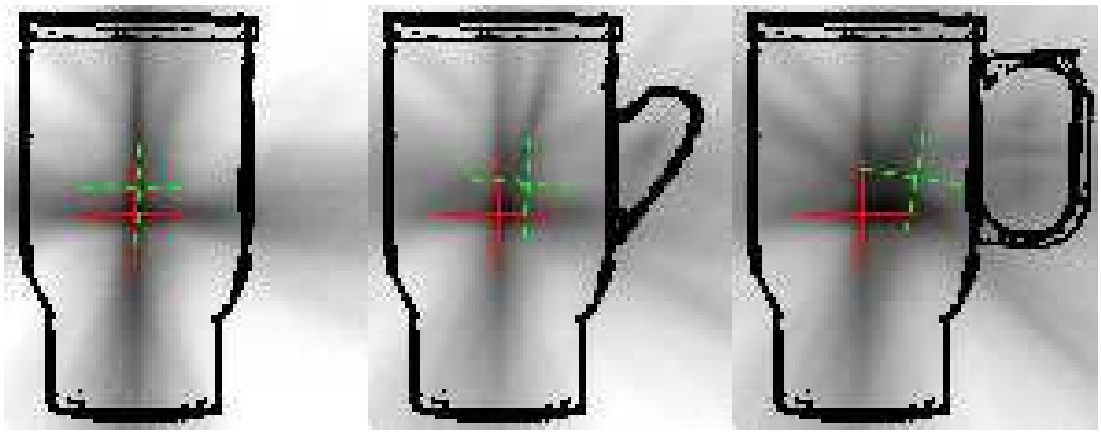
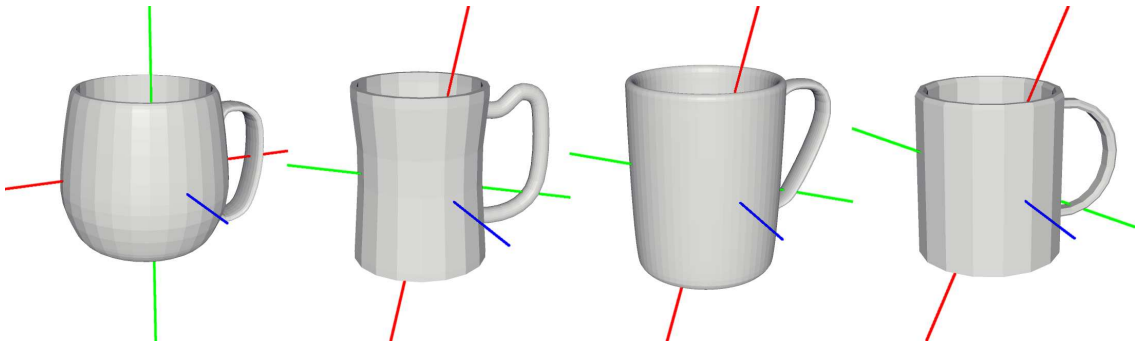


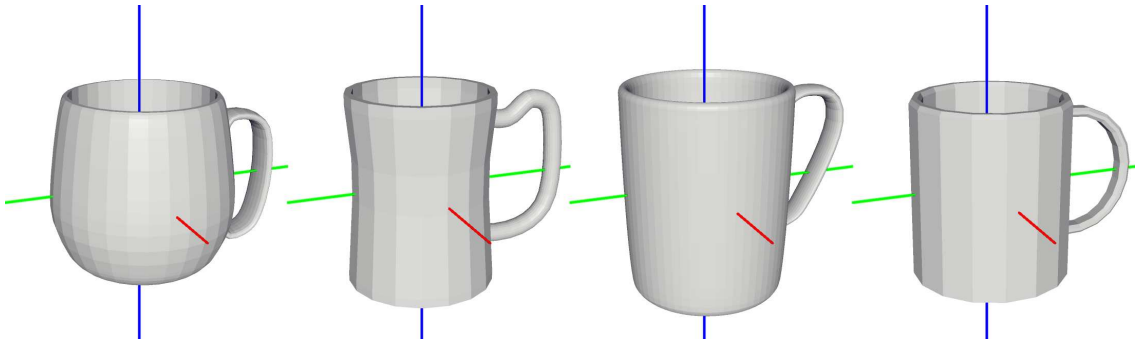
Figure 4.1: A line drawing of a mug with and without handles. The the center of mass and PCA axes are drawn in dotted green — note that they move depending on the presence of handles. A visualization of the PRST is overlaid on the drawings, and the center of symmetry and principal symmetry axes are shown in solid red — they remain stable under perturbation of the shape.

Figures 4.1 and 4.2. Most humans would suggest that the central axis of these mugs runs straight up and down through the middle of the cup, and the center is somewhere along this axis. However, in all cases shown, the center of mass and principal axes are biased towards the handle to different degrees.

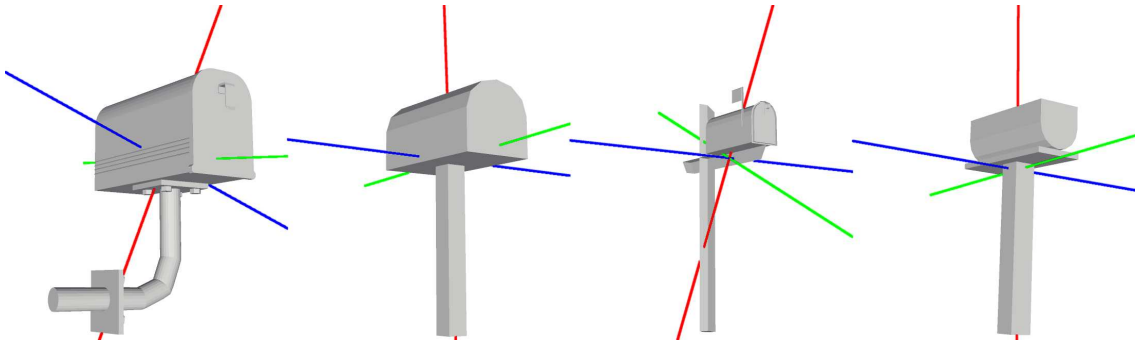
Using symmetry to align classes of models was first suggested by Kazhdan et al. [24, 25]. As mentioned in Section 2.4, they created a symmetry descriptor to measure the reflective symmetry of an object through all planes passing through the center of mass. They used the symmetries of the model for rotation alignment but still used the center of mass for translation alignment. While they got improved results in their experiment, a weakness of their approach is that they only capture symmetry passing through the center of mass. Since any perfect symmetry passing through the center of mass in the original model is provably captured by PCA as well, the improvement in the alignment of those models is not great. For models where the principal symmetries do not pass through the center of mass, the symmetry transform is needed.



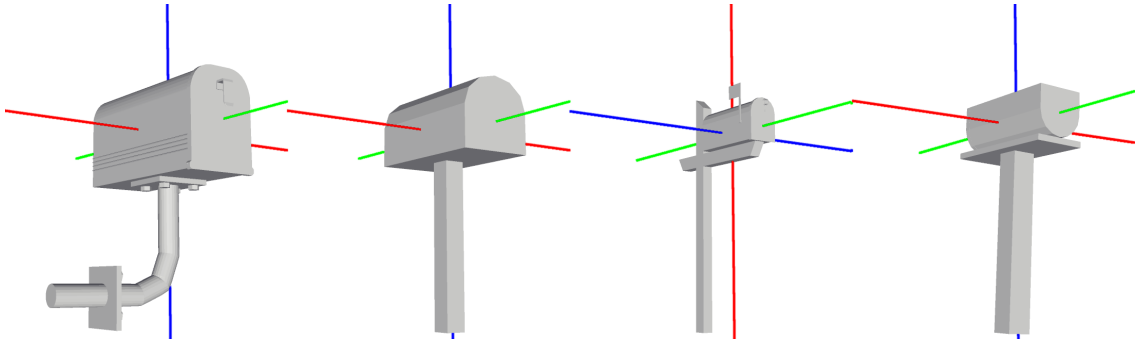
Center of mass and PCA



Center of symmetry and principal symmetry axes



Center of mass and PCA



Center of symmetry and principal symmetry axes

Figure 4.2: Alignment for translation and rotation based on centers of symmetry and principal symmetry axes, as compared to center of mass and PCA. In each case, the red, green, and blue lines represent the first, second, and third principal axes, and their intersection is the computed center.

More recently, Fu et al. [15] proposed a method for finding the upright orientation of man-made objects by searching through all possible bases for the object and applying learning techniques to quantify how "stable" each possible orientation is. Acknowledging the prevalence of left-right symmetry, especially in man-made objects, an explicit component of their metric is an approximation of the reflectional symmetry of the object with respect to planes perpendicular to the ground.

4.1.2 Symmetry Transform

Our intuition for using symmetry to align classes of models is based on the premise that the shape of an object is very dependent on the fact that we live in a physical world. Consider that a bird without left-right symmetry would not be able to fly well, or that building gears without rotational symmetry is not a simple task. Based on this insight, we hope to show that not only is symmetry a good candidate for aligning models consistently, but that the resulting alignment will be one that is also intuitive to humans.

We begin by introducing two new concepts, the *principal symmetry axes* (PSA) and the *center of symmetry* (COS), as robust global alignment features of a model. Intuitively, the center of symmetry is a point around which the symmetry is greatest, and the principal symmetry axes are the normals of an orthogonal set of planes with maximal symmetry passing through that point.

Specifically, given a PRST, we find the plane with maximal symmetry (the point in the function with the highest score), and choose its normal to be the first principal symmetry axis. We then select the second axis by searching for the plane with maximal symmetry among those perpendicular to the first. Since the third axis must be perpendicular to the first two axes, these two planes are enough to create a canonical rotation. In order to find

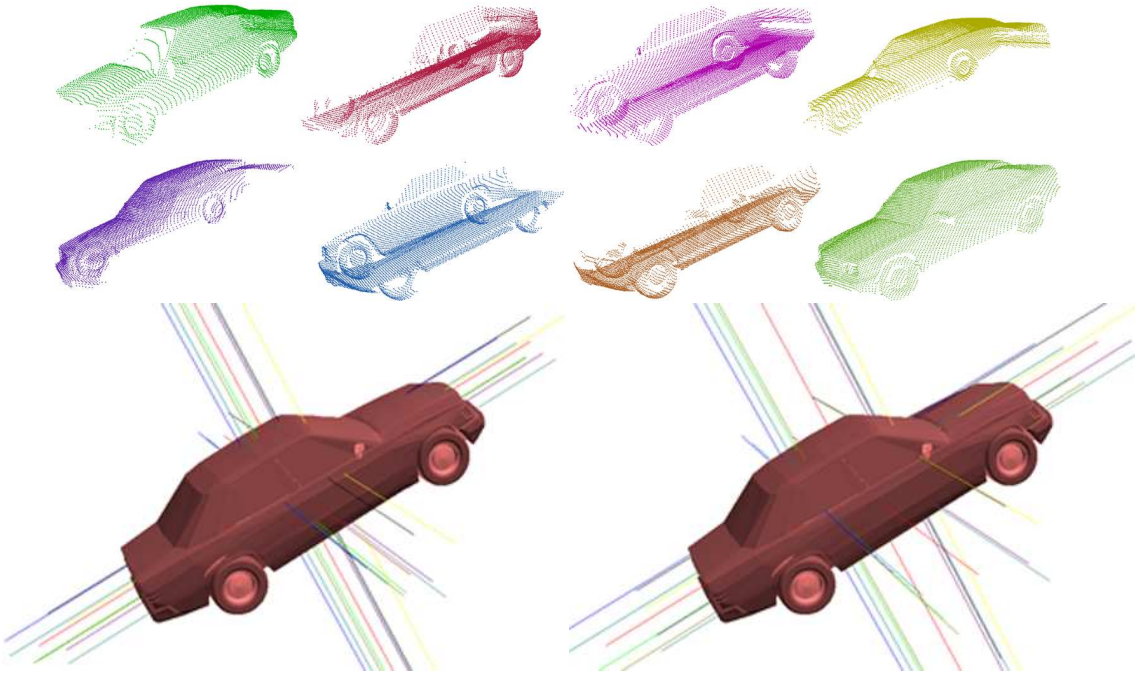
the center of symmetry, we find the plane with maximal symmetry that is perpendicular to the first two planes. The intersection of the three planes is our center of symmetry.

We find that this simple method produces coordinate frames that are indeed both robust and semantically meaningful for most objects. Referring to the examples mentioned above, for the mugs shown in Figures 4.1 and 4.2, the center of symmetry and principal symmetry axes appear right in the middle of the cylindrical cup in all cases. Similarly, for the mailboxes shown in Figure 4.2, the center of symmetry and principal symmetry axes consistently reside in the middle of the box — unlike the center of mass and principal axes, they are not affected by the shapes of the stands. In general, we find that the principal symmetry axes and center of symmetry are determined by an object's large parts with significant symmetries, and thus they closely match our intuition of an object's canonical coordinate frame.

4.1.3 Results

We ran a number of tests in order to measure the robustness of symmetry-based alignment. In the first test, we experimented with alignments of synthetically generated range scans of objects. This experiment is motivated by an object recognition application in which (partial-object) scans are acquired and registered to (whole-object) meshes stored in a database, with the hope of automatically recognizing which object was scanned [43]. For this application, it is useful to align the partial scan to the complete object automatically.

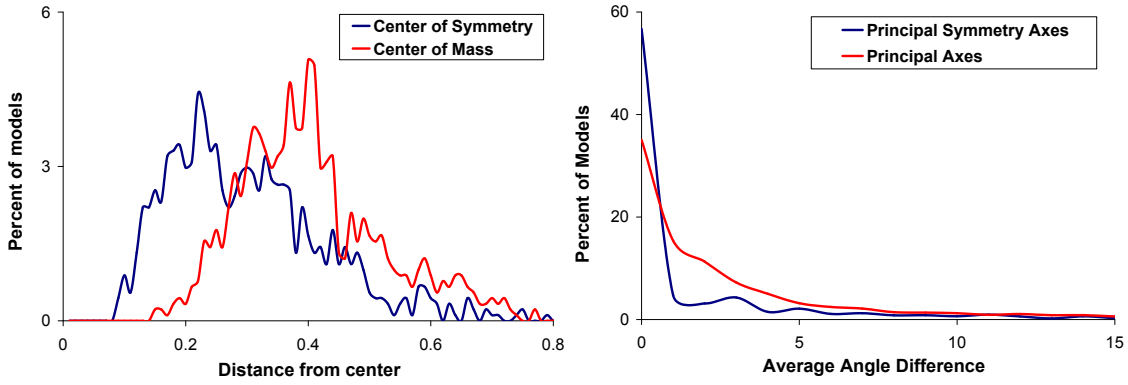
For the experiment, we used a ray tracer to generate eight synthetic range scans of approximately 10,000 points (Figure 4.3, top) for the 907 meshes provided as part of the Princeton Shape Benchmark test set [44]. For each mesh, the virtual scanner was in turn placed at each of the eight corners of a cube surrounding the model, always pointing



(Center of Symmetry) (Center of Mass)

Figure 4.3: To evaluate the stability offered by symmetry-based alignment in a scan recognition application, we computed eight virtual scans of 907 models (top two rows of images), and for each computed coordinate frames using our symmetry-based approach and PCA. Note how the centers of symmetry computed from the partial scans (shown as cross-hairs) cluster near the center of the whole car better than do the centers of mass.

toward the center of the mesh bounding box. The view distance was twice the length of the bounding box diagonal, and the field of view was 0.4 radians (Figure 4.3 top). For each scan, we voxelized the point cloud, computed the PRST on an 64x64x64 grid, and extracted the principal symmetry axes and center of symmetry (Figure 4.3, bottom left). Then, we evaluated how well the coordinate frames computed for each of the partial model matched the frames computed for the complete meshes. We then ran the exact same experiment, using the traditional principal axes + center of mass (Figure 4.3 , bottom right).



(a) (b)

Figure 4.4: Histograms of translational (a) and rotational (b) misalignment in our virtual-scan experiment. In blue we show alignment based on center of symmetry and PSA, with center of mass and PCA in green. Larger values near the left of each graph indicate better matching performance.

Figure 4.4 shows the results of this experiment. In the first plot, we see histograms of the translational misalignment between the partial range scans and the whole objects when aligned with the center of symmetry (blue curve) and center of mass (green curve). In general, the partial scans were sufficient to recover the major symmetries of the object correctly, leading to lower average errors for center-of-symmetry alignment as compared to center-of-mass. Of the 907 models tested, the center of symmetry for a scan was closer to that of the entire model 90% of the time. On average, they were closer by a factor of 1.5, with better results occurring when a particular viewpoint caused significant portions of the model to be missing.

Figure 4.4b shows histograms of the rotational misalignment of the partial scans with respect to the full object when aligned with principal symmetry axes (PSA) and with the principal axes (PCA). Note the large peak near zero in the blue curve, indicating that PSA recovered the rotation for most range scans to within a few degrees of that computed for the entire object. In contrast, PCA provided a larger spread of misalignment angles. Moreover, even though less than half of the surface was available in any scan, we found

that the coordinate frame chosen with PSA was within 5 degrees of a human chosen set of axes for the whole object in 70% of the scans, as opposed to only 50% for PCA.

4.1.4 Limitations and Future Work

There are a number of limitations of using PSA. As in the case of PCA, the main limitation of this method is that axes are only defined up to reflection. Specifically, the coordinate system obtained by our method is invariant to reflection of the model through its center. In order to overcome this, we use the same solution suggested by [12] and define "right" ("up", and "out" respectively) to be the side with the most mass. A second limitation of the method is that the algorithm for choosing the planes is greedy. Especially in the case of partial models, rather than choosing the first plane independently, a more robust method might be to find the intersection of three orthogonal planes the contain the most symmetry. The problem with this alternative method is that the brute-force search over the entire PRST to find such a point would take $O(n^6)$ in the size of the grid. A more efficient method to find such points is in the realm of future work. Finally, another venue for future work is to combine PCA and PSA, as each of those methods measures a slightly different quality of the model.

4.2 Matching

Over the last two decades, the ever increasing availability of 3D content online has caused a shift in the general approach people use to create digital models. In the past, most digital models were created from scratch; by combining basic shapes such as spheres and cubes to create progressively complex forms, applying virtual modeling tools to basic shapes to sculpt a form, or by scanning a real-world 3D model. Now that large

databases of models (in many cases professionally created and textured) are prevalent, a more common approach is to search for models that are of the shape required or that are similar enough to the desired shape so that simple modification suffices. This is especially true for mechanical designs, where the option of using existing components is a serious consideration.

In this context, a very interesting question is "given a large database of 3D content, how do I find a model that has a shape similar to the one I am looking for"? While a textual search may yield models that have the appropriate labels added, and in certain cases searching based on color and texture may suffice, in order to solve the general problem, some method for comparing the shape characteristics of 3D models is necessary.

To address precisely this issue, over the last few years, a variety of retrieval algorithms have been proposed that enable the efficient querying of model repositories for a desired shape. Typically, the query is in the form of a 3D model, and algorithm returns a list of models from the repository that it has computed to have similar shape characteristics to the query model. As with any general database retrieval application, the two principal concerns of such an algorithm are to discriminate between different classes of models effectively, and to compute matches efficiently in both space and time.

In this section we will examine some of the traditional algorithms used for shape matching, discuss the motivations and characteristics of those algorithms, and explore using the PRST and symmetry information in order to improve shape matching.

4.2.1 Shape Descriptor

In practice, most shape retrieval algorithms use a *shape descriptor* for matching, a representation of shape that can be matched efficiently, and computed robustly. Typically, such a descriptor is a vector of fixed dimensionality, making each model a point in

an N-dimensional descriptor space. This approach has the advantage of reducing the computation of similarity or "distance" between any two models to computing the distance between two points, often (but not always) using a simple Euclidean distance metric, generally satisfying the efficiency requirement of the application. Furthermore, the descriptor for every model in the database may be computed in a preprocessing step, making the actual retrieval step even simpler.

In order to create a robust shape descriptor, the values in the descriptor must capture the shape characteristics of the model and not be susceptible to noise or other modifications that do not affect our perception of is shape. One of the key difficulties arising from this requirement is that humans identify shapes as similar even under various global transformations. For example, consider the car models in Figure 4.5. Even though some are rotated through various axes, or scaled by various amounts, a human will still identify them all as having a similar shape. Thus, a major challenge faced by a shape descriptor is to effectively measure the minimal distance between two shapes over the space of all such transformations. In general, there are three solutions to this challenge: an exhaustive search over the entire space of transformations, normalization of the models into a canonical coordinate frame so that the models may be automatically assumed to be optimally aligned, and creating a shape descriptor that is invariant to that class of transformation. In practice, exhaustively searching over the space of transformations is prohibitive and so most shape descriptors use one of the remaining two solutions for each of the basic similarity transforms: Translation, Rotation, and Scale.

In the next subsection we will review some of the shape descriptors in common use.

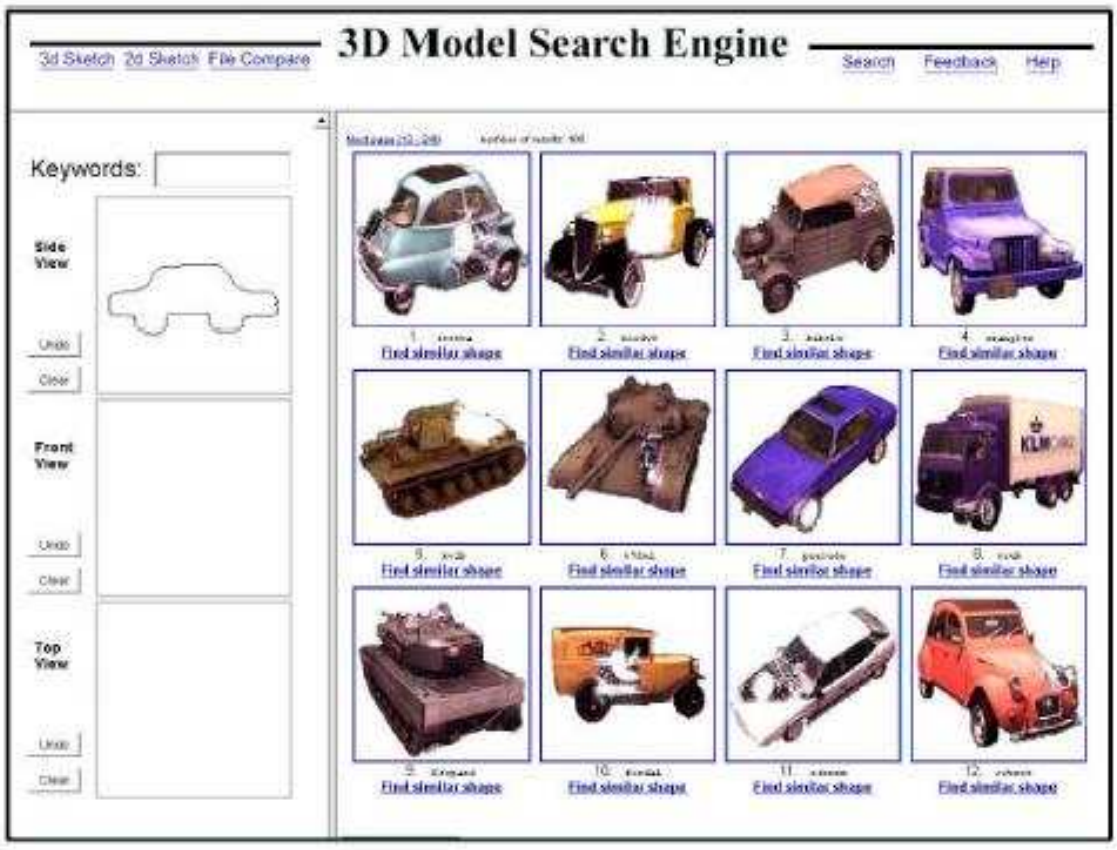


Figure 4.5: While these cars have different sizes, rotations, and color, they are all considered to have similar shape. A robust matching algorithm needs to account for such transformations when producing a similarity metric. (images courtesy of the Princeton Shape Repository).

4.2.2 Previous Work

A basic approximation of the shape of a model can be computed by storing a histogram of the surface points, either as a function of the distance from the center of mass (*Shells*), as a function of the spherical angle (*Sectors*), or a combination of both. These types of approximations were first used by Ankerst et al. [2] to address the challenge of protein matching, and have the advantage of being translation invariant. Using the distance from the center of mass gives an invariance to rotation and using the spherical angle allows

invariance to scale. Typically, such a histogram is stored in a series of "buckets" to ensure robustness to noise, with the values in the buckets possibly smoothed.

As an extension of the Shells and Sectors method, Osada et al. [40] describe a shape descriptor that uses a histogram of distances between point-pairs (and areas of point-triples, etc.) on the surface of the model. As with the Shells method, the *D*2 Shape Distribution method is translation and rotation invariant. Through a series of experiments using various model databases available online, they show that Shape Distributions are both robust to noise and able to discriminate between classes of models.

Horn [21] proposed a shape descriptor known as the Extended Gaussian Image (EGI) that represents a 3D model by a spherical function. Given a 3D triangle model, EGI computes for every point on the surface of a unit sphere the probability that a random point on the model will lie on a triangle that is facing in that direction. This descriptor is translation and scale invariant and is invertible for convex models.

For a given polygonal model, the Gaussian Euclidean Distance Transform (GEDT) [24], creates a 3D voxel grid that encompasses the model and computes for every voxel the Gaussian euclidean distance to the nearest point on the surface of the model. Using the GEDT as a descriptor is based on the notion that the similarity between two models is a function of the distance between corresponding points on their surfaces. Thus, the GEDT is a voxel representation that describes not only where the point on the surface is, but also how close an arbitrary point in space is to that surface. The GEDT is not invariant to scale, rotation or translation and must be normalized for each of those transformations.

All the above methods use local surface properties to compute a meaning of shape. In contrast, Kazhdan et al.[24] described an approach using their reflective symmetry descriptor (RSD). They employed the maximum difference between the symmetry measures of any two corresponding planes through the center of mass as a dissimilarity measure

for a pair of 3D meshes. Our method extends the RSD to use the Symmetry Transform as a descriptor.

4.2.3 Symmetry

Our efforts are motivated by the observation that symmetry properties are often consistent within a class of objects (Figure 4.6). For example, although chairs may vary in their size, whether or not they have arms, etc., their reflective planar symmetries are almost always the same (perfect left-right symmetry, a weaker global symmetry between the back and seat, local symmetries through the seat and back, etc.). This is true for many other object classes as well, including airplanes, tables, people, etc. Perhaps it is possible to classify 3D meshes automatically by comparing their computed symmetries to those of meshes in a supervised training set.

We extend the work done by Kazhdan et al. [24, 26] by considering symmetries with respect to all planes through an object's bounding volume. In our matching method, we measure the dissimilarity between a pair of aligned meshes as the L^2 distance between their discrete PRSTs (Section 3.2.3), weighting the differences between corresponding bins of the PRST by $\sqrt{\sin\theta}$ (where θ is the polar angle of the plane represented by the bin) to account for different bin sizes. This measure produces a large distance when there are planes for which one object is (nearly) symmetric, while the other is not.

4.2.4 Results

In order to evaluate the PRST as a shape descriptor for shape-based retrieval and classification applications, we ran a set of "leave-one-out" experiments with the Princeton Shape Benchmark test set [44], a database of 907 polygonal models partitioned into 92 classes

Figure 4.6: Symmetries can play an important role in recognizing objects within the same class. Note that even though the chairs have different heights and widths, they all share the same symmetries: perfect left-right symmetry, and strong symmetries in the seat, leg and back (images courtesy of Digimation).

commonly used for shape matching evaluations. In order to focus our study on shape representation rather than alignment, we manually registered all models into a common coordinate frame before matching every model against all the others. The L^2 distances between PRSTs were used to produce a ranked retrieval list for each "query" model, and then statistics were computed to evaluate how often models within the same class appear at the front of the computed retrieval lists.

Figure 4.7 shows average precision-recall plots comparing our retrieval performance with that of Kazhdan et al.'s planar reflective symmetry descriptor (PRSD) and the Gaussian Euclidean Distance Transform (GEDT), which is currently used in at least one shape based search engine [16]. The horizontal axis of this plot represents increasing recall

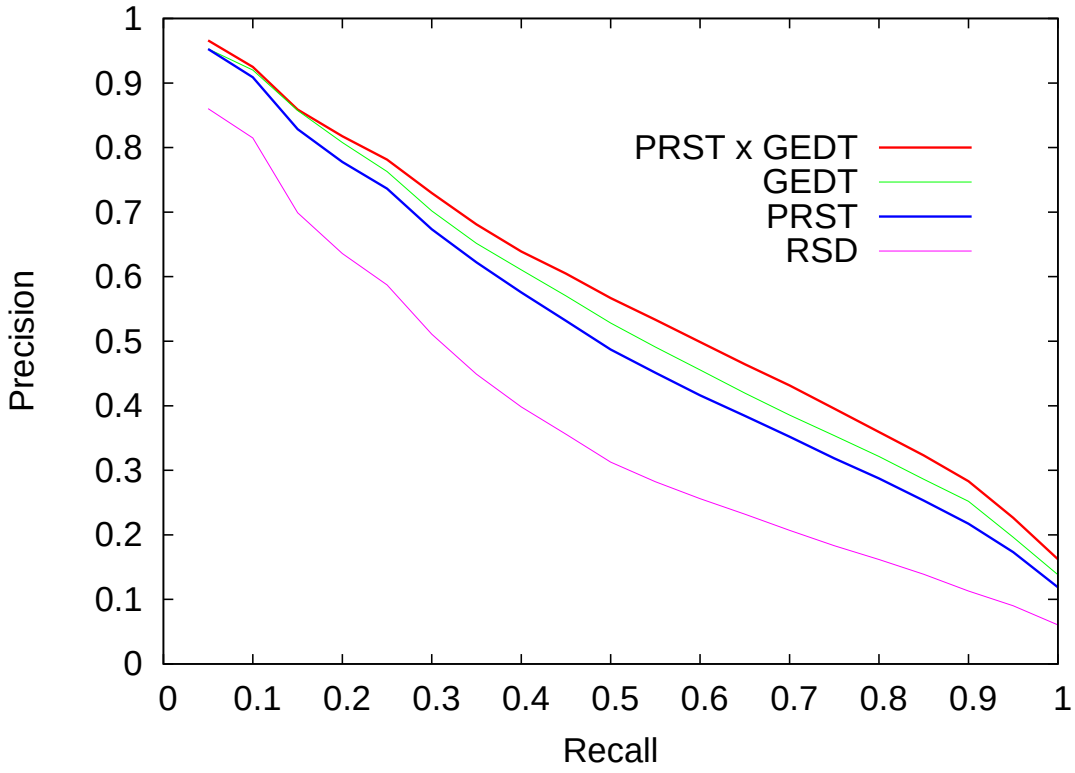


Figure 4.7: Plot of precision versus recall achieved with four shape matching methods: PRSD (magenta), PRST (thick blue), GEDT (green), and the combination of PRST and GEDT (thick red). Note that the PRST captures information complementary to other shape matching methods, hence can be used to augment their retrieval performance.

values (fraction of the query's class retrieved), while the vertical axis represents retrieval precision (fraction of the retrieved models that are in the same class as the query). Higher curves represent better performance.

We find that the precision achieved when matching with the PRST (thick blue curve) is higher than with the PRSD (magenta curve) for every recall value. This confirms the expectation that the extra information provided by the PRST (symmetries for off-center planes) adds precision for shape matching. Of course, it is more expensive to store ($32,768$ floats versus $1,024$ floats) and to compare (0.1 ms versus 0.004 ms), but the extra cost seems worth the improved performance for most applications.

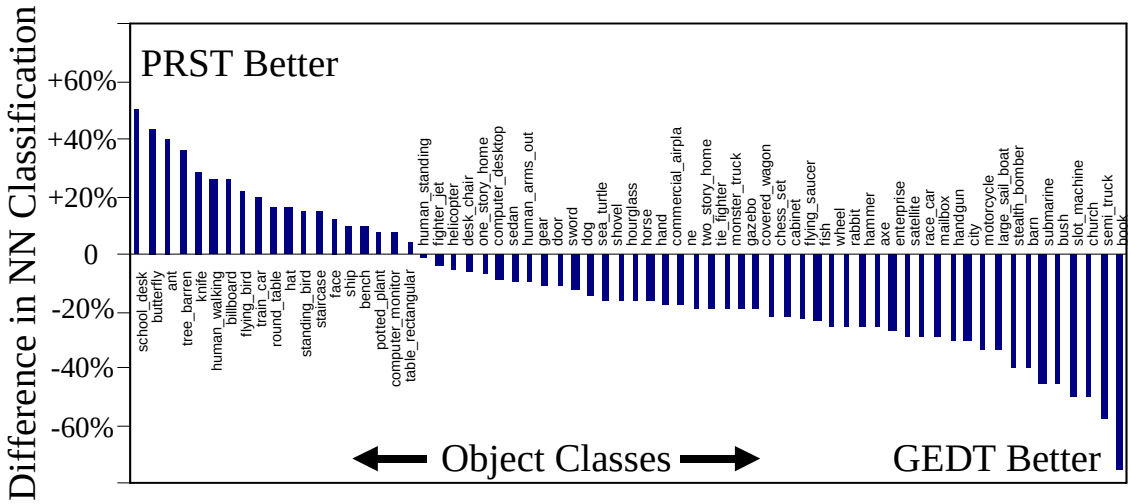


Figure 4.8: The PRST provides better retrieval performance for some classes (top), while the GEDT is better for others (bottom). Combined, they produce better retrieval performance than either alone.

We also find that both the PRSD and PRST provide less matching precision than the GEDT (green curve) on average. We believe that this is because many object classes within the Princeton Shape Benchmark have the same symmetries (e.g., almost all man-made objects have perfect left-right symmetry). Even though the symmetries may be consistent within a class, they do not always help discriminate between classes. However, we observe that the PRST provides better matching results for some types of objects (i.e., ones with distinctive symmetries), while the GEDT provides better results for others (Figure 4.8). So, we find that combining the two shape descriptors (by simply multiplying the L^2 distances computed separately for the two descriptors) provides better retrieval performance than either alone (the thick red curve in Figure 4.7). The nearest neighbor classification rate and discounted cumulative gain scores for the combined method were 69.2% and 68.6%, respectively, which represent good retrieval performance for this data set [44]. This leads us to conclude that the PRST, while perhaps not the best shape representation for retrieval of this type of data on its own, can provide useful information

for shape-based matching and can be used to discriminate classes of objects that are difficult to distinguish with other methods.

4.3 Viewpoint Selection

”You are trying to build a catalog of one hundred models of real-world objects. In order to showcase these models, you need to take pictures of them to put in the catalog. From what direction do you take those pictures?” This problem and others like it may be difficult to solve, because 3D objects may look significantly different when viewed from diverse directions.

Neuroscientists and Psychologists have long studied the process by which humans recognize 3D objects. One theory is that the recognition process is based on a number of primary images of the same object from various viewpoints, and that the brain tries to make sense of the current view by trying to interpolate those primary images. Support for this theory may be found in the surprise shown by people viewing an object from a new direction for the first time, (e.g. someone seeing their profile in a mirror for the first time).

Further research has shown that certain viewpoints can be said to be preferred by humans over other viewpoints. This preference may be measured using a number of criteria such as the response time or recognition error when naming objects seen for a short period. Other criteria include the viewpoint which people say best describes the model, or even an assignment of “goodness” to various viewing directions.

In computer graphics there have been a number of heuristics proposed for finding optimal viewing directions for various problems, but to this date, none have included symmetry. In this section we will explore an automatic solution for finding preferred

viewpoints of 3D models based on the intuition that symmetry represents redundancy, and hence should be minimized.

4.3.1 Previous Work

User Studies A preferred viewing direction for an object is called a "canonical view". This term was first used by Palmer et al. [37]. to describe a view "humans find easiest to recognize and regard as most typical". They ran a series of experiments where people classified multiple views (top, side, front and back, as well as intermediate views of 45 degree angles) of a single object (e.g. a horse), and gave subjective ratings to those views, based on how well they depicted the object using a scale of 1 ("very like") to 7 ("very unlike"). Further experiments gave a different set of participants the same set of images in a classifying experiment. The images were shown for a short time, and participants were asked to classify the object seen. The best results were obtained from images that had a high ranking in the previous experiment (the preferred viewpoints). Analysis of these experiments and others indicated that participants preferred the same viewpoints, independent of the task. Specifically, for most models, participants preferred off-axis views to head-on or side viewpoints.

Blanz et al. [5] performed a similar user study to determine factors that influence the canonical views chosen by humans used to display 3D models. In their study, participants were given a number of fully shaded digital models to view. In one of the experiments, rather than being given a set of images of an object from a fixed set of viewpoints, the participants were allowed to choose the viewpoint from which they would take a photograph. The viewing direction was manipulated using a three-degrees-of-freedom input device which gave the participants full control to choose their preferred viewing direction. Results from their experiments show that views containing significant visible

features of the surface are preferred, and that many times off-axis views are preferable to front or side axis views, results similar to those found by [37].

Automatic Viewpoints There are several proposed algorithms in the graphics community to generate automatic canonical viewpoints for various models:

Based on the observation that a good viewpoint is one that allows a viewer to see the most features of a polygonal model, Kamada et al. [22] seek to minimize the number of degenerate polygons in the image. These are polygons in the current view whose edges project over the each other. In order to achieve this goal, their method simply minimizes the maximal angle deviation of the view direction with the normal of the polygons in the model. While this works well for wire-frame models, this method does not take into account the loss of feature detail resulting from occlusion in real-world models.

A similar effort is introduced by Vázquez et al. [49]. They try to find quality viewpoints by maximizing the "interesting" content in a view. They introduced the notion of *viewpoint entropy* as a method of determining how much information is conveyed by a given set of planar faces. Their method proposes that the amount of information in a scene is determined by the amount of faces seen (since every face contains some information about the scene), and how well those faces are seen (degenerate faces to not contribute to the information gathered). As such, they use the metric

$$I(S,p) = \sum_{i=0}^{N} \frac{A_i}{A_t} log \frac{A_i}{A_t}$$

Where A_i is the projected area of face i over a sphere centered at p, A_t is the area of the sphere, and N is the number of faces in the scene. Entropy will be maximized when the relative area of each polygon is the same. Occlusion is included in determining the projected area of a polygon, solving the issue of hidden surfaces, but the algorithm

assumes that every face is equally important, and does not take into account that certain portions of the model may be more informative than others.

Rather than trying to classify a viewpoint by the amount of surface area it sees, Lee et al. [28] attempt to measure the information content of every portion of the surface. They note that a flat plane composed of many polygons is less informative than one with rugged features. Similarly, a portion of the surface that contains some reoccurring texture might not be as important judged by its information content as a unique pattern (i.e. letters). In order to quantify the information content of a viewpoint, they look at the *Saliency* of a mesh. The saliency of a point on a mesh is found by aggregating the surface curvature of the mesh at that point on a number of scales. They propose that changes in the curvature of the mesh correspond to interesting features of the surface. For a given viewpoint, the quality of the viewpoint is determined by summing up the total saliency seen. Note that their definition of saliency, is very much a local property of the shape (though the curvature may be computed on mid-sized portions of the mesh), thus surfaces with large variation at high frequency, may be deemed "interesting" without really contributing to the optimality of the viewing direction.

Finally, Abbasi et al. [1] and Lee et al. [29] are interested in finding a minimal set of primal images of a given object (specifically a face in [29]) such that any other image of the same object taken from any viewing direction would look similar to one of the primal images. They accomplish this by taking images of the object from many viewing directions. These images are grouped by how similar their contours look, and most are discarded to leave a small set that depicts the possible contours an image of that object might have from any viewing direction. While this method doesn't solve the problem of what is the "best" viewing direction, one or more of the primal images probably correspond to canonical viewpoints.

4.3.2 Symmetry

The previous methods of [22, 49, 28] for computing the optimal viewing directions are all based on local characteristics of the model. In each case, the contribution of a portion of the model to the "goodness" of the viewing direction is independent of what other parts of the model are seen in the same view. If there was a strong repeating pattern on the surface of the object, the method suggested by [28] might automatically reduce the value of those points, but in general terms of information content, these methods all fail to account for redundant information in the form of symmetry. We propose to remedy this by explicitly accounting for the symmetry of the model.

In our approach, we minimize the amount of symmetry seen. If a model has perfect left-right symmetry, and we know exactly what one side looks like, no extra information is gathered from observing the other side. However, even finding a viewpoint for half a model is not simple. A purely side view of the model might not capture enough information about the front or back of the model, and might contain redundant information itself if the model has any front-back symmetry.

Our method begins with the primary symmetry of the object and then uses the local maxima extracted from the PRST to minimize the amount of symmetry in the direction of the viewer. More specifically, for each plane appearing as a local maximum in the PRST, the preferred viewing direction is along the normal to the plane. We compute the viewpoint score for a view direction v as $S(v) = \sum_{u \in W} |v \cdot u| \cdot M(u)$ where $u \in W$ is a plane of local symmetry and $M(u)$ is the symmetry score for that plane.

With this relatively simple scoring function it is enough to do an exhaustive search to find the optimal viewpoint, although a gradient descent method such as introduced by [28] may be used to accelerate the computation.

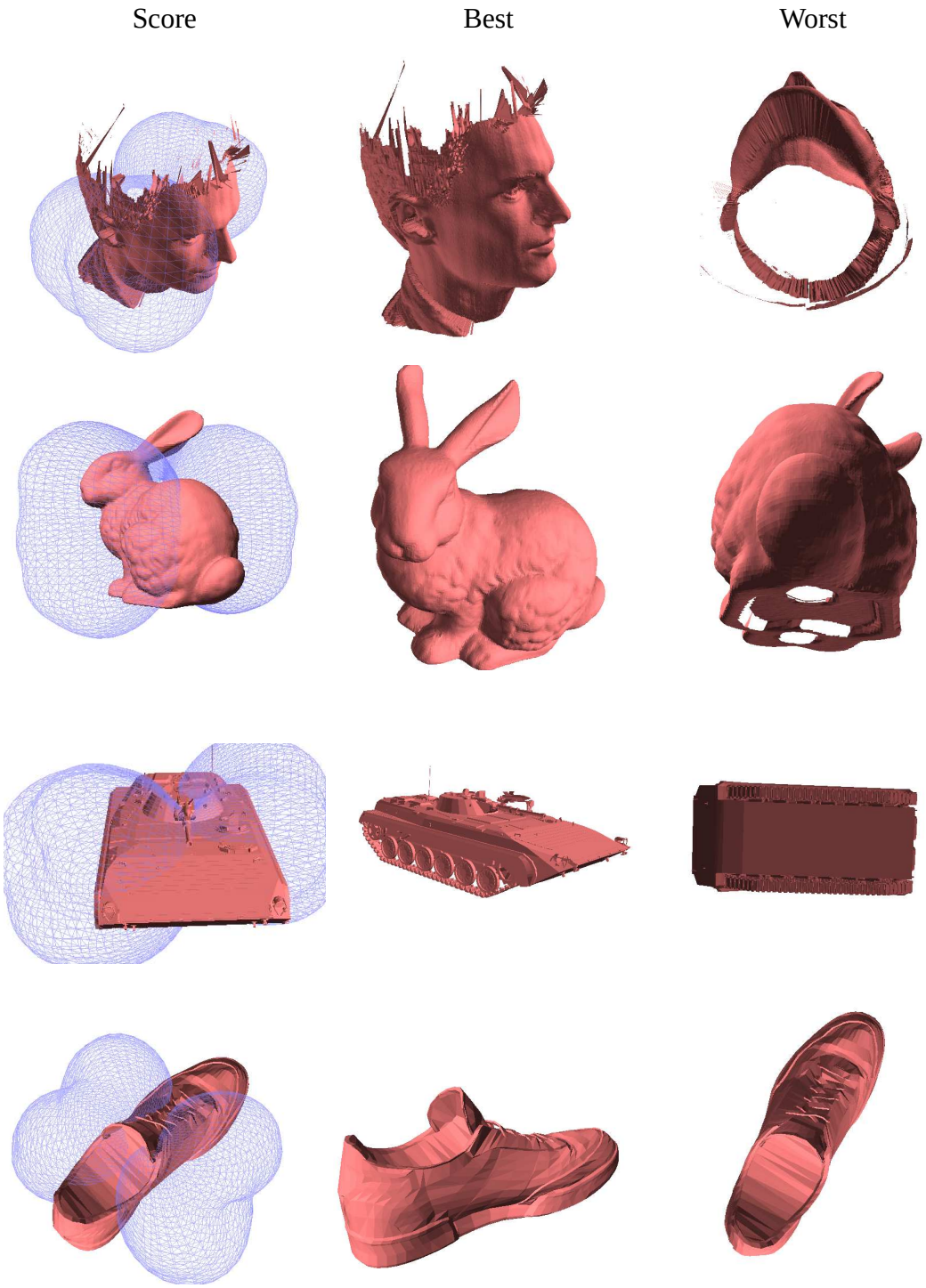


Figure 4.9: At left, we show the viewpoint score for each model as a spherical function. The visualization is obtained by scaling unit vectors on the sphere in proportion to the quality of the viewpoint from that direction. The images at center show the best viewpoint selected by our algorithm. The images at right show the worst viewpoint selected.

4.3.3 Results

We ran our algorithm on 22 different models, computing the PRST on a 64x64x64 grid, and extracted the best viewing direction for each. As can be shown in middle column of Figure 4.9, the best viewpoint found for these object was always $20-35$ degrees off-axis, similar to the findings obtained by [5].

We show the results of our approach in the middle column of Figure 4.9 — each object's local symmetries repel the viewpoint, such that the final selected view is off of the major axis of the objects. Note that even though we only show the best viewpoint, it is possible to extend this implementation to produce multiple "interesting" viewpoints for the user's selection. This may be done by either considering secondary local maxima of the viewing function (left column in Figure 4.9) or by removing from the computation the planes that contribute most to the best viewpoint and trying again. Viewpoints to be avoided may be found by finding the minima of the viewpoint function (right column).

4.3.4 Limitations and Future Work

Our algorithm chooses the optimal viewing direction, but there is still a degree of freedom in choosing the rotation around the viewing axis. In our implementation we always used 0 degrees rotation, keeping the original angle of the object was loaded. In practice however, the 'up' direction of the model should be the 'up' direction in which it is most commonly found. There is no real only way to measure this with symmetry expect maybe to assert that the strongest symmetry is probably left-right, and use that as a basis.

A possible future extension of the application would be to incorporate more model information in order to improve the the viewpoint function. Relevant information could include the stability of a pose (the lower the center of mass the better), an automatic

computation of the “up” direction such as method suggested by [15], or other global shape characteristics,

Chapter 5

Editing Applications

In contrast to analysis applications, editing applications actively change the model they are applied to. Examples of editing applications include morphing, segmentation, simplification, remeshing, composing, and smoothing. In this chapter we will describe two applications, segmentation and remeshing, and show how we can use symmetry information to improve their results.

5.1 Remeshing

The increasing practicality of 3D scanning has caused an ever growing trend toward more accurate 3D models of objects. While the creation of accurate models is beneficial for practically any computer graphics application, this comes at the cost of model complexity. Indeed, over the last few years, models with millions and even tens of millions of polygons have become commonplace, and new scanning technologies are promising to increase this limit further.

As a result, many applications that require rapid computation such as interactive rendering, editing or physical simulation are hard-pressed to cope when dealing with very large models. Consequently, over the last decade many methods have been developed to *remesh* 3D models, simplifying them while closely approximating their geometry. While such approximation methods maintain the geometry of the original model while remeshing, they generally do not explicitly preserve other types of high-level information that might be important to the application.

In this section, we will review some of these methods, and propose a new "symmetry-aware" remeshing algorithm that will automatically detect symmetric regions and actively preserve and even strengthen those symmetries during simplification. Portions of this work were performed in collaboration with Aleksey Golovinskiy, Szymon Rusinkiewicz and Thomas Funkhouser, and were previously published as [38].

5.1.1 Previous Work

Surface Approximation Due to the general usefulness of simplified models for many applications in computer graphics, there is a long history of techniques designed to approximate 3D surfaces while optimizing for various types of geometric error.

Probably the most common method for mesh simplification involves approximating the surface locally either by greedily clustering local groups of triangles or by collapsing the edges of a mesh using a local error metric that captures the geometry deformation of the simplified area [19, 27, 18, 48]. Generally the error of such a method captures a measure of the local L^2 distance between points on the original and approximating surfaces. Other methods, including Katz et al. [23] and Shlafman et al. [45], seek to combine sets of faces containing similar properties to generate characteristic regions.

An alternative paradigm for simplification involves generating a maximally-accurate approximation with respect to a global error metric, while reducing the number of faces. Work such as Hoppe et al. [20] generates an energy functional based on a point-to-surface distance of the input mesh. This error function captures the curvature and surface variations from the original model. Departing slightly from the use of surface distance (the L^2 metric), Cohen-Steiner et al. [11] define a metric based on the normals of the surface (The $L^{2,1}$ metric). They also solve the global error function by fixing a number of *proxies* and then optimally placing these proxies to best approximate the surface. Using normals rather than the more widely used surface distance is motivated by the desire to generate more visually pleasing results, for example by more accurately retaining highlights.

It is important to note that in all these methods, the metric used is one that will approximate the surface locally. At no time does the approximation of any part of the surface explicitly affect the approximation of the rest of the model, except as part of a global optimization. Thus there is no attempt to preserve global features such as symmetry. Consider for example, the mask in Figure 5.1a. The model is fairly left-right symmetric, but the remeshed models created using the algorithms suggested by [18] (Figure 5.1b) and [11] (Figure 5.1c) do not preserve that symmetry.

Based on on the observation that in general, people are sensitive to symmetry, and will notice departure from symmetry more readily than some general deformation of the surface, our remeshing technique incorporates symmetry information from the PRST directly into the algorithm, yielding a symmetric approximation (Figure 5.1d).

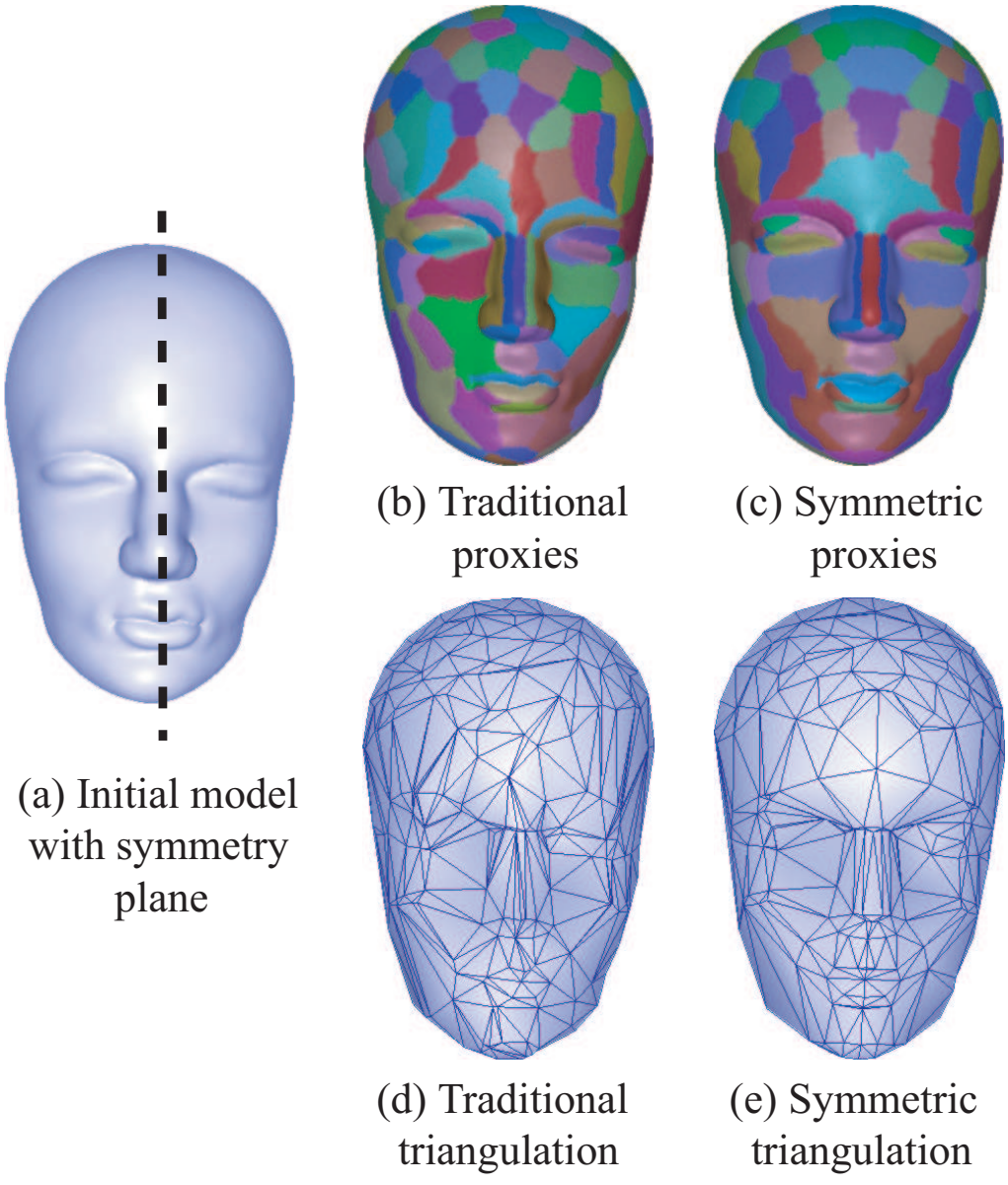


Figure 5.1: Remeshing without symmetry does not create a symmetric triangulation without explicit accounting for symmetry. Column (a) shows a model (62K triangles) of a relatively symmetric mask. Column (b) shows the triangulation of the model using QSlim [18] algorithm, column (c) shows the triangulation using [11], and column (d) shows the result of our algorithm.

5.1.2 Symmetric Remeshing

Our goal is to remesh a surface in a "symmetry aware" environment. If we were only concerned with perfect symmetries, a simple algorithm would be to remesh using any existing technique, then force symmetry by mirroring the output. However, in most practical situations, the symmetries will be imperfect. For example, there may be near-symmetries that are not perfect because of noise or differing tessellation, or partial symmetries in which regions of the model may be symmetric about different planes. In addition, some models may exhibit approximate symmetry. In these cases, faithful remeshing (resulting in a large number of polygons), should prioritize geometric accuracy. However, remeshing such models with *fewer* polygons should result in a symmetric output, provided that doing so introduces error of the same order as that necessarily introduced by remeshing. In this way, we avoid symmetrizing if there is no reason to do so: the choice to remesh with many polygons shows that the user's overriding concern is to preserve detail accurately.

To address this, we propose a framework for model simplification that automatically detects symmetric regions and actively preserves and even strengthens those symmetries during simplification. Our approach is to modify the algorithm proposed by [11] so that it explicitly preserves symmetry. We chose to begin with this algorithm because it has been shown to produce good results for low polygon counts, for which the careful choice of symmetrization algorithm has the most visible impact. Specifically, we have adapted all stages of Variational Shape Approximation to include symmetry. During the proxy generation stage, we extend the notion of a proxy to include multiple connected components related by a pre-defined set of symmetry transformations (e.g., planar reflection). Thus, while growing proxies we consider not only neighboring triangles, but also reflected triangles. In the triangulation stage, we explicitly find corresponding symmetric

points based on the proxies grown previously. We force the triangulation to follow these correspondences as much as possible, yielding a more symmetric triangulation.

Symmetric Proxy Generation The first step of the algorithm is to generate *proxies*, planar regions that closely approximate local sections of the surface. This is done with an iterative technique based on Lloyd's algorithm. At every iteration:

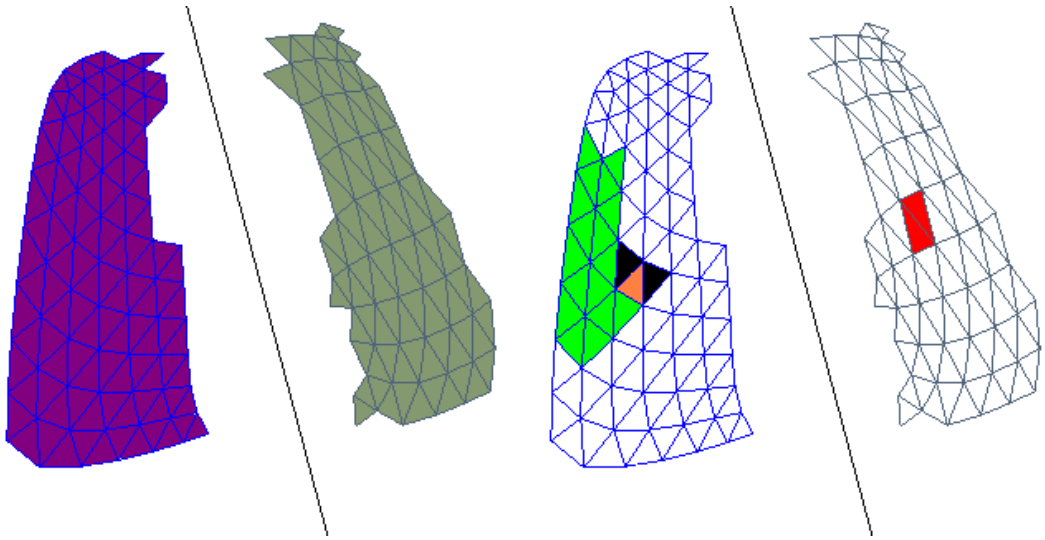
1. Every proxy is assigned triangles of the model from a single priority queue. When a triangle Δ is removed from the queue and assigned to a proxy P, all triangles adjacent to Δ are added to the queue with a weight defined by the compatibility of those triangles to P. This assures that each proxy is a single connected component.

2. Once all triangles have been assigned, optimal proxy parameters (i.e., center and normal) are re-computed based on the triangles currently assigned to the proxy.

In order to avoid converging to local minima, small proxies are occasionally deleted and new proxies introduced at appropriate locations ("teleportation").

Our method follows this approach but generalizes the definition of a proxy by allowing it to represent a planar region, possibly transformed by a discrete set $T_1..T_k$ of symmetry transformations. This is a key component of our algorithm, as proxies may now contain multiple connected components symmetric to one another. We allow a maximum of one connected component per symmetry transformation (including the identity), called *patches*. As an example, we show in Figure 5.2(a) a proxy with two patches, one associated with the identity (purple triangles) and one associated with the reflection plane shown (green triangles).

In our method, a triangle Δ may be assigned to a proxy P if it is adjacent to a triangle previously assigned to P, **or** if the triangle nearest to $T_i(\Delta)$ has previously been assigned



(a) (b)

Figure 5.2: (a) A proxy with two patches. The purple patch is associated with the identity and the green patch is associated with the reflection plane shown. (b) When a triangle (orange) is assigned to a proxy (green), the neighboring triangles (black) are added to the queue. The triangles across the plane of reflection (red) are added as well.

to P. Consider the example shown in figure 5.2(b). The orange triangle was recently assigned to the proxy shown (dark green), so any of the black triangles may now be added to the proxy; under our extended definition, the red triangles may be added as well.

The remainder of the iterative algorithm is nearly unchanged, with triangles extracted from the priority queue in order of increasing error, and new patch centers and positions computed after each iteration. The teleportation process is augmented to allow individual *patches*, in addition to entire *proxies*, to be deleted.

After a few iterations, we observe that, where possible, proxies will have symmetric patches corresponding to "good-enough" symmetry transformations. This will ultimately lead to symmetric outputs, since the boundaries between the proxies determine the topology of the final surface. Note that the order with which triangles are selected from the priority queue completely determines whether or not the final proxies are symmetric. That

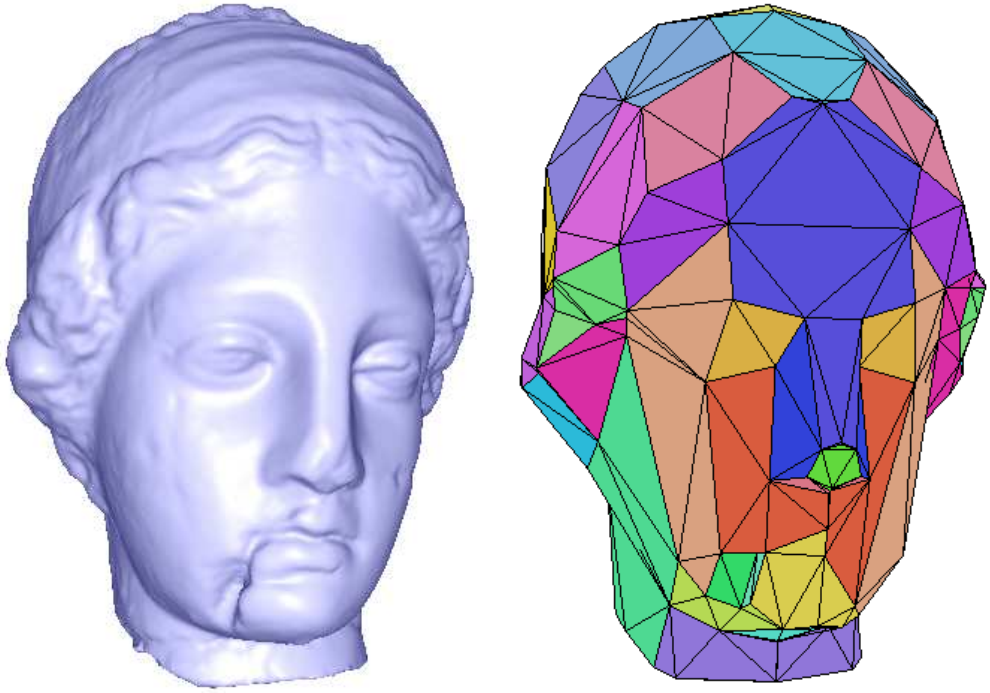


Figure 5.3: This model is quite symmetric, except for the gash in the right side of the chin. Note how our algorithm does not create symmetric patches for the proxies that approximate that area, because the error introduced would be too great. The remainder of the head however, has symmetric patches. The model was remeshed using 50 proxies and one plane of symmetry (up to two patches per proxy).

is, symmetric proxies will be created if and only if the error of doing so is comparable to the error introduced by the remeshing itself. Multiple planes of local symmetry are used automatically in the appropriate regions, and no user intervention is required. An example of this can be seen in Figure 5.3, where the area around the chin is not symmetric. Therefore, in this region, unlike the the rest of the face, the proxies are generated without a reflection.

Point Correspondences Once the proxies are placed, our goal is to place a set of symmetric points on the surface that we will later triangulate.

[11] place two categories of points onto the new surface. The first category of points is *anchor vertices*, placed at junctions where three or more proxies meet. The second category of points is *secondary points*, which are placed along the boundaries between proxies to improve the geometry approximation of the new surface and to assure at least three points per proxy boundary. In each case, the new vertex position V is computed from the original position v by calculating for every proxy P_i the optimal position $P_i(v)$ (the nearest point on the proxy), then averaging these optimal positions to obtain the final position $V = \frac{1}{n}\sum P_i(v)$ on the new mesh.

Our modifications to this vertex-placing algorithm involve determining symmetric correspondences between points. The clearest indication of correspondence occurs when a set of anchor points are adjacent to the same three proxies for different symmetric transformations $T_0..T_k$. Thus, for anchor vertices, we check whether the proxies adjacent to some v_i all have reflections that meet in an identical configuration at some v_j. If this is the case, then we establish a correspondence between v_i and v_j, and further adjust their new positions to be perfect reflections of each other.

We find matching secondary points by establishing correspondences between proxy boundaries: mesh-edge paths that run between corresponding anchor vertices are considered to be in correspondence with each other. When adding a secondary vertex to a boundary, we search through corresponding boundaries for the nearest symmetric vertex. As with anchor vertices, we adjust the positions of such correspondences to be perfect reflections.

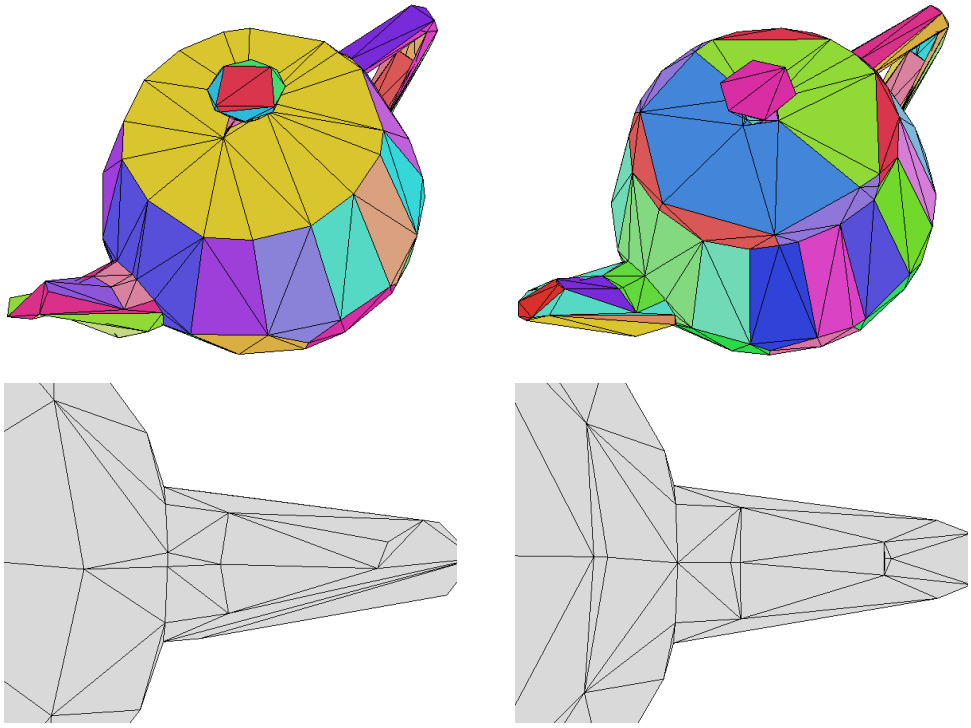
Triangulation Once we obtain all our correspondences between anchor and secondary points, all that remains is to triangulate the set of points symmetrically.

We triangulate the proxies in the same manner suggested by [11]. This consists of flooding using Dijkstra's shortest-path algorithm, where the sources are the anchor and secondary points, and with each edge weighted according to its length. At the end of the flooding, adjacency of regions implies that their source points should be connected in the resulting triangulation. As a final pass, edge flipping and edge removal are run to obtain a better tessellation.

We use the same flooding algorithm, but run edge flipping on each proxy separately to prevent edge flips in one proxy from altering the triangulation of its neighbors. This process is guaranteed to converge to the same topology for symmetric proxies if the associated triangles are co-planar. In practice, we find that this method produces a consistent topology even when the triangles are not co-planar.

Proxies Spanning Reflection Planes The algorithms for proxy assignment and triangulation, as described above, operate at the granularity of entire triangles of the input. This leads to potential problems for proxies that lie near the planes of symmetry, since triangles that originally cross a symmetry plane will be assigned entirely to one or another component (patch) of a proxy. The result is non-symmetric triangulations along the planes of reflection, as can be seen in Figure 5.4a. Obtaining a symmetric output therefore requires us to treat this situation as a special case.

We preprocess the model by splitting each triangle that crosses a reflection plane, creating new vertices where the original triangle's edges intersect the reflection plane and replacing the original with three new triangles. The proxy-fitting stage proceeds as previously described. Then, at the beginning of the triangulation stage, we explicitly check each of the split triangles to see if both sides of the triangle were assigned to different patches. If so, we explicitly enforce that the vertices we inserted to perform the



(a) (b)

Figure 5.4: This figure shows the need for splitting faces that cross over reflection planes. (a) Without splitting faces, triangles that cross the plane of reflection are assigned to only one of the reflections of the proxy. This causes a jagged triangulation. (b) When we split the triangles that cross the reflection plane, the tessellation is symmetric. The models have 342 and 358 faces respectively.

split (lying on the reflection plane) be anchor vertices. The rest of the algorithm proceeds as above, with the result that the triangulation now becomes symmetric (figure 5.4b). In the examples shown, splitting triangles adds an average of 3% to the number of triangles.

Error Metric While it is not possible to change the amount of weight given to symmetry vs. geometric deformation, since all we do is add more triangles for the priority queue to consider, in certain cases it might be important to consider symmetry more strongly than surface deformation. In these cases, we have observed that the $L^{2,1}$ metric proposed by [11] is not ideal for preserving symmetry. This is because it considers only normals,

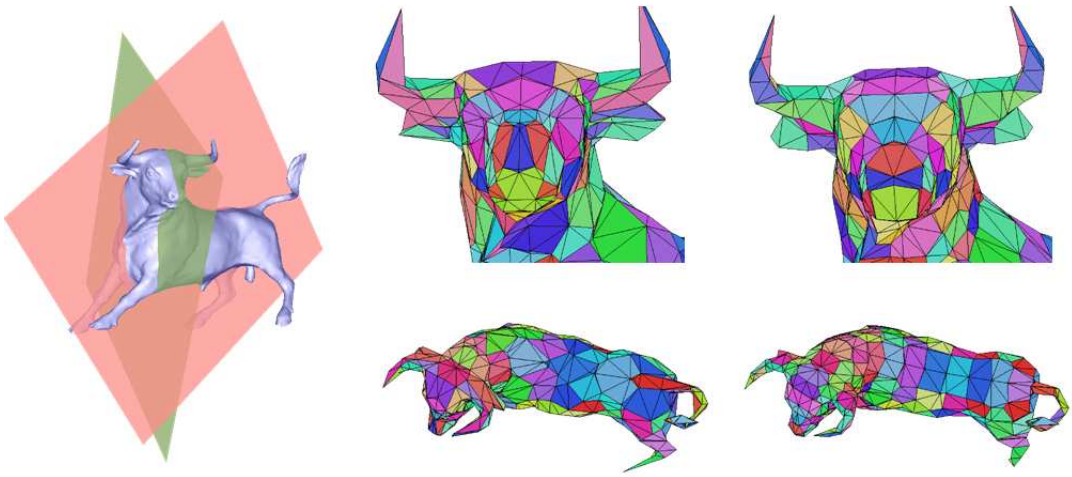
which tend to be more sensitive to noise and small deformation than does the geometry itself. Therefore, in order to allow more freedom for the algorithm to capture a model's near-symmetries, we have considered other, more "symmetry friendly" error metrics.

We begin by noting that in the $L^{2,1}$ metric, the distance of a triangle to a proxy depends only on the normal, and indeed the center of the proxy is not relevant, being set as the weighted average of the triangles in the proxy for book-keeping purposes only. We note that if the metric were simply the weighted Euclidean Distance of the centers of the triangles to the center of the proxy, then the optimal proxy position would also be at the weighted average of the triangles, without consideration of the normal. We expect that such a purely position-based error metric would allow the algorithm greater opportunity to capture near-symmetries, at the expense of less-faithful preservation of the original geometry.

Thus, we have investigated a combined error metric, with one term dependent only on positions and one only on normals. Specifically, we take

$$\begin{aligned} E_{combined} &= \alpha \|c_{triangle} - c_{proxy}\|^2 + \\ &\quad (1-\alpha) A_{avg} \|n_{triangle} - n_{proxy}\|^2, \end{aligned}$$

where c and n represent the average positions and normals of triangles and proxies, A_{avg} is the average triangle area (included to ensure that the two terms are dimensionally compatible and the metric is scale invariant), and α is a user-selected parameter. In Figure 5.5 we show an example of remeshing a bull with $\alpha = 0$ and with $\alpha = 0.4$. Note that the proxies look more symmetric as we increase α, at the expense of a less-faithful reproduction of the original surface.



Bull planes ($\alpha = 0$) ($\alpha = 0.4$)

Figure 5.5: At left, we show a bull remeshed to 200 proxies with two planes simultaneously, one passing through the head and one passing through the body. In the center column we show results using the basic $L^{2,1}$ metric ($\alpha = 0$). At right, we show results when $\alpha = 0.4$. Note that the model becomes more symmetric, at the expense of geometric accuracy.

5.1.3 Results

We evaluate the algorithm described in the preceding sections using a number of well-known 3D meshes. Our goal is to produce simplified meshes that maintain and even enhance symmetries, while minimizing geometric approximation error. In the following subsections, we analyze the algorithm for both single and multiple planes of symmetry and present running times.

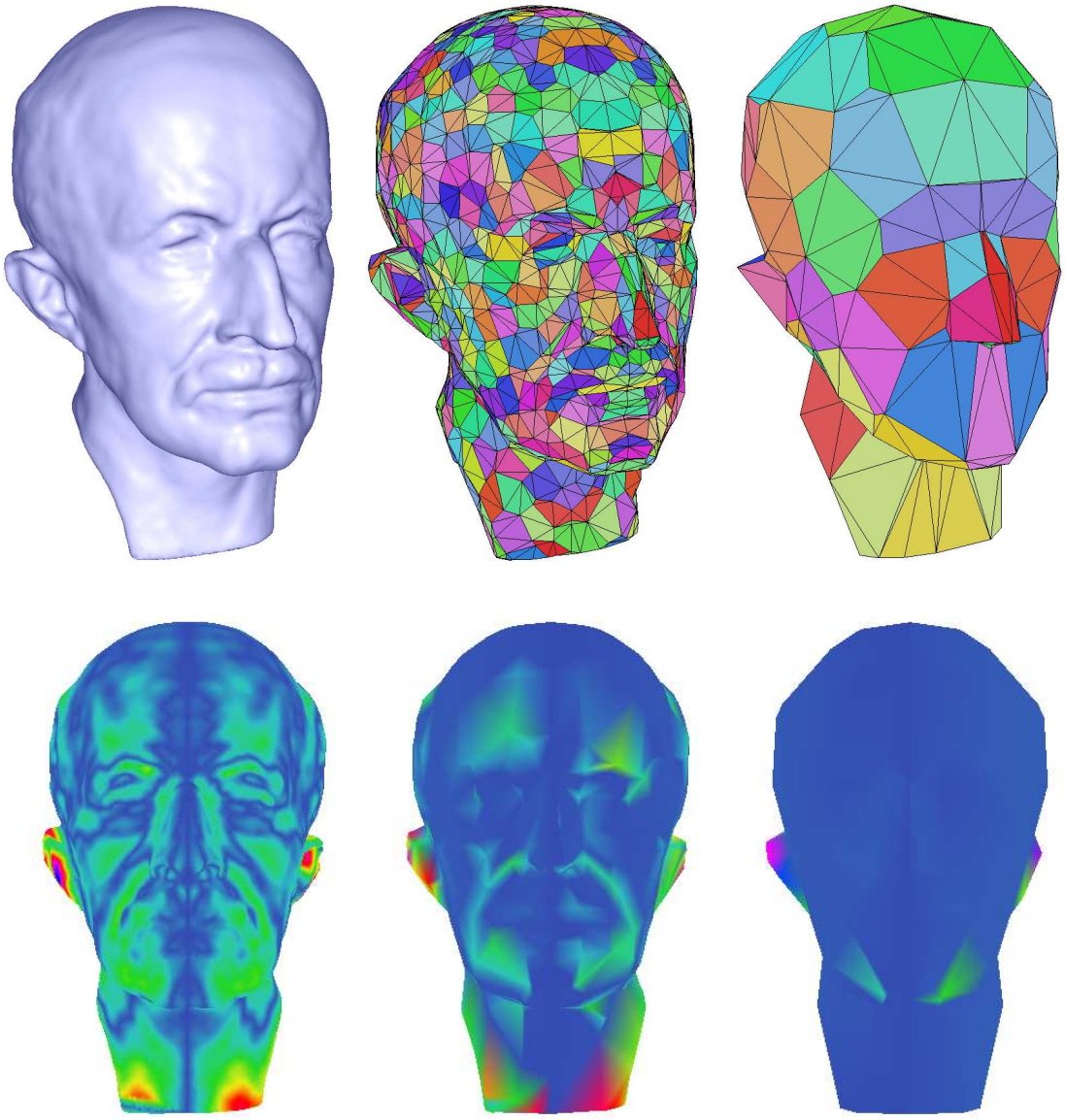
Approximate Symmetries A first example shows results for a mask (Figure 5.1): the original model (a) contains 62K faces and is only roughly symmetric. Figure 5.1(b) shows proxies for the model generated using the unmodified algorithm of Cohen-Steiner et al. [11]. Note that the proxies are not symmetric. Figure 5.1(c) shows the result of using our symmetry-aware algorithm: note that the proxies are now symmetric. Since

symmetric remeshing allows proxies to have multiple patches, we used 100 proxies in the basic method and 50 proxies in the symmetry-aware version to ensure a similar-quality tessellation. Note that our algorithm (e) produces a triangulation with more symmetry than the traditional method (d), as expected. Moreover, we find that it introduces very little geometric error in the shape approximation as compared to the original Cohen-Steiner algorithm — the RMSD increases by 4%, while the symmetry error (RMSD to reflection) decreases by 50%. This result suggests that more-symmetric topology can be provided for approximately symmetric models at very little cost.

Figure 5.6 demonstrates that increasing the amount of simplification results in meshes that are more symmetric. At top we show the Max Planck model simplified to 600 and 50 proxies, while at bottom we illustrate deviations from perfect symmetry (blue is most symmetric while green, yellow, and red indicate more asymmetry). This illustrates the property of our algorithm that it automatically captures symmetry if doing so is compatible with the current deformation error.

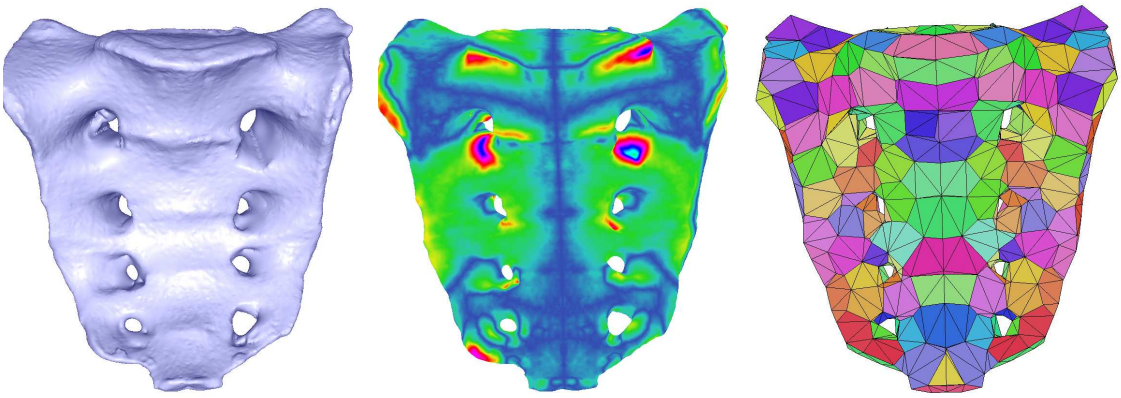
A third example is the sacrum in Figure 5.7 ($605K$ faces) which was remeshed using 200 proxies. This is model of a natural object that is close to symmetric yet not perfectly so, as shown in the color-coded visualization at center. In addition, this model was created from a set of $3D$ scans, meaning that high-frequency scanning noise caused further deviations from symmetry. At right we show our remeshed result, produced in just over eight minutes. Our method remains robust despite the high-frequency noise, the presence of holes and the large size of the mesh.

Multiple Symmetries Our first example of remeshing while considering multiple planes of symmetry is the bull (Figure 5.5). This model has separate planes of reflection indicating approximate symmetries of the head and the body. These planes were automatically



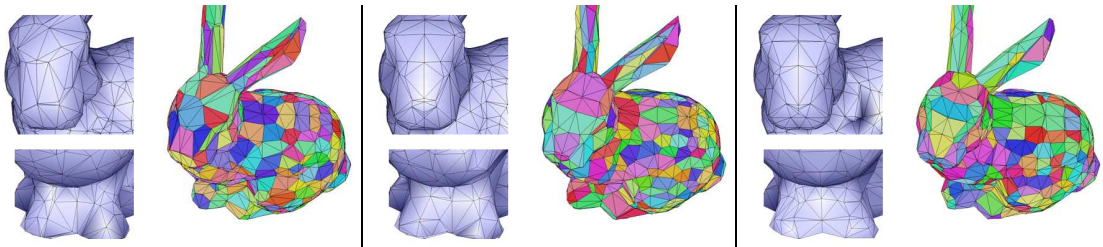
Original 600 50

Figure 5.6: Increasing the amount of geometric simplification (towards right) results in greater symmetry preservation. Images in the bottom row are colored to represent deviations from symmetry (how far each point is from the reflected surface) with blue indicating perfect symmetry.



Original Asymmetry Remeshed

Figure 5.7: 3*D* scan of a sacrum (605K polygons, shown at left) remeshed with 200 proxies (right). Note that the model is only approximately symmetric (middle).



2 symmetry planes 3 symmetry planes 4 symmetry planes

Figure 5.8: The bunny is shown here remeshed with 250 proxies, using 2, 3, or 4 planes of symmetry. Note that increasing the number of symmetries results in progressively more symmetric and more intuitive triangulations, while retaining plausible triangulations at the boundaries of symmetric regions.

extracted by choosing the top two principal planes of the model, as described in Section 3.3. Note that the remeshing results in symmetric triangulations for the head and the body with a reasonable (though not symmetric) triangulation at the neck.

Our second example (Figure 5.8) shows results for the Stanford bunny.

Recalling that the bunny has four major planes of symmetry (3.3), we show in Figure 5.8 the result of remeshing with respect to two, three, and four planes of symmetry. Note that remeshing with only three planes fails to capture the symmetry of the feet, while

considering only two planes (through the body and ears — the strongest planes resulting from the symmetry analysis) fails to triangulate the head symmetrically. In contrast, the result at right shows that we are able to take advantage of all four symmetry planes to produce an intuitive result with locally-symmetric triangulations and smooth transitions between the symmetric regions.

To produce these results, we segmented the input mesh according to the strongest symmetry at each point, then constrained the proxy flooding to only consider one candidate symmetry per triangle. The remeshing itself, however, was still run on the entire model, allowing smooth blends between symmetry regions to be computed automatically.

Computation Time We ran our experiments on a 3GHz PC with 1GB of RAM. As a preprocess, for every model we calculated a symmetry transform using a 64x64x64 grid, then ran the Iterative Symmetric Points algorithm (described in Section 3.3) to refine each of the extracted principal symmetries. While most models exhibit a single strong plane of symmetry, we have extracted up to four principal planes for the models considered in this section.

Our remeshing algorithm can take up to several minutes for large (i.e., 600k polygons) models. The most time-consuming phase is the distortion-minimizing flooding, which takes $O(m \cdot n \log n)$ time for m iterations on a mesh with n triangles. Note that this time is similar to what would be required for the unmodified remeshing algorithm. In comparison, the time required for anchor and secondary point placement and triangulation is much lower, ranging from a few seconds to half a minute if the model is simplified to have large patches. The total time for the clustering part of the algorithm can be seen in Table 5.1. Unless otherwise noted, all the models were remeshed with $\alpha = 0.002$.

Model	# Triangles	# Proxies	Time (s)
Mask	62K	50	62
Igea	130K	50	99
Teapot	8.5K	100	7
Bull	49K	200	72
Max Planck	98K	600	85
Sacrum	605K	200	491
Bunny	69K	30	49

Table 5.1: Timing results for the proxy-growing stage of our method, for various models and different numbers of proxies.

5.1.4 Discussion

Since human beings can recognize symmetry easily, retaining the symmetry of models during simplification is important. We have shown how to modify the existing remeshing technique proposed by [11] to directly incorporate symmetry without deforming the model too much. In fact, the approximation error introduced by our algorithm can be trivially bounded. The upper bound is simply the error of the unmodified algorithm with the same number of proxies, while the lower bound is the error of the original algorithm using a number of proxies equal to the number of our *patches*. For example, if we remesh using only one plane of symmetry (up to 2 patches per proxy) using K proxies, the error will be between the errors resulting from using the traditional method with K and $2K$ proxies.

Though we have demonstrated that our algorithm is able to take advantage of imperfect symmetries during remeshing, and can produce regions of symmetric tessellation joined by plausible triangulations for multiple symmetries, our method has limitations that suggest avenues for future work.

First, our approach to symmetric remeshing is limited by the fact that our final edge-flipping technique does not explicitly search for symmetry. While the method works

in cases where entire proxies are symmetric, if a proxy contains both symmetric and non-symmetric anchor and secondary points, the edge-flipping technique might adversely affect the triangulation in the symmetric areas. A further weakness of this technique is that running edge-flipping on patches separately, rather than on the entire mesh at once, may reduce the quality of the final triangulation of the model. Thus, symmetry-preserving edge-flipping is a topic for future work.

Another limitation of the approach is our non-reflexive method of dealing with symmetry. If for example, one side of a model is reflected only once while the other side is reflected twice, the number of patches for a proxy in that section depends on the starting triangle. This is merely a manifestation of a problem with Lloyd's algorithm - it doesn't converge to a minimum.

There are other limitations of our technique inherited from the underlying Variational Shape Approximation algorithm. For example, the patch flooding requires a computationally expensive iteration, and the final triangulation algorithm is constrained to use original vertices of the model and is not error-driven. Moreover, the algorithm is optimized for the case of significantly fewer output than input polygons, thus making our symmetry-enabled variant inappropriate for symmetric remeshing that preserves all original detail.

A further limitation of this work is that the use of symmetry is necessarily coupled to the error metric. Although we believe that this coupling is theoretically sound and intuitively understandable, it may be inappropriate for some applications. Therefore, a future extension of this work may consider explicitly allowing for greater deformation if doing so will strengthen symmetry.

Finally, we use planar proxies as the foundation of our algorithm, but our algorithm should work for general proxies. An possible avenue of future work is to implement our algorithm using the generalized proxy definitions of [51] or [52].

5.2 Segmentation

Assume you are trying to color a 3D model of a car. You want the wheels to be colored a dull black, the body of the car to be a bright blue, the interior to be leather brown, and the hubcaps to shine. The problem is that the model is a bunch of triangles floating in space. How do you label these triangles in order to be able to color them correctly? This general problem of segmentation is a key step in a wide range of tasks including collision detection, texture mapping, compression, and partial-object matching.

While for some applications (e.g. collision detection) it might be enough to decompose an object into generic components such as convex regions, for many other computer graphics applications, a key step of the algorithm is identifying an *intuitive* segmentation of the model. Consider that if an animal is described as “looks like a giraffe, but with a short neck”, or that a car “has a speaker system on top and doesn’t have any wheels”, you get an immediate picture in your mind. Humans readily identify objects by their component parts, and so any application for 3D models that has goal of assisting humans with some form of shape analysis or shape processing (e.g matching, metamorphosis, modeling by parts) would benefit greatly from intuitive segmentation.

In computer graphics, there have been various methods proposed for computing such a segmentation, but many of these rely on local surface and volumetric information. In this section we will review those methods, and show how to include symmetry information to improve the quality of those algorithms. As we explain below, we believe that the

non-local nature of symmetry allows its use to generate more desirable segmentations for many applications.

5.2.1 Previous Work

Convex Segmentation Convex decomposition of 3D models has long known to be a complex computational geometry problem. For example, while there are many relatively simple sub-quadratic solutions for computing triangulations in 2D (e.g. [17, 10, 8]), in 3D there are polygonal meshes that cannot be triangulated (i.e. tetrahedralized) without the addition of Steiner points. In fact deciding if a mesh can be triangulated without adding Steiner points is provably $NP-Complete$ [41]. If Steiner points are allowed, then there are a number of methods available, such as [7] or [9].

Unfortunately, a convex segmentation is not always intuitive for humans. If a 3D model contains small concavities, the solutions produced by the above algorithms would necessarily contain many small convex components. In addition, large planar surfaces may get unnecessarily partitioned, or partitioned differently based on small perturbations of the surface. Instead, for shape analysis type applications, a segmentation algorithm that captures some manner of human expectation is more likely to be useful.

Intuitive Segmentation In the previous decade, there have been a number of methods suggested for intuitive segmentation of 3D models.

Li et al. [30] segment 3D models using a set of generalized cylinders called “sweep paths”. In their method, a polygonal model is first converted into a skeleton using edge contractions. Skeleton edges are converted into a single skeleton-tree, and an ordered sweep of space is created based on this tree. Thus, the entire model is defined as the cross-section of a walk along the branches of the skeleton-tree. The model is segmented by

creating boundaries where the cross-section changes rapidly or the tree branches. While the approach is generally robust to noise, a downside of this method is that smoothing of the model can create situations where the cross-section changes slowly enough so that segmentation does not happen.

Mortara et al. [36] use a method similar to sweep paths named Tailor. Given a point p in a 3D model M, they characterize the surface around p using the intersection of M with a series of concentric spheres centered p. Depending on the nature of the intersection, the local surface is described as a disc, a tube, or branching. Their segmentation algorithm uses this description to split a model into "limbs" and "body". Each limb is then grown until some stop criteria is reached, creating a smooth segmentation. Further segmentation may be achieved using smaller radii.

Shlafman et al. [45] use K-means clustering to segment polygonal meshes. The weighting scheme they use to cluster the faces of the mesh is based on geodesic distance, convexity and curvature. They also suggest decomposing into various amounts of clusters in order to find an optimal segmentation. A general problem with this method is that the boundaries between the components tend to be jagged and susceptible to noise.

Katz and Tal [23] use a technique they call *fuzzy segmentation* in order to obtain smoother boundaries. In their method, they first split the model into meaningful components while keeping the boundaries between the components fuzzy by allowing triangles to belong to multiple components. The metric they use to compute the distance between triangles is similar to the one used by [45], and includes the geodesic distance between centers of adjacent triangles and the the angle between their normals. Extra weight is given to concave angles, because convex shapes make better candidates for boundaries. Once the fuzzy components are computed, their technique resolves one boundary at a time using a graph-cut algorithm to ensure that the split is smooth and follows the

contours of the model. A refinement to this method is proposed in [31], where the coarse segmentation is based on first finding prominent feature points on a pose insensitive representation.

5.2.2 Symmetry

As noted in chapter 2, there are many objects that while not perfectly symmetry themselves, are composed of symmetric components. For example, a chair might only have left right-symmetry, but its legs will contain front-back (and top-bottom) symmetry as well. An engine block may not contain any global symmetries, but it's component gears and tubes need to be symmetric in order to function. More importantly, each component has **its own unique set of symmetries**. We propose that a meaningful segmentation of a 3D model may be computed by analyzing the partial symmetries (as defined in section 3.3) inherent in the model and clustering sections of the model based on which symmetries they share.

Given a model in the form of a polygonal mesh, our method begins by computing the PRST and obtaining a list $P_{j=0}^{K}$ of the principal symmetries. Then, for every polygon T_i in the mesh, we compute the *local support* $L_{i,j}$ for each plane P_j in the list. Specifically, for every triangle, $L_{i,j}$ is the average value in the PRST of a point in T_i reflected by P_j. An example of a 2D model with local support may be seen in Figure 5.9.

At the end of this step, each triangle T_i is represented by a feature vector of K numbers defining how well T_i is reflected by each of the principal symmetries. An example of this may be found in Figure 5.10. We now follow the same strategy suggested by Katz and Tal and run fuzzy clustering on these feature vectors, followed by a graph-cut algorithm to obtain the final boundaries. Intuitively, this method clusters faces that are symmetric with

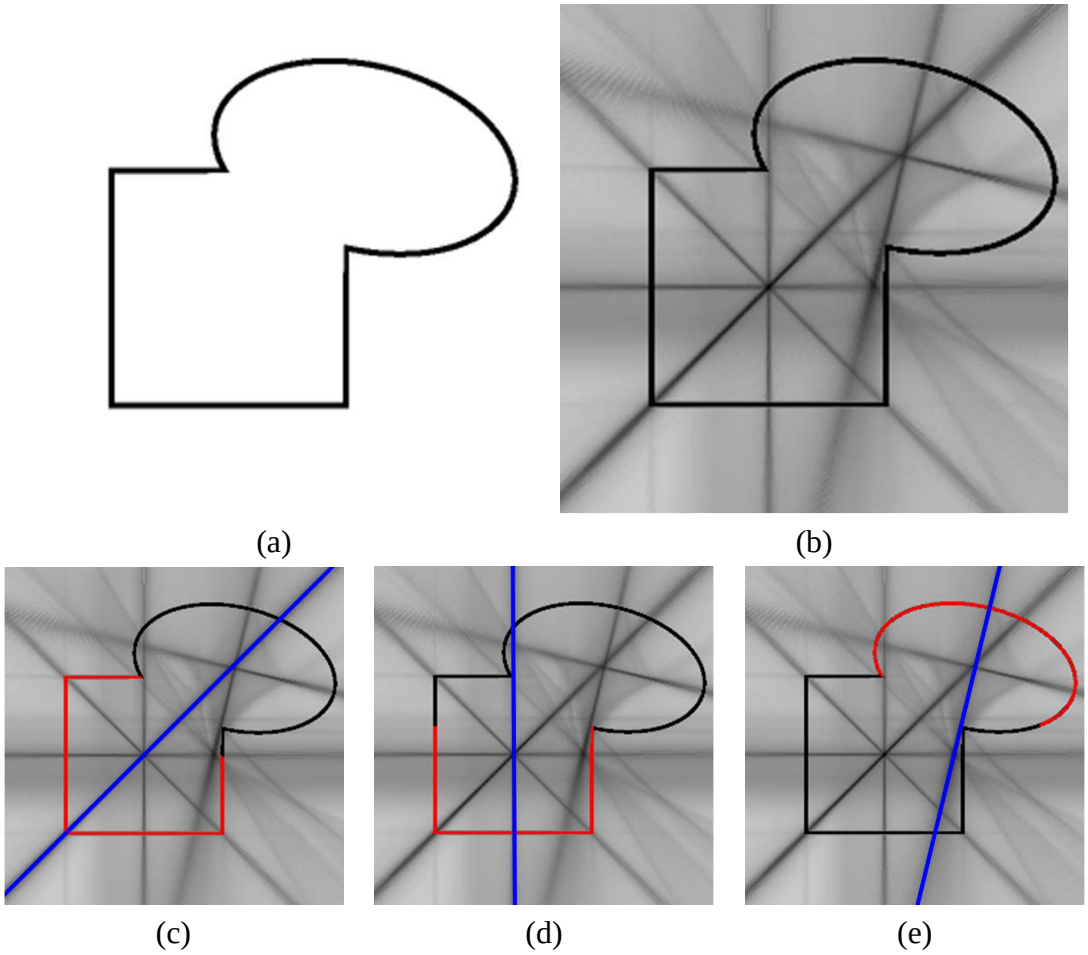
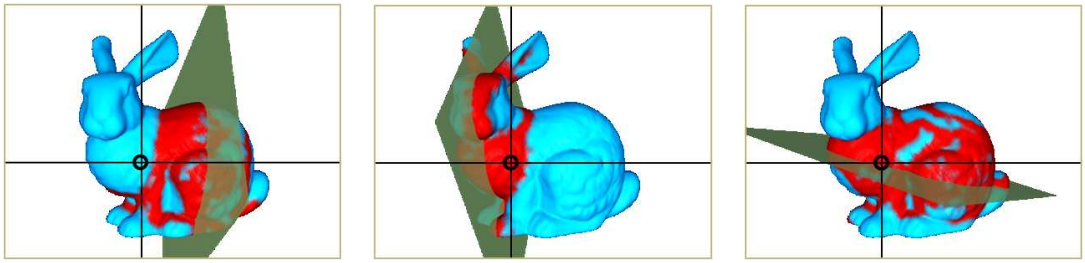


Figure 5.9: Example of local support: Given a 2D model (a), we compute the symmetry transforms on it (b), and extract principal symmetries. Then, for every symmetry we find which portion of the model it reflects well (c,d,e). In the example shown, the square is reflected by the first two planes of symmetry (c,d), while the ellipse is reflected the the third (e). This leads to an intuitive segmentation of the model (square vs. ellipse). In this visualization, principal planes are shown in blue and the support of those planes is shown in red.



Support = 0.1 Support = 0.5 Support = 0.9

Figure 5.10: In this example, the point on the bunny *supports* the three planes shown to varying degrees. Concatenating the support values creates a feature-vector describing how symmetric this point is with respect to each plane of symmetry. The feature-vector for this point is $(0.1, 0.5, 0.9...)$. In this visualization points are colored based on how symmetric they are with respect to the plane shown, with red meaning symmetric and blue meaning non-symmetric.

respect to the same distinct set of planar symmetries, and will thus segment the model into usable components.

A limitation of clustering using the PRST is that the major symmetries dominate the segmentation. The symmetry of a small part of the model might not show up in the transform at all. In order to support segmentation into ever finer components, we modify the above algorithm by using a hierarchy of splits. For each split, we perform k-means clustering (with $k = 2$) to establish an initial segmentation. Then, for every segment, the discrete PRST is recomputed and its local maxima extracted for use in the next level of segmentation. Segmentation is terminated at a user-supplied depth, or when the only planes of local maxima reflect either more than 90% or less than 10% of the surface onto itself.

The result of this process is a segmentation tree, with the property that lower levels in the tree capture increasingly local symmetries, hence allowing strong symmetries of even small parts to influence the segmentation.

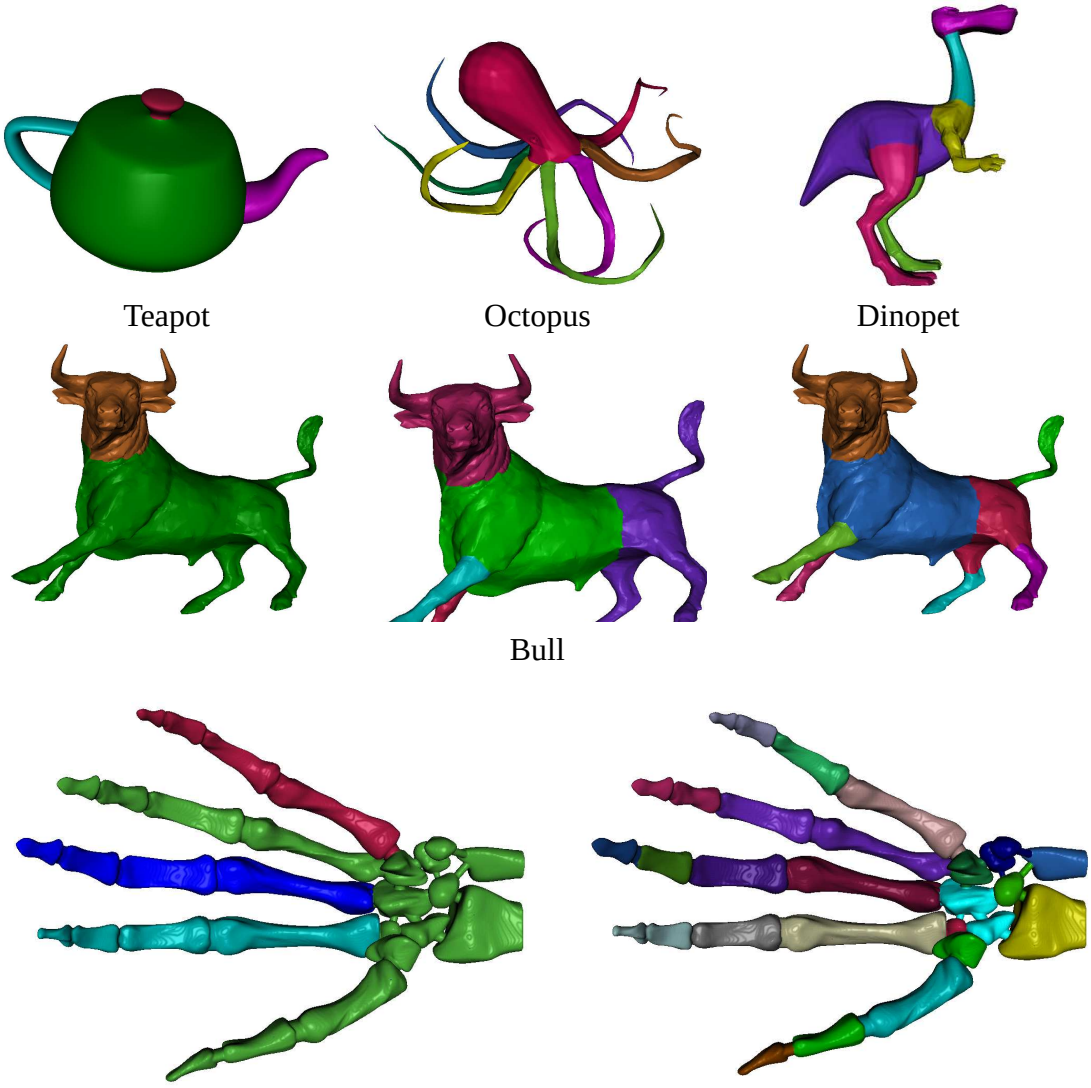


Figure 5.11: These images show segmentations of a range of models. For the bull we show segmentation into $2, 4$, and 8 segments. The skeletal hand is shown segmented into 4 and 18 parts.

5.2.3 Results

Figure 5.11 shows some examples of the segmentation produced by our method. Note that for the Teapot the strongest planes of symmetry pass through the body of the pot. So, the handle, spout, and top are removed precisely because they are *not* symmetric

with respect to those planes — i.e., the body of the pot is removed from the smaller parts, rather than vice-versa. For the Octopus and Skeletal Hand models, local symmetries of parts are important for obtaining the segmentation shown. Finally, a weakness of our scheme can be seen in the segmentation of the legs of the Dinopet and Bull. Because we use a min-cut to smooth our initial guess, the final segmentation will seek a shorter cut, and thus avoid the upper sections of the thigh. Integrating symmetry information into the min-cut algorithm is a topic for future work.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

In this thesis we have defined the Symmetry Transform and explored a number of general methods for using symmetry information extracted from this transform to improve a wide range of analysis and editing applications. In particular, our contributions are four-fold.

First, we introduce the point symmetry transform and the planar reflective symmetry transform, which measure the symmetry of an object with respect to all points in space, planes through its bounding volume respectively. We show how to compute them efficiently, using convolution methods and provide a fast Monte-Carlo method for computing the PRST when dealing with surface meshes. In empirical evaluations, we explore the properties of these transforms, and demonstrate that it is stable under small perturbations, and intuitively responds to rotations and scale.

Second, we provide a method for extracting local maxima from the discretized transformation and computing their exact position. We also show how these maxima provide a useful subset of planes for describing the principle symmetries of a model.

Third, we investigate the utility of the symmetry transform for several geometric analysis applications. In particular, we propose that the *center of symmetry* and *principal symmetry axes* are useful for aligning 3D objects in a common coordinate frame. We also show that the reflective symmetry transform can be used for registering 3D range scans into a common coordinate system, matching 3D polygonal models of the same class, and that good viewpoints for visualization of meshes may be found by minimizing symmetry.

Finally, we show that the symmetry transform is useful for editing applications as well. In particular, we show how symmetry may be used to find meaningful parts in 3D models, and how to remesh models in a manner that the main symmetries are explicitly maintained.

6.2 Future Work

The investigation of the PRST presented in this thesis is a first step. Our implementation has several limitations, and there are many avenues for future research.

6.2.1 Symmetry

First, we have investigated only the transform that maps a 3D object to its point and planar reflective and symmetries. While these types of symmetry are perhaps the most prevalent in real-world objects, and thus makes a good starting point, it is possible to consider other types of symmetry in future work. For example, one might define a rotational symmetry transform, in which a measure of symmetry is computed for every possible rotation around every possible axis (5D), or a translation by any offset (3D).

Second, our transform measures symmetries of an entire object. However, for objects containing multiple symmetric parts a useful investigation would be to understand how

symmetries can be detected at multiple scales, corresponding to different sized regions of local support [32]. While we have shown an automatic segmentation algorithm that extract symmetries of large parts of the model hierarchically, interesting local symmetries may "be drowned" and not found during this process. A multi-resolution scheme, where the PRST is computed on local sections of the model would be interesting to investigate.

Third, our investigation include only a small subset of the analysis and editing applications available. Further applications that might benefit from symmetry include completion, compression, texture mapping, and symmetrization.

6.2.2 Inversion

Since the PRST is a 3D $\rightarrow$ 3D mapping, we raise the question of whether the transform is invertible. While at first glance the transform is composed of operations that cause irrecoverable loss of information, precluding inversion, we believe that there are strong constraints provided by, for example, the requirement that the model be bounded (i.e. that the function is zero at the boundary) that make inversion possible. Using these constraints we can already show that the transform is invertible in the 1D and 2D cases, and we believe we can extend this to 3D. The inversion is currently sensitive to noise however, so further research is necessary to determine if additional constraints or stronger variational relations might make our method practical. We hypothesize that the ability to invert the transform will lead to applications in a variety of domains, with the ability to not only analyze but also synthesize symmetries.

Bibliography

[1] S. Abbasi and F. Mokhtarian. Automatic view selection in multi-view object recognition. In *Proc. ICPR*, volume 1, page 1013, 2000.

[2] M. Ankerst, G. Kastenmller, H. peter Kriegel, and T. Seidl. 3d shape histograms for similarity search and classification in spatial databases. pages 207–226. Springer, 1999.

[3] M. Atallah. On symmetry detection. *IEEE Trans. on Computers*, 34:663–666, 1985.

[4] P. J. Besl and N. D. McKay. A method for registration of 3-D shapes. *IEEE Trans. PAMI*, 14(2):239–256, 1992.

[5] V. Blanz, M. Tarr, H. Buelthoff, and T. Vetter. What object attributes determine canonical views. *Perception*, 28, 1999.

[6] H. Blum. A transformation for extracting new descriptors of shape. In W. Whaten-Dunn, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, 1967.

[7] B. Chazelle. An optimal convex hull algorithm and new results on cuttings (extended abstract). In *FOCS*, pages 29–38, 1991.

[8] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, 1991.

[9] B. Chazelle, D. P. Dobkin, N. Shouraboura, and A. Tal. Strategies for polyhedral surface decomposition: an experimental study. In *SCG '95: Proceedings of the eleventh annual symposium on Computational geometry*, pages 297–305, New York, NY, USA, 1995. ACM.

[10] K. L. Clarkson, R. E. Tarjan, and C. J. V. Wyk. A fast las vegas algorithm for triangulating a simple polygon. In *SCG '88: Proceedings of the fourth annual symposium on Computational geometry*, pages 18–22, New York, NY, USA, 1988. ACM.

[11] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. *ACM Transactions on Graphics (Proc. Siggraph)*, 23(3):905–914, 2004.

[12] R. Duda, P. Hart, and D. Stork. *Pattern Classification, Second Edition*. John Wiley & Sons, New York, 2001.

[13] M. Enquist and A. Arak. Symmetry, beauty and evolution. *Nature*, 372:192–172, 1994.

[14] R. W. Ferguson. Modeling orientation effects in symmetry detection: The role of visual structure. In *Proc. Conf. Cognitive Science Society*, 2000.

[15] H. Fu, D. Cohen-Or, G. Dror, and A. Sheffer. Upright orientation of man-made objects. *ACM Trans. Graph*, 27(3), 2008.

[16] T. Funkhouser, P. Min, M. Kazhdan, J. Chen, A. Halderman, D. Dobkin, and D. Jacobs. A search engine for 3D models. *ACM Trans. Graph.*, 22(1):83–105, 2003.

[17] M. Garcy, D. Johnson, F. Preparata, and R. Tarjan. Triangulating a simple polygon. volume 7, pages 175–179, 1978.

[18] M. Garland and P. S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 263–269, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[19] H. Hoppe. Progressive meshes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108, New York, NY, USA, 1996. ACM Press.

[20] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 19–26, New York, NY, USA, 1993. ACM Press.

[21] B. Horn. Extended Gaussian images. *Proc. of the IEEE*, 72(12):1671–1686, December 1984.

[22] T. Kamada and S. Kawai. A simple method for computing general position in displaying three-dimensional objects. *Comput. Vision Graph. Image Process.*, 41(1):43–56, 1988.

[23] S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *Proceedings of ACM SIGGRAPH*, 22(3):954–961, 2003.

[24] M. Kazhdan, B. Chazelle, D. Dobkin, T. Funkhouser, and S. Rusinkiewicz. A reflective symmetry descriptor for 3D models. *Algorithmica*, 38(1), Oct. 2003.

[25] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Rotation invariant spherical harmonic representation of 3D shape descriptors. In *Symposium on Geometry Processing*, June 2003.

[26] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Symmetry descriptors and 3D shape matching. In *Proc. Symposium on Geometry Processing*, 2004.

[27] R. Klein, G. Liebich, and W. Strasser. Mesh reduction with error control. In *VIS '96: Proceedings of the 7th conference on Visualization '96*, pages 311–318, Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[28] C. H. Lee, A. Varshney, and D. W. Jacobs. Mesh saliency. *Proceedings of ACM SIGGRAPH*, 24(3):659–666, 2005.

[29] J. Lee, B. Moghaddam, H. Pfister, and R. Machiraju. Finding optimal views for 3d face shape modeling. In *FGR*, pages 31–36. IEEE Computer Society, 2004.

[30] X. Li, T. Toon, T. Tan, and Z. Huang. Decomposing polygon meshes for interactive applications. In *Proceedings of the 2001 Symposium on Interactive 3D graphics*, pages 35–42, 2001.

[31] S. K. G. Liefman and A. Tal. Mesh segmentation using feature point and core extraction. *The Visual Computer*, 21:865–875, 2005.

[32] R. Manmatha and H. Sawhney. Finding symmetry in intensity images. Technical Report UM-CS-1997-007, University of Massachusetts, Jan. 1997.

[33] A. Martinet, C. Soler, N. Holzschuch, and F. Sillion. Accurately detecting symmetries of 3D shapes. Technical Report RR-5692, INRIA, September 2005.

[34] P. Minovic, S. Ishikawa, and K. Kato. Symmetry identification of a 3D object represented by octree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):507–514, May 1993.

[35] N. J. Mitra, L. Guibas, and M. Pauly. Partial and approximate symmetry detection for 3d geometry. In *ACM Transactions on Graphics*, volume 25, pages 560–568, 2006.

[36] M. Mortara, G. Patanè, and M. Spagnuolo. From geometric to semantic human body models. *Computers & Graphics*, 30(2):185–196, 2006.

[37] S. Palmer, E. Rosch, and P. Chase. Canonical perspective and the perception of objects. *Attention and Performance IX*, pages 135 – 151, 1981.

[38] J. Podolak, A. Golovinskiy, and S. Rusinkiewicz. Symmetry-enhanced remeshing of surfaces. In *Symposium on Geometry Processing*, July 2007.

[39] J. Podolak, P. Shilane, A. Golovinskiy, S. Rusinkiewicz, and T. Funkhouser. A planar-reflective symmetry transform for 3D shapes. *ACM Transactions on Graphics (Proc. Siggraph)*, 25(3), July 2006.

[40] O. R., F. T., C. B., and D. Dobkin. Matching 3d models with shape distributions. *Shape Modeling and Applications*, pages 154–166, May 2001.

[41] J. Ruppert and R. Seidel. On the difficulty of tetrahedralizing 3-dimensional non-convex polyhedra. In *SCG '89: Proceedings of the fifth annual symposium on Computational geometry*, pages 380–392, New York, NY, USA, 1989. ACM.

[42] M. I. Shah and D. C. Sorensen. A symmetry preserving singular value decomposition. *SIAM Journal of Matrix Analysis and it's Application,* October 2005.

[43] Y. Shan, B. Matei, H. S. Sawhney, R. Kumar, D. Huber, and M. Hebert. Linear model hashing and batch ransac for rapid and accurate object recognition. *IEEE International Conference on Computer Vision and Pattern Recognition*, 2004.

[44] P. Shilane, P. Min, M. Kazhdan, and T. Funkhouser. The Princeton Shape Benchmark. In *Proc. Shape Modeling International*, 2004.

[45] S. Shlafman, A. Tal, and S. Katz. Metamorphosis of polyhedral surfaces using decomposition. In *Computer Graphics forum*, 2002.

[46] C. Sun and J. Sherrah. 3D symmetry detection using the extended Gaussian image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(2):164–168, February 1997.

[47] S. Thrun and B. Wegbreit. Shape from symmetry. In *Proceedings of the International Conference on Computer Vision (ICCV)*, Bejing, China, 2005. IEEE.

[48] G. Turk. Re-tiling polygonal surfaces. *SIGGRAPH Comput. Graph.*, 26(2):55–64, 1992.

[49] P.-P. Vázquez, M. Feixas, M. Sbert, and W. Heidrich. Viewpoint selection using viewpoint entropy. In *VMV '01: Proceedings of the Vision Modeling and Visualization Conference 2001*, pages 273–280. Aka GmbH, 2001.

[50] J. D. Wolter, T. C. Woo, and R. A. Volz. Optimal algorithms for symmetry detection in two and three dimensions. *The Visual Computer*, 1:37–48, 1985.

[51] J. Wu and L. Kobbelt. Structure recovery via hybrid variational surface approximation. In *EUROGRAPHICS*, volume 24, pages 277–284, 2005.

[52] D.-M. Yan, Y. Liu, and W. Wang. Quadric surface extraction by variational shape approximation. *GMP*, pages 73–86, 2006.

[53] H. Zabrodsky, S. Peleg, and D. Avnir. Completion of occluded shapes using symmetry. In *Proc. CVPR*, pages 678–679, 1993.

[54] H. Zabrodsky, S. Peleg, and D. Avnir. Symmetry as a continuous feature. *Trans. PAMI*, 17(12):1154–1166, 1995.

[55] J. Zhang and K. Huebner. Using symmetry as a feature in panoramic images for mobile robot applications. In *Proc. Robotik*, volume 1679 of *VDI-Berichte*, pages 263–268, 2002.