FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Co-Analyse und Parametrisierung von 3D-Formkollektionen für Form-Synthese

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Kurt Leimer
Matrikelnummer 0825842

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Projektass.(FWF) Dr.techn. Dipl.-Mediensys.wiss. Przemyslaw Musialski

Wien, 15. Jänner 2016

Michael Wimmer

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Co-Analysis and Parameterization of 3D Shape Collections for Shape Synthesis

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Kurt Leimer
Registration Number 0825842

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Projektass.(FWF) Dr.techn. Dipl.-Mediensys.wiss. Przemyslaw Musialski

Vienna, 15th January, 2016

Michael Wimmer

Erklärung zur Verfassung der Arbeit

Kurt Leimer
Brigittaplatz 1/4/2, 1200 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 15. Jänner 2016

Danksagung

Ich möchte meinen Betreuern Przemyslaw Musialski und Michael Wimmer danken, deren Unterstützung diese Diplomarbeit möglich gemacht haben. Ich danke außerdem Michael Birsak, der mir bei organisatorischen Fragen oft Rat gegeben hat. Aber vor allem möchte ich mich bei meiner Familie bedanken, die mich während des gesamten Studiums so großartig unterstützt hat.

Acknowledgements

I would like to thank my advisors Przemyslaw Musialski and Michael Wimmer whose support made this Master Thesis possible. I would also like to thank Michael Birsak who often gave me advice when I had organizational questions. But most of all I would like to thank my family who greatly supported me during the entire course of my studies.

Kurzfassung

Dank täglich wachsender Online-Datenbanken für 3D Models eröffnen sich sowohl für erfahrene, als auch für unerfahrene Benutzer neue Möglichkeiten zur Erstellung von neuen Inhalten. Eine solche Möglichkeit ist die Erstellung von neuen Formen durch das Kombinieren von Teilen bereits existierender Formen. Zu den Vorteilen dieser Methode der Formsynthese gehören einerseits, dass sie weniger Zeit beansprucht als herkömmliche Modeling Methoden, andererseits, dass sie ebenfalls von unerfahrenen Benutzern verwendet werden kann. Diese Diplomarbeit präsentiert ein Grundgerüst für diesen Typ der Formsynthese, das aus 4 Schritten besteht und eine neue Art der Parametrisierung und Erkundung von Formkollektionen beinhaltet. Mithilfe einer modularen und erweiterbaren Vorgehensweise werden im Co-Analyse-Schritt Teile von Formen basierend auf ihrer Funktion zu Kategorien gruppiert und dadurch eine Korrespondenz zwischen Teilen verschiedener Formen erstellt. Indem Relationen zwischen Teilen und die Art und Weise, wie deren räumliche Verteilungen innerhalb der Kollektion variieren, anaylsiert werden, kann im Parametrisierungs-Schritt eine kleine Anzahl an Parametern gefunden werden. Beginnend mit einer existierenden Form als Startpunkt können diese Parameter im Erkundungs-Schritt dazu genutzt werden, die Kollektion zu durchsuchen, entweder indem die Parameter direkt geändert werden, oder durch Interaktion mit der Form selbst. Schließlich können im Formsynthese-Schritt neue Formen erstellt werden, indem Teile der Start-Form durch korrespondierende Teile von anderen Formen ausgetauscht werden, die während der Erkundung gefunden wurden.

Abstract

With online model repositories growing larger every day, both experienced and inexperienced modelers are presented with new possibilities for content creation. One such possibility is the creation of new shapes by combining parts of already existing shapes. The advantages of this shape synthesis method are that it takes less time than traditional modeling approaches and that it can be used even by inexperienced users. This thesis introduces a framework for this type of shape synthesis that consists of four stages, incorporating a new way for parameterization and exploration of shape collections. Using a modular and extensible approach, the co-analysis stage groups parts of shapes into categories based on their function, creating a correspondence between parts of different shapes. By analyzing relations between pairs of parts and how their spatial arrangements vary across the collection, a small number of parameters is found in the parameterization stage. Starting with an initial shape, these parameters can then be used to browse the collection in the exploration stage, either by altering the parameters directly or by interacting with the shape itself. Finally, in the synthesis stage a new shape can be created by exchanging parts of the initial shape with corresponding parts of the shapes found during the exploration.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

One of the major bottlenecks in the field of computer graphics is the one of content creation, particularly the creation of 3D models. High-quality 3D models are a necessity in many industries, such as entertainment, computer-aided design, urban planning, the manufacturing of physical objects, as well as medical or scientific visualizations and simulations. These models can come from a variety of sources. Manually creating a 3D model not only requires a lot of artistic, but also technical skills. Traditional modeling applications often contain hundreds of different tools that all take significant effort to learn how to use properly. For inexperienced users this means that a large time investment is necessary before even simple 3D models can be created. Furthermore, even experienced users need to spend a lot of time in order to create high-quality models. Since the amount of detail that can be rendered and displayed in real time increases every year thanks to the frequent improvements in technology and algorithms, there are also higher expectations when it comes to the quality and detail of 3D models, making the creation process even more time consuming. As an example, recreating the Notre Dame cathedral for the video game Assassin's Creed Unity took a total of 5000 man-hours [GG].

Another way to create 3D models is by scanning real-life objects, but this requires special equipment and often produces 3D models that need additional manual cleanup to satisfy a high level of quality. Naturally, this also restricts the resulting models to shapes that already exist in the real world, making it less useful when the user has a concrete vision of the desired shape. Procedural generation, that is the automatic generation of content using algorithms that often incorporate randomness and statistical methods, has also become more prevalent in recent years. While this method can create a large variety of shapes in a short time, developing the necessary algorithms can be difficult, as can creating a shape with specific properties since the user usually does not have precise control over the end result.

However, with online repositories for 3D models like 3D Warehouse [Tri] becoming more prevalent and growing ever larger, new possibilities have opened up for both experienced and inexperienced users alike. The availability of large collections of shapes

Figure 1.1: The availability of large model collections from online repositories like 3D Warehouse has opened up new possibilities for both experienced and inexperienced users, such as shape synthesis by combining parts of already existing shapes.

(a small example of which can be seen in Figure 1.1) makes it possible to synthesize new shapes by combining different parts from already existing shapes. This form of shape synthesis can be both easy to learn and use, enabling the user to create a large variety of different shapes in a short time span. But naturally this comes with its own set of unique challenges that need to be overcome, making it an interesting field of research.

1.1 Problem Statement

There are multiple questions to consider when designing a framework for shape synthesis from shape collections. First of all, since we would like to exchange parts between shapes, it is necessary to define what constitutes a *part*. In their survey about structure, Mitra et al. define a part of a shape as "a logical entity of semantic significance that controls the appearance of part geometry" [MWZ+13]. This means that the geometry of a part is determined by its meaning or function, and not the other way around. To give an example, the parts of an airplane might consist of a fuselage, wings, stabilizers, engines and wheels, each of which are designed to fulfill a specific function.

However, shapes are often represented by low-level geometric primitives such as points or triangles. They may be connected to form a single surface, having no clear boundaries between individual parts. On the other side of the spectrum, a shape may consist of many small disconnected segments that should ideally be grouped together to form a single part. Furthermore, shapes taken from online repositories can vary greatly in their quality and often have no clear distinction between their parts. The question of how to segment a shape into parts automatically is a common problem in the field, one which has seen much research [SM00, KT03, GF08]. The level of detail with which a shape is represented also matters when defining parts. Segmenting a shape into three or four parts might suffice when simply populating an environment scene with objects. A more

technical view of a shape on the other hand might require small items such as screws and wires to be designated as separate parts. Thus even manual segmentations of shapes can vary greatly depending on the application.

Next, there is the problem of *part correspondence*, or identifying which parts can be reasonably exchanged for other parts. Basing such a correspondence purely on geometric similarity is a possibility, but even within a specific family there can be much variation in the geometry of the shapes. High-level analysis of shapes and the structural similarities within their families may be a better solution to the problem and has seen much research in recent years [MWZ+13].

The next question concerns the topic of *exploration* – how to choose from the large list of parts to create a new shape. If the shape collection is very large, it can be very time consuming to browse the collection in an attempt to find the specific kind of part one is looking for. Existing solutions to this problem show that there is generally a trade-off between control and simplicity. Some methods automatically generate new shapes, for example by means of a genetic algorithm [XZCOC12], giving the user no control over the results, but also requiring no interaction at all. Others may suggest parts during the synthesis process using machine-learning algorithms [CKGK11, KCKK12] or based on some measure of similarity [FKS+04, KJS07].

Another option is given by parameterizing the shape collection, that is to find a possibly small number of parameters that the user can alter to find specific shapes or parts in the collection. The trade-off between control and simplicity exists here as well. It is certainly possible to reduce the number of parameters to a very small number, for example by embedding the shapes into a two-dimensional space based on their similarity [AKZM14]. Exploring the collection using just two parameters is simple, but the drawback is that it can be difficult to find specific shapes since the only guarantee is that similar shapes are close to each other, yet the way in which altering the parameters changes the shape is not immediately apparent.

On the other side of the control-simplicity spectrum, it is also possible to provide the user with more direct control, for example by defining different relations such as angle or distance between pairs of parts [FAvK+14]. This lets the user explore the collection by altering these relations, giving a more concrete idea of the resulting shapes. The disadvantage is of course that this results in a large number of parameters, some of which might not even contribute much to the variation between shapes. Finding a good middle ground between control and simplicity can be difficult and is one of the major challenges regarding these kinds of methods.

Finally, there is the problem of *part alignment*. Shapes can come in a variety of different sizes and orientations, especially when taking them from an online repository. To be able to exchange parts between different shapes, it is necessary to align them so the resulting shape looks natural. Even scaling the parts to be of the same size is not always trivial – parts of shapes may be physically attached to other parts, making the size of the contact area equally important as the total size of the part to ensure a natural transition between the connected parts. Orientation is also a problem. Many existing methods assume the shapes to be in upright position or even globally aligned to make

the alignment process easier. Of course this is not always guaranteed for shapes taken from an online repository, possibly requiring manual rectification before existing shape synthesis methods can be used.

1.2 Contribution and Overview

The main contribution of this thesis is two-fold. First, we introduce a new method of parameterizing and exploring shape collections using the relations between parts of the shapes. This provides a possible solution to the question of how to explore the shape collection in an intuitive and efficient way. By analyzing the spatial arrangements between types of parts and how they vary across the collection, it is possible to find a small number of parameters that allow the user to browse the collection in an intuitive manner. Our main inspiration is the approach by Fish et al. [FAvK+14], but instead of using each possible type of relation as a parameter, we attempt to find correlations between these relation types to reduce the size of the parameter space. The exploration of the shape collection is then done by picking a shape to start with, selecting a pair of parts and changing the associated parameters either directly or by interacting with the shape. An important aspect of this exploration process is the coupling between the parameters and the visual representation of the parameter changes.

The second contribution is a shape synthesis framework that allows the user to create new shapes by combining parts from existing shapes. This framework deals with the other problems mentioned in Section 1.1, by incorporating existing algorithms in addition to our proposed methods of parameterization and exploration. Given a collection of shapes as input, the framework contains all the necessary functions for the shape synthesis process. This includes the identification of parts that can be exchanged, the parameterization and exploration step that enables the user to find the desired parts, and the actual shape synthesis stage where corresponding parts are exchanged to create a new shape.

The first step uses a modular and extensible co-analysis system to deal with the problems of part definition and correspondence. The aim of co-analysis is to segment shapes belonging to a certain family into parts and group these parts into categories such that parts that fulfill the same function belong to the same category. The reasoning behind choosing this method is that what constitutes a part is not solely defined by its geometry, but more importantly by its function. Thus it seems like a natural choice to allow parts that fulfill the same function to be exchanged. Instead of focusing on a particular co-analysis algorithm, this step is designed to be modular, allowing the use of different segmentation and clustering algorithms in the co-analysis process.

The parameterization and exploration stages are as described, allowing the user to browse the collection by first analyzing and then changing the arrangements between pairs of parts. Once the desired parts are found, the corresponding parts of the currently displayed shape can be exchanged with the found parts to create a new shape in the synthesis stage. This stage is meant as a proof-of-concept; while we do deal with the problem of aligning parts of different shapes, we do not concern ourselves with mesh cleanup such as the closing of holes that are created when exchanging parts.

The rest of this thesis is structured as follows. In Chapter 2 we go into the mathematical background behind the methods presented in this thesis, dealing with the topics of eigenvalues, graph theory and spectral clustering. Chapter 3 gives an overview of related work on methods regarding co-analysis, parameterization and exploration of shape collections and shape synthesis. The main contribution of this thesis is presented in Chapter 4, where we detail our proposed shape synthesis framework and method for parameterizing and exploring shape collections. The results of our methods are shown in Chapter 6. Finally, Chapter 7 concludes this thesis with an evaluation of the results and a look towards possible future work in the field.

CHAPTER 2

Foundation

This chapter explains some of the mathematical background of methods applied in this thesis. Section 2.1 deals with the properties of eigenvalues and eigenvectors which are useful tools for data analysis and mesh processing. Section 2.2 explains one of the applications of eigendecomposition called Principal Component Analysis. Since we are working with triangle meshes, Section 2.3 gives an overview of graph theory, highlighting the properties of graphs that are most useful to our purposes. Finally, Section 2.4 gives an introduction to the topic of spectral clustering which utilizes the properties of eigenvectors and graphs to solve the clustering problem. Proofs and further information on these topics can be found in many texts on linear algebra [TB97, Mey00] and spectral graph theory [Chu97, Lux06, ZVD10].

2.1 Eigenvalues and Eigenvectors

This section discusses eigenvalues and eigenvectors of matrices since they are useful tools for data analysis and mesh processing. Let $\mathbf{A}$ be a $n \times n$ square matrix. Then a non-zero vector $\mathbf{v}$ is an eigenvector of $\mathbf{A}$ if it satisfies

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}, \tag{2.1}$$

with λ being a scalar that is the eigenvalue associated with $\mathbf{v}$. In other words, when $\mathbf{v}$ is transformed by the matrix $\mathbf{A}$, the result is $\mathbf{v}$ itself multiplied by λ. A visual representation of this can be seen in Figure 2.1. Furthermore, any vector that is a scalar multiple of $\mathbf{v}$ is also an eigenvector of $\mathbf{A}$ with the same eigenvalue λ. Equation 2.1 can also be written as

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{v} = \mathbf{0}, \tag{2.2}$$

with $\mathbf{I}$ being the $n \times n$ identity matrix. Since any equation of the form $\mathbf{A}\mathbf{x} = 0$ only has a solution if and only if $\det(\mathbf{A}) = 0$, it follows that every eigenvalue λ of $\mathbf{A}$ satisfies

$$\det(\mathbf{A} - \lambda\mathbf{I}) = 0\,. \tag{2.3}$$

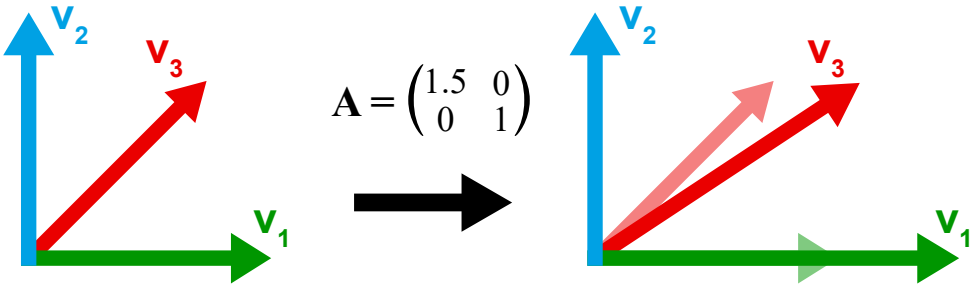

Figure 2.1: The vectors $\mathbf{v_1}$, $\mathbf{v_2}$ and $\mathbf{v_3}$ are transformed by the matrix $\mathbf{A}$ which scales the vectors in x-direction by a factor of 1.5. The directions of $\mathbf{v_1}$ and $\mathbf{v_2}$ remain the same, thus they are eigenvectors of $\mathbf{A}$ with corresponding eigenvalues $\lambda_1 = 1.5$ and $\lambda_2 = 1$. Since the direction of $\mathbf{v_3}$ is changed by the transformation, it is not an eigenvector.

The left-hand side of this equation is called the *characteristic polynomial* $p(\lambda)$ of $\mathbf{A}$. The degree of $p(\lambda)$ is n and thus it also has n roots, which are the eigenvalues of $\mathbf{A}$. An eigenvalue of $\mathbf{A}$ may be a complex number even if every entry of $\mathbf{A}$ is real. Furthermore, eigenvalues may not always be distinct, meaning it is possible that $\lambda_i = \lambda_j$ for $i \neq j$. The set of distinct eigenvalues is called the *spectrum* of $\mathbf{A}$.

Of particular interest to us are symmetric matrices. A matrix $\mathbf{S}$ is symmetric if $\mathbf{S} = \mathbf{S}^T$. Symmetric matrices have a number of properties that are useful for mesh processing and data analysis. Some of these properties are given by the Spectral Theorem. Let $\mathbf{S}$ be a real symmetric $n \times n$ matrix, then

$$\mathbf{S} = \mathbf{V}\mathbf{D}\mathbf{V}^T = \sum_{i=1}^{n} \lambda_i \mathbf{v_i} \mathbf{v_i}^T \tag{2.4}$$

is the *eigendecomposition* of $\mathbf{S}$, with $\mathbf{V} = (\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_n})$ being the matrix of eigenvectors of $\mathbf{S}$ and $\mathbf{D}$ being the diagonal matrix of the eigenvalues of $\mathbf{S}$. It is possible to derive the following properties from this theorem:

- The eigenvalues of $\mathbf{S}$ are real numbers.
- $\mathbf{S}$ has n eigenvectors that form an orthogonal basis.
- $\mathbf{S}$ is diagonalizable.

Another important theorem is the Courant-Fischer theorem. Given a symmetric $n \times n$ matrix $\mathbf{S}$ with eigenvalues λ_i sorted in ascending order, that is $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, the eigenvalues satisfy

$$\lambda_i = \min_{\substack{\mathcal{V} \subset \mathbb{R}^n \\ dim\mathcal{V} = i}} \max_{\substack{\mathbf{v} \in \mathcal{V} \\ \|\mathbf{v}\| = 1}} \mathbf{v}^T \mathbf{S} \mathbf{v}, \tag{2.5}$$

with $\mathcal{V}$ being a subspace of $\mathbb{R}^n$ with dimension i. As a result, the smallest eigenvalue λ_1 is given by

$$\lambda_1 = \min_{\|\mathbf{v}\|=1} \mathbf{v}^T \mathbf{S} \mathbf{v}, \tag{2.6}$$

and the largest eigenvalue λ_n is given by

$$\lambda_n = \max_{\|\mathbf{v}\|=1} \mathbf{v}^T \mathbf{S} \mathbf{v}\,. \tag{2.7}$$

Note that the condition $\|\mathbf{v}\| = 1$ can be removed when one replaces the right side of the equation with the Rayleigh quotient $\frac{\mathbf{v}^T \mathbf{S} \mathbf{v}}{\mathbf{v}^T \mathbf{v}}$. From this it is simple to obtain all eigenvalues λ_i and their corresponding eigenvectors $\mathbf{v_i}$. Using the previous equation to compute the largest eigenvalue λ_n, the second largest eigenvalue λ_{n-1} can be found. We need to find the vector $\mathbf{v}$ that minimizes $\max \mathbf{v}^T \mathbf{S} \mathbf{v}$ under the constraint $\|\mathbf{v}\| = 1$ over all subspaces $\mathcal{V} \subset \mathbb{R}^n$ with dimension $n-1$. We already know that the vector $\mathbf{v_n}$ maximizes this equation, and thus the desired subspace must be orthogonal to $\mathbf{v_n}$, which leaves only one choice that we denote as $\mathcal{V}_{n-1}$. Thus the equation becomes

$$\lambda_n = \max_{\substack{\mathbf{v} \in \mathcal{V}_{n-1} \\ \|\mathbf{v}\|=1}} \mathbf{v}^T \mathbf{S} \mathbf{v}\,. \tag{2.8}$$

This process can be continued to compute each λ_i by maximizing the equation over the vectors of the subspace $\mathcal{V}_i$ which has dimension i and is orthogonal to all previously computed eigenvectors λ_j, $j = i+1, \ldots, n$. This makes the theorem useful for optimization problems, an example of which is explained in the following section.

2.2 Principal Component Analysis

Principal Component Analysis (PCA) [Jol02] is a procedure that transforms a set of correlated variables into a set of uncorrelated variables called *principal components* that are ordered by their variance. This makes it a useful tool in trying to solve the problem of dimensionality reduction by discarding the principal components with low variance. Consider the data set $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}\}$ with $\mathbf{x_i} = (x_{i1}, x_{i2}, \ldots, x_{im})^T$, containing n samples with m observed variables each. A large m can be troublesome when it comes to visually representing the data or when the data is used in computer algorithms whose performance depends on m. Thus it is desirable to reduce the dimensionality of the data by finding a representation $\mathbf{y_i} = (y_{i1}, y_{i2}, \ldots, y_{ip})^T$ for every $\mathbf{x_i}$ with $p < m$. Of course, reducing the number of dimensions also means discarding some of the information contained in the data. Thus the aim is to minimize the amount of information that is lost.

An example can be seen in the data plot shown in Figure 2.2, consisting of 25 samples with 2 observed variables each. The left side shows the data in the space formed by the 2 observed variables. The simplest way to reduce the dimensionality would be to pick either of the two variables and discard the other one. However, a lot of information would be lost that way. A much better pick would be the line shown in red – it is the

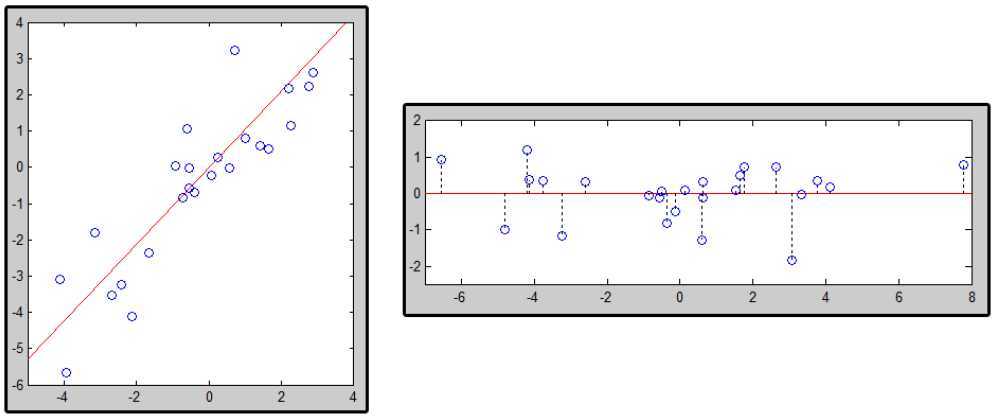

Figure 2.2: Two plots of 25 data samples with 2 observed variables each. On the left side the x- and y-axis correspond to the observed variables and the red line shows the first principal component. The plot on the right shows the data in the principal component space, using the first and second principal components as the x- and y-axis respectively.

direction in which the data set has the largest variance, also called the first principal component. Thus we want to find the corresponding coefficients that let us express the data using the first principal component as the first coordinate axis. In general, we want to find p coefficients $\mathbf{c_j} = (c_{j1}, c_{j2}, \ldots, c_{jm})^T, j = 1, \ldots, p$ that can be used to transform $\mathbf{x_i}$ into $\mathbf{y_i}$ using

$$y_{ij} = \mathbf{c_j}^T \mathbf{x_i} \,. \tag{2.9}$$

First a number of definitions. The mean $\bar{x}_j$ of the vectors $\mathbf{x_i}$ for variable j is given by

$$\bar{x}_j = \frac{1}{n} \sum_{i=1}^{n} x_{ij} \,. \tag{2.10}$$

The mean is used to compute the variance of the data, which describes how much the data varies for a given variable. The variance σ_j^2 of variable j is given by

$$\sigma_j^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_{ij} - \bar{x}_j)^2 \,. \tag{2.11}$$

The variance only gives information about each variable individually. However, there is the possibility of variables being correlated, meaning that the value of one variable depends on the value of the other. This is given by the covariance σ_{jk} between variables j and k:

$$\sigma_{jk} = \frac{1}{n-1} \sum_{i=1}^{n} (x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k) \,. \tag{2.12}$$

It is easy to see that the covariance σ_{jj} is exactly the variance σ_j^2. Using the variances and covariances we can now construct the $m \times m$ covariance matrix $\mathbf{\Sigma}$ with entries σ_{jk} for $j, k = 1, \ldots, m$. Since $\sigma_{jk} = \sigma_{kj}$, $\mathbf{\Sigma}$ is also symmetric.

As mentioned before, our goal is to find the coefficients $\mathbf{c_j}$ for the directions which maximize the variance of the data samples. Starting with $j = 1$ we want to maximize $\text{var}(\mathbf{c_1}^T\mathbf{x})$. Inserting $\mathbf{c_1}^T\mathbf{x}$ into Equation 2.11, this can also be written as $\mathbf{c_1}^T\mathbf{\Sigma c_1}$. At a glance, this form looks very similar to the formulation of the Courant-Fischer theorem, suggesting that the desired coefficients $\mathbf{c_j}$ may be the eigenvectors of $\mathbf{\Sigma}$.

To see if that is the case, we can use the method of Lagrange multipliers. Using the constraint $\|\mathbf{c_1}\| = 1$ which is equivalent to $\mathbf{c_1}^T\mathbf{c_1} - 1 = 0$, we want to maximize

$$\mathbf{c_1}^T\mathbf{\Sigma c_1} - \lambda(\mathbf{c_1}^T\mathbf{c_1} - 1), \tag{2.13}$$

with λ being a Lagrange multiplier. Differentiating with respect to $\mathbf{c_1}$ results in

$$(\mathbf{\Sigma} - \lambda\mathbf{I})\mathbf{c_1} = \mathbf{0}, \tag{2.14}$$

where $\mathbf{I}$ is the $p \times p$ identity matrix. It follows that $\mathbf{c_1}$ is an eigenvector of $\mathbf{\Sigma}$ and λ is its corresponding eigenvalue. We want to maximize $\mathbf{c_1}^T\mathbf{\Sigma c_1}$ and since $\mathbf{c_1}$ is an eigenvector of $\mathbf{\Sigma}$, this means that

$$\mathbf{c_1}^T\mathbf{\Sigma c_1} = \mathbf{c_1}^T\lambda\mathbf{c_1} = \lambda\mathbf{c_1}^T\mathbf{c_1} = \lambda\,. \tag{2.15}$$

So the direction of the largest variance is the eigenvector of $\mathbf{\Sigma}$ corresponding to the largest eigenvalue λ_1, which is also the variance of the first principal component. This is also true in general – the jth principal component is the eigenvector $\mathbf{c_j}$ corresponding to the jth eigenvalue λ_j (sorted in descending order) which is also the variance of the data in the direction of $\mathbf{c_j}$.

Showing that this is true is a little more complex, since it requires the additional constraint that the variance of the jth principal component is uncorrelated with all previous principal components. For $j = 2$ we have the additional constraint $cov(\mathbf{c_1}^T\mathbf{x}, \mathbf{c_2}^T\mathbf{x}) = 0$ with

$$cov(\mathbf{c_1}^T\mathbf{x}, \mathbf{c_2}^T\mathbf{x}) = \mathbf{c_1}^T\mathbf{\Sigma c_2} = \mathbf{c_2}^T\mathbf{\Sigma c_1} = \mathbf{c_2}^T\lambda_1\mathbf{c_1} = \lambda_1\mathbf{c_1}^T\mathbf{c_2}\,. \tag{2.16}$$

We can use $\mathbf{c_1}^T\mathbf{c_2} = 0$ as the constraint that the covariance between the two principal components is zero. Following the same process as before, we want to maximize

$$\mathbf{c_2}^T\mathbf{\Sigma c_2} - \lambda(\mathbf{c_2}^T\mathbf{c_2} - 1) - \phi\mathbf{c_2}^T\mathbf{c_1}, \tag{2.17}$$

with λ and ϕ being Lagrange multipliers. Differentiating with respect to $\mathbf{c_2}$ leads to

$$\mathbf{\Sigma c_2} - \lambda\mathbf{c_2} - \phi\mathbf{c_1} = \mathbf{0}, \tag{2.18}$$

and multiplying with ${\mathbf{c_1}}^T$ results in

$$\mathbf{c_1}^T\mathbf{\Sigma c_2} - \lambda\mathbf{c_1}^T\mathbf{c_2} - \phi\mathbf{c_1}^T\mathbf{c_1} = 0\,. \tag{2.19}$$

Since the first two terms are the covariance, they are both zero, so it follows that $\phi = 0$. Inserting into Equation 2.18 results in $(\mathbf{\Sigma} - \lambda\mathbf{I})\mathbf{c_2} = \mathbf{0}$, thus $\mathbf{c_2}$ is an eigenvector of $\mathbf{\Sigma}$ and λ is its corresponding eigenvalue. Again we want λ to be as large as possible, but because we have the constraint $\mathbf{c_1}^T\mathbf{c_2} = 0$, it follows that $\mathbf{c_1} \neq \mathbf{c_2}$ and thus $\lambda \neq \lambda_1$

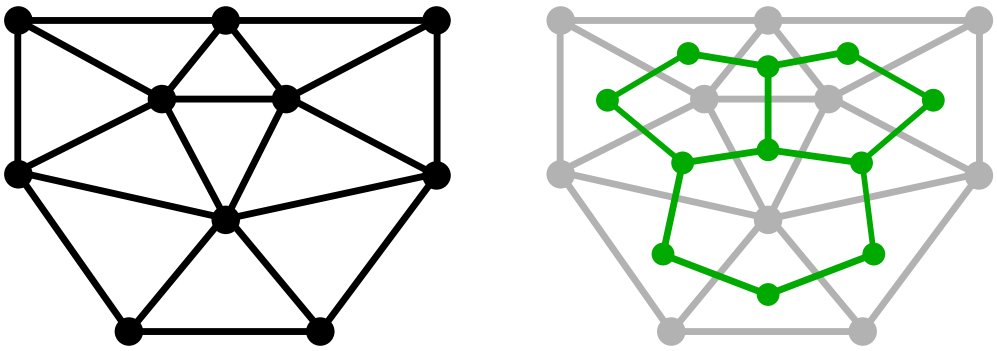

Figure 2.3: A triangle mesh represented as a graph (left) and its corresponding dual graph (right). Vertices are drawn as circles and edges are drawn as line segments. The vertices and edges of the graph on the left correspond to the vertices and edges of the triangle mesh. The dual graph on the right is shown in green, and its vertices correspond to the faces of the triangle mesh, with the edges showing which faces are adjacent.

(unless $\mathbf{\Sigma}$ has some non-distinct eigenvalues, a case which is discussed in [Jol02]). So λ must be the second-largest eigenvalue λ_2. For further λ_j this can be shown in a similar way.

So the coefficients we need to transform the $\mathbf{x_i}$ with dimension m into $\mathbf{y_i}$ with dimension p are the eigenvectors corresponding to the p largest eigenvalues of the covariance matrix $\mathbf{\Sigma}$. Let $\mathbf{C}$ be the matrix that has the p largest eigenvectors $\mathbf{c_j}$ as its columns, then $\mathbf{y_i}$ can be obtained by

$$\mathbf{y_i} = \mathbf{C}^T \mathbf{x_i} \,. \tag{2.20}$$

The right side of Figure 2.2 shows PCA applied to the example dataset mentioned earlier. The x- and y-axis correspond to the first and second principal components respectively and the dashed lines show the projection of the data points to the first principal component.

2.3 Graph Theory

Mathematically, a triangle mesh is usually represented as an undirected graph $G = (V, E)$, with $V = \{v_1, \ldots, v_n\}$ being the set of vertices of the mesh and $E = \{e_{ij} | v_i, v_j \in V\}$ being the set of edges connecting vertices v_i and v_j. G being undirected means that every edge is undirected, that is $e_{ij} = e_{ji}$. A triangle mesh can also be represented as the dual graph $H = (F, E)$, with $F = \{t_1, \ldots, t_k\}$ being the faces of the triangle mesh and $E = \{e_{ij} | t_i, t_j \in F\}$ being the edges connecting faces t_i and t_j. A weighted graph has the additional property that all edges have a weight $w_{ij} \geq 0$ associated with them, usually a measure of distance or similarity. Naturally it follows that $w_{ij} = w_{ji}$ for an undirected edge.

It is also possible to represent a graph by its adjacency matrix $\mathbf{W}$, which is a square matrix with dimension n. For a weighted graph, the entries of $\mathbf{W}$ are the edge weights w_{ij} if there is an edge that connects vertex v_i to v_j, and 0 otherwise. In an unweighted graph, the weight of every edge is simply set to 1. Also, if the graph is undirected, then $\mathbf{W}$ is symmetric. The degree d_i of a vertex v_i is the sum of the weights of its incident edges, given by

$$d_i = \sum_{j=1}^{n} w_{ij} \,. \tag{2.21}$$

This also means that in an unweighted graph, d_i is equal to the number of edges incident to v_i.

A *walk* in a graph is a sequence of vertices $(v_1, v_2, \ldots, v_s)$ where every edge $e_{i-1,i} \in E$. A *path* is a walk where every vertex in the sequence is unique. A graph is called *connected* if for any two vertices $v_i, v_j \in V$ there exists a path connecting them, and *fully connected* if there exists an edge $e_{i,j} \in E$ between every pair of vertices $v_i, v_j \in V$. A subset $U \subseteq V$ is called a connected component if there exists a path connecting any two vertices $u_i, u_j \in U$, but no path between any two vertices $u_i \in U, v_j \in V \setminus U$. A graph is called *bipartite* if the set of vertices V can be partitioned into two disjoint sets V_1 and $V_2 = V \setminus V_1$ such that $v_i \in V_1, v_j \in V_2$ for every edge $e_{ij} \in E$.

A *random walk* is created by choosing a vertex as the first element of the walk and using the edge weights of its incident edges as probabilities to pick the next vertex in the sequence. The probabilities can be written as a $n \times n$ matrix $\mathbf{P}$ with entries $p_{ij} = w_{ij}/d_i$. Given a row vector $\mathbf{f} = (f_1, f_2, \ldots, f_n)$ with $\sum f_i = 1$ that contains the initial probability distribution to start at each vertex, the distribution after k transitions between neighbors is given by $\mathbf{fP}^k$. A distribution $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ is called *stationary* if $\pi\mathbf{P} = \pi$. A graph is called *ergodic* if there is a unique stationary distribution π that satisfies

$$\lim_{k \to \infty} \mathbf{fP}^k = \pi, \tag{2.22}$$

for any initial distribution $\mathbf{f}$. It can be shown that any connected non-bipartite graph is ergodic. In such a case the entries of π are given by

$$\pi_i = \frac{d_i}{\mathrm{vol}(V)}, \tag{2.23}$$

with $\mathrm{vol}(V)$ being the sum of the edge weights of all vertices in V given by

$$\mathrm{vol}(V) = \sum_{i=1}^{n} d_i \,. \tag{2.24}$$

A k-way *cut* of a graph is a partitioning of the set of vertices V into k disjoint sets $V_1, V_2, \ldots, V_k$. The corresponding *cut set* is the set of edges $E_C = \{e_{ij} | v_i \in V_I, v_j \in V_J, I \neq J\}$ where each edge connects vertices of different partitions. The weight of the cut is then the sum of edge weights of all edges $e \in E_C$. A cut is called the *minimum cut* if it is the cut with minimum weight out of all possible cuts for a given k.

2.4 Spectral Clustering

The problem of clustering poses a challenge in many fields such as computer graphics, data analysis, machine-learning and more. Given a set of data $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}\}$, the goal of clustering is to assign each $\mathbf{x_i}$ to a cluster C_j such that, given a measure of dissimilarity such as the distance between the data points, the dissimilarity between any two points of the same cluster is low and the dissimilarity between any two points of different clusters is high.

A popular clustering algorithm is the k-means algorithm. As the first step, k points are selected as the centroids for each cluster, either randomly or through some heuristic. Then each data point is assigned to the cluster whose centroid is closest to the data point, followed by recomputing the centroid of each cluster from the average of its assigned data points. The process of assigning points to clusters and recomputing the centroids is then repeated until there is no change in the cluster assignments.

However, the k-means algorithm has a number of drawbacks, such as having to specify the number of clusters k beforehand and the quality of the solution converging only to a local minimum, as it depends on the centroids chosen in the first step. In general, the euclidean distance between the data points may not be a good measure for similarity, especially when the optimal clusters are non-linear. Consider the example shown on the left side of Figure 2.4. Merely by looking at the plot of the data it is easy to see that the distribution of the data points forms two circles which would serve well as the desired clusters. However, applying the k-means algorithm with $k = 2$ results in the clusters shown on the right of Figure 2.4. The k-means algorithm separates the data based on a linear discriminant, which leads to an undesirable result because the optimal clusters are non-linear.

One solution to this problem is to find a representation of the data in a different space where it is easier to separate the data. There are different ways to achieve this, but we will focus on the method of *spectral clustering* since it uses a graph representation of the data, which is highly relevant for us because we are dealing with triangle meshes.

To create a similarity graph of the data, each data point is treated as a vertex and connected to the other data points with edges. The weight of each edge is set to be the similarity between the vertices of the edge, which is usually inversely proportional to their distance. There are different ways to connect the vertices, such as connecting the nearest neighbors of each vertex, connecting all vertices whose distance is below a threshold ϵ, or simply creating a fully connected graph. The latter requires the similarity measure to model a local neighborhood though because otherwise the resulting weight matrix would not be sparse. When working with large matrices, it is often desirable that they be sparse, meaning that most entries are zero. This is because many numerical algorithms, like the computation of eigenvalues and eigenvectors, perform better with sparse matrices.

The reasoning behind representing the data as a graph is related to random walks and graph cuts. For the former, we can treat the edge weights as the probabilities to transition from one vertex to the other. We want to choose clusters such that most

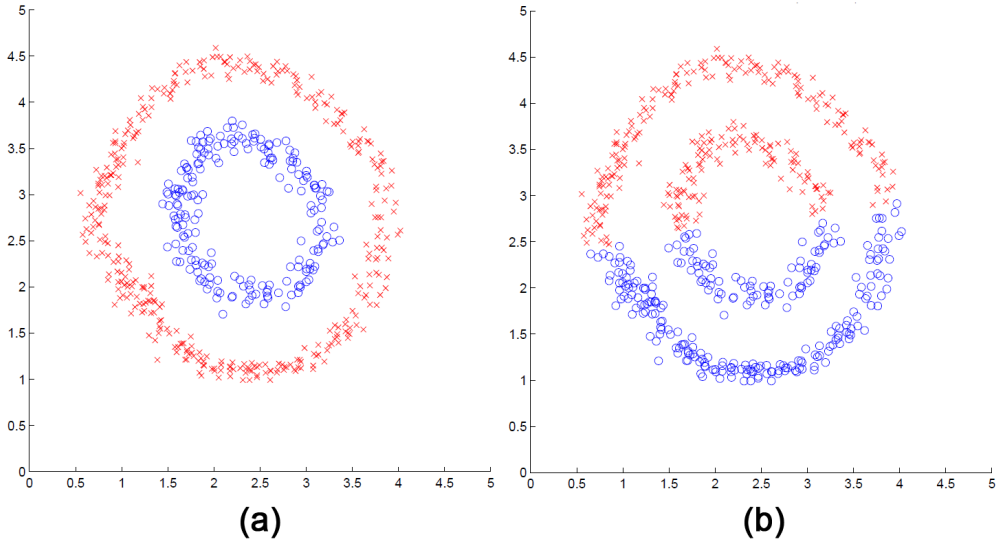

Figure 2.4: Clustering of the 2-dimensional *twocircles* dataset into 2 clusters. (a) The optimal clusters are non-linear and have the shape of a circle. These clusters can be obtained by spectral clustering. (b) The same dataset clustered with k-means, which leads to an undesirable result. Figure adapted from [NJW02].

vertices of a cluster are close to other vertices of the same cluster, and thus the probability to stay inside a cluster is higher than the probability to transition to a different cluster.

The graph cut view is that we want to partition the set of vertices such that each partition corresponds to a cluster. Since we want the similarity between clusters to be low, we want the weight of the cut to be low as well, which leads to an optimization problem. However, the minimum cut is not always the best solution to the problem, since it tends to put isolated vertices into separate clusters because they are connected only by a small number of edges, which leads to a smaller cut cost. The RatioCut cost [HK92] tries to alleviate this by normalizing by the number of nodes in each cluster, thus aiming for the size of each cluster to be about equal. It is given by

$$\text{RatioCut}(V_1, V_2, \ldots, V_k) = \sum_{i=1}^{k} \frac{\text{cut}(V_i, V \setminus V_i)}{|V_i|}, \tag{2.25}$$

where $\text{cut}(A, B)$ is the sum of edge weights connecting vertices of A with vertices of B. An even better solution is provided by the Normalized Cut cost [SM00] which is given by

$$\text{Ncut}(V_1, V_2, \ldots, V_k) = \sum_{i=1}^{k} \frac{\text{cut}(V_i, V \setminus V_i)}{\text{vol}(V_i)}, \tag{2.26}$$

where $\text{vol}(A)$ is the sum of weights for all edges incident to vertices in A. In other words, we want the similarity of edges inside the clusters to be high and the similarity of

edges between clusters to be low. Unfortunately, finding the indicator vector $\mathbf{f}$ (a vector with discrete values that indicate which cluster each vertex belongs to) that minimizes either cut cost is NP-hard. However, by introducing the relaxation criterion that the resulting indicator vector is also allowed to take on real values, it is possible to compute an approximate solution through spectral clustering. Relaxing the minimization problem for the RatioCut cost with $k = 2$ leads to

$$\min_{\mathbf{f}} \text{RatioCut}(\mathbf{f}) = \min_{\mathbf{v}} \frac{\mathbf{v}^T \mathbf{L} \mathbf{v}}{\mathbf{v}^T \mathbf{v}}, \tag{2.27}$$

with $\mathbf{L} = \mathbf{D} - \mathbf{W}$ being called the *Laplacian* of the graph $G(V, E)$, $\mathbf{D}$ being the diagonal matrix of the vertex degrees and $\mathbf{W}$ being the weight matrix of the graph. Thus the Laplacian is a $n \times n$ matrix with entries

$$\mathbf{L}(i,j) = \begin{cases} d_i - w_{ii}, & \text{if } i = j \\ -w_{ij}, & \text{if } e_{ij} \in E \\ 0, & \text{otherwise} \end{cases}. \tag{2.28}$$

Since both $\mathbf{D}$ and $\mathbf{W}$ are symmetric, $\mathbf{L}$ is also symmetric and has n real eigenvalues and n eigenvectors that form an orthogonal basis. Furthermore, if G is connected, then the smallest eigenvalue is $\lambda_1 = 0$, with the corresponding eigenvector being the one vector $\mathbf{1}$ where every entry is 1. If G has k connected components, then the eigenvalues are $\lambda_j = 0$ for $j \leq k$ and $\lambda_j \neq 0$ for $j > k$. The eigenvectors corresponding to the λ_j, $j \leq k$ are the indicator vectors $\mathbb{1}_j$ with the ith entry $\mathbb{1}_j(i) = 1$ if $v_i \in V_j$ and $\mathbb{1}_j(i) = 0$ otherwise, where the V_j are the connected components of G.

Considering these properties and applying the Courant-Fischer theorem, the optimal 2-way partitioning is given by the second smallest eigenvector $\mathbf{v_2}$ of $\mathbf{L}$ which is also called the *Fiedler vector*. The vertices v_i of the graph can then be assigned to one of the two clusters based on the sign of their corresponding entry $\mathbf{v_2}(i)$. The clusters for $k > 2$ can be obtained by recursively partitioning the set of vertices using the second to kth eigenvectors. Alternatively, it is possible to approach the problem from the other direction and group the vertices based on the k largest eigenvectors, resulting in the following algorithm:

1. Compute the Laplacian matrix $\mathbf{L}$.
2. Compute the k eigenvectors $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_k}$ corresponding to the k largest eigenvalues of $\mathbf{L}$.
3. Construct the $n \times k$ matrix $\mathbf{V} = (\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_k})$ with the computed eigenvectors as its columns.
4. Treat the n rows of $\mathbf{V}$ as points $\mathbf{y_i}$ in $\mathbb{R}^k$ and perform k-means clustering.
5. Assign each original vertex v_i to a cluster C_j if its corresponding point $\mathbf{y_i} \in C_j$.

The problem with using $\mathbf{L}$ in the spectral clustering algorithm is that doing so only optimizes the criterion that the similarity between points of different clusters should be low, but it does not necessarily guarantee a high similarity between points in the same cluster. This is the objective of the Normalized Cut cost. By relaxing the Normalized Cut cost minimization problem it can be written as

$$\min_{\mathbf{f}} \mathrm{Ncut}(\mathbf{f}) = \min_{\mathbf{v}} \frac{\mathbf{v}^T \mathbf{L} \mathbf{v}}{\mathbf{v}^T \mathbf{D} \mathbf{v}} \,. \tag{2.29}$$

Applying the Courant-Fisher theorem leads to the generalized eigenproblem $\mathbf{L}\mathbf{v} = \lambda \mathbf{D}\mathbf{v}$ and thus to the formulation of the normalized Laplacian $\mathbf{L_{rw}} = \mathbf{D}^{-1}\mathbf{L}$ with entries:

$$\mathbf{L_{rw}}(i,j) = \begin{cases} 1 - \frac{w_{ii}}{d_i}, & \text{if } i = j \\ -\frac{w_{ij}}{d_i}, & \text{if } e_{ij} \in E \\ 0, & \text{otherwise} \end{cases} . \tag{2.30}$$

This formulation is closely related to random walks, as the transition matrix $\mathbf{P}$ can be written as $\mathbf{P} = \mathbf{I} - \mathbf{L_{rw}}$. Note that $\mathbf{L_{rw}}$ is not a symmetric matrix, which in general means that it does not necessarily have real eigenvalues. However, one can extend $\mathbf{L}\mathbf{v} = \lambda\mathbf{D}\mathbf{v}$ to $\mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}\mathbf{v} = \lambda\mathbf{D}^{\frac{1}{2}}\mathbf{v}$. This leads to a different formulation of the normalized Laplacian $\mathbf{L_{sym}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$ with entries

$$\mathbf{L_{sym}}(i,j) = \begin{cases} 1 - \frac{w_{ii}}{d_i}, & \text{if } i = j \\ -\frac{w_{ij}}{\sqrt{d_i d_j}}, & \text{if } e_{ij} \in E \\ 0, & \text{otherwise} \end{cases} , \tag{2.31}$$

which is a symmetric matrix and thus has n real eigenvalues and n eigenvectors that form an orthogonal basis. Let us denote the eigenvectors of $\mathbf{L_{sym}}$ as $\mathbf{u_i}$, then they are related to the eigenvectors $\mathbf{v_i}$ of $\mathbf{L_{rw}}$ with $\mathbf{u_i} = \mathbf{D}^{\frac{1}{2}}\mathbf{v_i}$. Thus the eigenvectors $\mathbf{v_i}$ are indeed the real valued solution to minimizing the Normalized Cut cost. They can be used in the same way as the eigenvectors of the normalized Laplacian $\mathbf{L}$ to solve the clustering problem.

Of course it is also possible to use the eigenvectors $\mathbf{u_i}$ of $\mathbf{L_{sym}}$ directly for the clustering as proposed in [NJW02]. In this case it is necessary to normalize each row of the matrix $\mathbf{U} = (\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_k})$ between steps 3. and 4. of the spectral clustering algorithm. Figure 2.5 shows this method applied to the same dataset as in Figure 2.4.

As mentioned before, spectral clustering is of special interest to us because it can be directly applied to triangle meshes. On example application is mesh segmentation. By defining a similarity function on the edges of the dual graph of the mesh, the faces of the mesh can be clustered into disjoint segments. A common choice for the similarity function is based on the dihedral angle and edge length between the faces, thus preferring short boundaries on concave regions of the mesh.

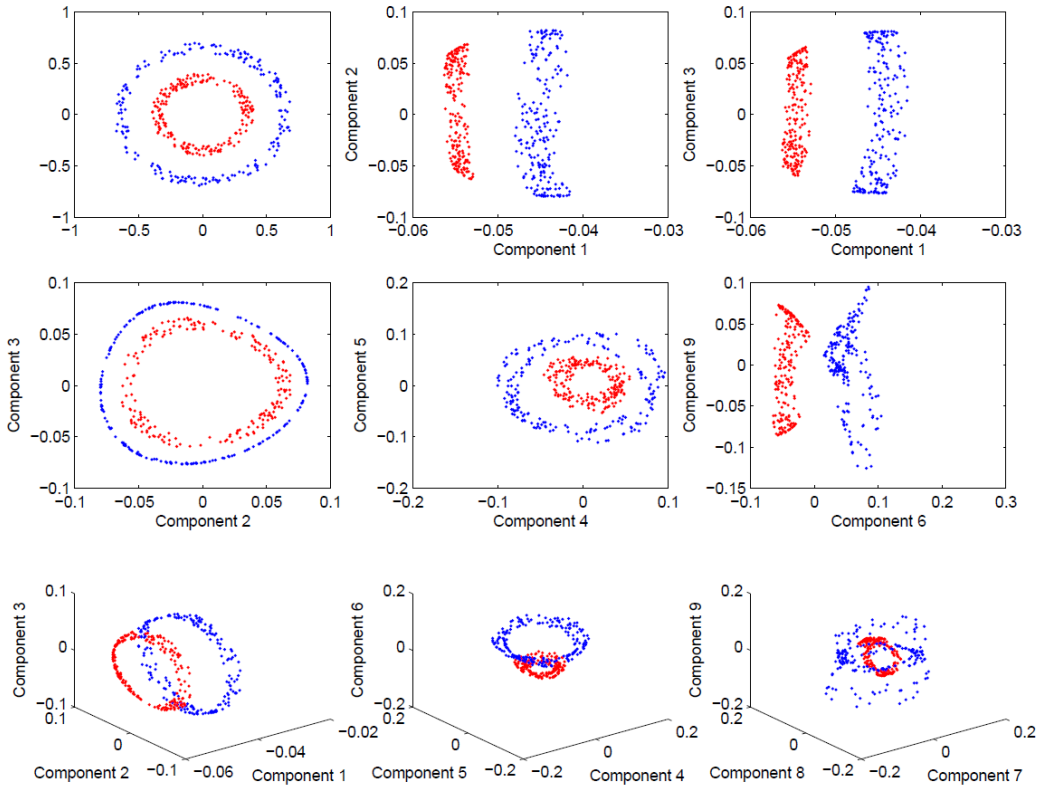

Figure 2.5: Plots of the eigenvalues obtained by applying the method of Ng et al. [NJW02] to the *twocircles* dataset. Using the eigenvalues, it is much easier to find the desired clusters. Figure adapted from [Ray04].

CHAPTER 3

Related Work

The framework proposed in this thesis consists of three stages: co-analysis, analysis of shape collections and shape synthesis. The following sections describe previous research related to each stage.

3.1 Co-Analysis

The aim of co-analysis is to obtain a consistent co-segmentation of a shape collection. Given a set of shapes that belong to the same family, the goal is to segment each shape in such a way that all parts that have the same semantic function are assigned the same label. The following subsections describe different approaches to achieve this goal.

3.1.1 Feature-based Methods

One approach to co-analysis is to compute features for each face of the shape and then use the distribution of these features to group the shape segments into categories. The following subsubsections provide more detailed information on the types of features and methods that have been proposed.

Features

Various features that capture local or global geometrical properties of a shape can be computed for each triangle of a shape. The idea behind using these features is that parts of a shape that serve the same function tend to have similar features. The following list of features is by no means exhaustive, but they are commonly used in feature-based co-analysis and also in our own framework. Figure 3.1 shows a number of features computed on a shape.

- The Average Geodesic Distance (AGD) [HS01] at a point on a surface is defined as the average of the geodesic distances between the point and all other points on

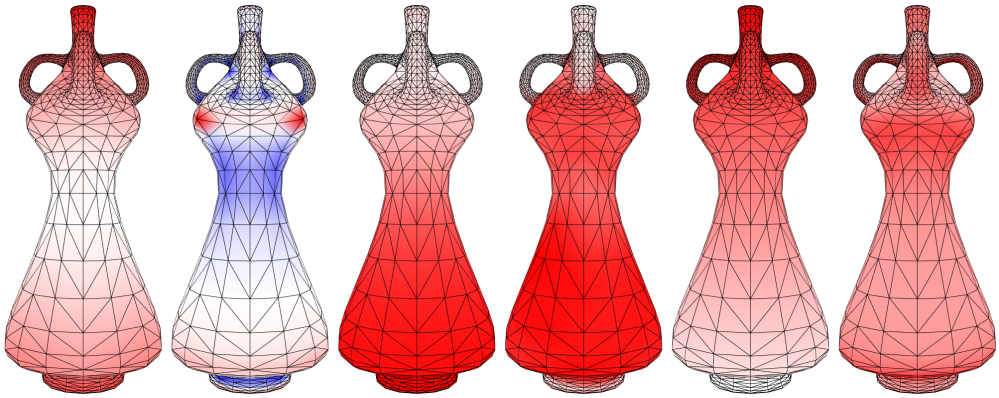

Figure 3.1: A number of features computed on a shape. Blue colors correspond to negative values, red colors to positive values and white colors to values close to zero. From left to right: Average Geodesic Distance, Gaussian Curvature, Conformal Factor, Shape Diameter Function, the distance from the base of the shape, and the normal orientation.

the surface. Since the exact computation of the AGD is expensive, it is usually approximated. One way to do this for a triangle mesh is to use Dijkstra's algorithm to compute the pairwise distances between all pairs of vertices, although to increase accuracy it might be necessary to add additional points and edges to the surface.

- The Gaussian Curvature (GC) at a point on a surface is the product of the two principal curvatures at that point. On a triangle mesh the GC can be approximated in a number of ways [GG06]. One example is the Angle Deficit method, which computes the GC at a vertex as 2π minus the sum of the angles around the vertex, weighted by the surface area associated with the vertex.

- The Conformal Factor (CF) [BCG08] is a scalar function on the surface which describes a mapping of the surface to another surface with the same topology and with constant GC. For a triangle mesh, the CF can be obtained by solving a set of linear equations:

$$\mathbf{L\Phi} = \mathbf{K^t} - \mathbf{K^o}, \tag{3.1}$$

  where $\mathbf{L}$ is the Laplacian of the mesh, $\mathbf{\Phi}$ is the CF and solution of the system, $\mathbf{K^o}$ is the GC of the original mesh and $\mathbf{K^t}$ is the target GC. $\mathbf{K^t}$ can be computed by assigning each vertex the average GC of the mesh weighted by the surface area associated with the vertex.

- The Shape Diameter Function (SDF) [SSCO08] is a scalar function that describes the diameter of the shape in the neighborhood of a point on the surface. To compute the SDF, a cone is placed at the point with the the reversed normal vector as its center. Several rays are shot within the cone and intersected with the the mesh. For each ray, only the first intersection with the mesh is considered, also ignoring

any intersections with the outside of the mesh (i.e. intersections where the normal vector points in the opposite direction of the ray direction). The SDF is then the weighted average of the length of each ray, with the weights being the inverse of the angle between the ray and the cone center because rays with a larger angle are more frequent.

- Shape Context (SC)s were originally proposed for object recognition in 2D [BMP02] and later adapted for use with 3D shapes [KPNK03]. A SC at a point describes the position of all other points of the shape relative to the current point. The relative position is expressed by the geodesic distance and the angle between the normal vector of the current point and the point-to-point vector. These two features are used to create a 2D histogram capturing the distribution of the other points relative to the current point.

Sidi et al. [SvKK+11] also propose two further features. The first is the geodesic distance of a point to the base of the shape, while the second is the orientation of the normal vector relative to the up-vector of the coordinate system. Both features assume that the shape has an upright orientation.

Supervised methods

Supervised co-analysis methods are based on machine-learning algorithms and require a training dataset where each shape part has already been assigned the correct label. Based on the training data the algorithm learns the probability of a face having a certain label, which can be used to classify further data. An example can be seen in Figure 3.2.

Kalogerakis et al. [KHS10] compute the labels of each face by minimizing an energy function consisting of two terms: an unary term and a pairwise term. The unary term consist of feature vectors for each face of the shape that contain features like the ones described above. This term describes the probability of the face having a certain label based on the computed features. The probabilities are learned by training classifiers using the training set. The pairwise term of the energy function consists of pairwise features computed for each pair of adjacent faces, such as the dihedral angle between the two faces and the differences between some of the unary features. This term is used to penalize adjacent faces being assigned different labels. The solution for minimizing the energy function is then obtained using graph cuts [BBV+01].

Van Kaick et al. [VTS+11] use a similar approach, but their joint labeling method further utilizes pairwise correspondences between shapes. The energy function is extended by a third term that describes whether or not a face on one shape S should have the same label as a face on another shape T. To learn how significant each unary feature is for this decision, a classifier is trained using the training dataset. For each face in S, its similarity to all faces in T is computed for each unary feature. Then a small number of the most similar assignments is chosen for each face in S, together with whether or not they have the same label, to create the set of vectors used to train the classifier. Finally, the classifier is used in the query step to find the most likely assignments between faces

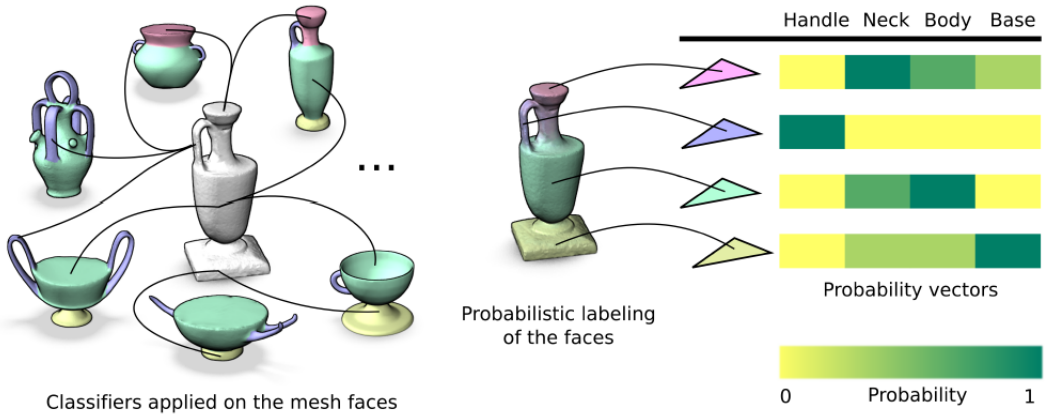

Figure 3.2: An example for supervised feature-based co-analysis [VTS+11]. Classifiers are trained to learn the probabilities of a face having a certain label based on its features. Then the classifiers are used to categorize faces of unknown shapes.

that should have the same label. For each a pair of shapes, the third term of the energy function consists of the top 20% assignments with the highest probability of having the same label.

Unsupervised methods

The advantage of unsupervised algorithms is that all the information used in the co-analysis is obtained from the input set of shapes itself, thus a pre-segmented training dataset is not required. In all of the following methods each shape is first segmented individually, creating an oversegmentation of the shape. These segments are then clustered based on the per-face features.

Sidi et al. [SvKK+11] use the SDF, the orientation of the face normals and the geodesic distance from the base of the shape as their per-face features. These features are used to create an initial oversegmentation of each individual shape using the mean-shift algorithm [CM02]. Per-segment features are also defined and include the distribution of the per-face features, the area of the segment and a vector of three components based on eigenvalues that describe the overall geometry of the shape. The segments are then embedded into a new space using diffusion maps [NLCK05] and clustered using an agglomerative hierarchical algorithm. Finally, the co-segmentation is refined by creating a statistical model of each label based on the per-face features and then minimizing an energy function consisting of two terms. The first term is the probability of a face having a certain label and the second term penalizes adjacent faces being assigned different labels based on the dihedral angle and the length of the edge connecting them.

Hu et al. [HFL12] create an oversegmentation using the Randomized Cuts algorithm [GF08] and then compute histograms capturing the distribution of the per-face features in each segment. The used features are AGD, GC, CF, SDF and SCs. Since SCs are

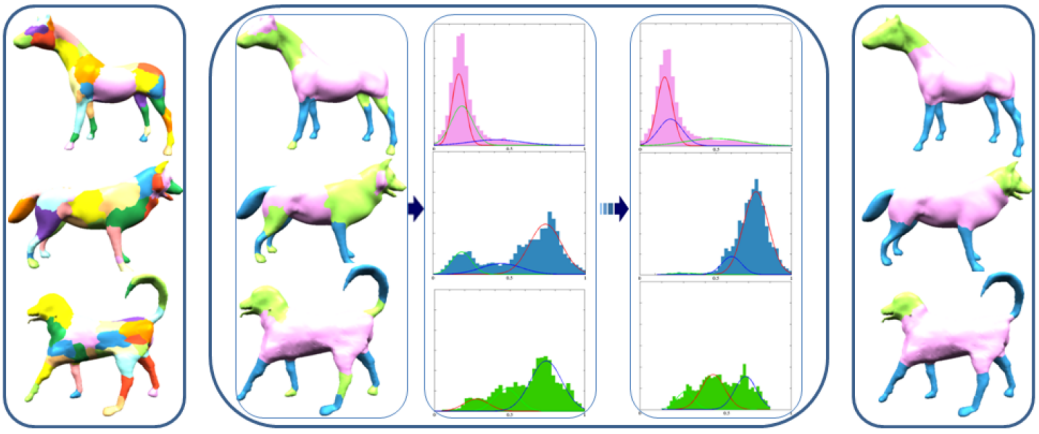

Figure 3.3: An example for unsupervised feature-based co-analysis [MXLH13]. Shapes are first over-segmented, then the segments are clustered using their face-level features. A statistical model is then used to refine the labeling of the faces.

histograms themselves, they are first clustered with a k-means algorithm and then binned into the histogram according to their associated cluster index. Each segment is then considered to be a point in a $\mathbb{R}^{H \times d}$ space, with H being the number of features and d being the number of histogram bins. They then perform subspace clustering [Vid11] to find multiple low-dimensional subspaces which correspond to the different labels. Finally, segment boundaries are refined using Fuzzy Cuts [KT03].

The approach of Meng et al. [MXLH13] is similar to that of Sidi et al. – each shape is oversegmented individually and the segments are clustered according to their per-face features. Then the segmentation is refined based on a statistical model. The four features used in the process are AGD, CF, SDF and SCs. The first two are considered intrinsic features, while the latter two are considered extrinsic features. This distinction is made because the intrinsic features are given a higher weight during the clustering step, while the extrinsic features are given a higher weight during the refinement step. For the clustering step, an affinity matrix is constructed based on the dissimilarity between the histograms of the feature distribution on the segments. The initial co-segmentation is then obtained by using the Normalized Cuts algorithm [SM00] on the affinity matrix. The refinement step is similar to the method of Sidi et al., but the optimization is performed iteratively instead of just once and, as mentioned before, the features are given different weights. The step-by-step process is depicted in Figure 3.3.

Wang et al. [WAV+12] suggest a semi-supervised approach that first computes a co-segmentation using any other unsupervised method and then refines that segmentation with the help of user input. Given a set of shapes that has been co-segmented, each shape segment is broken down into small super-faces that consist of several faces. The user can then define must-link or cannot-link constraints between pairs of super-faces. After setting the constraints, the super-faces are embedded using a spring system, represented by a fully-connected undirected graph. The nodes of the graph represent the super-faces and

are connected by different kinds of edges that also work as springs with a specified relaxed length. The relaxed length of *metric* edges is given by the feature-based dissimilarity of the connected super-faces. If a metric edge is longer or shorter than its relaxed length, it tries to get back to its relaxed length, thus exerting a force on the connected nodes. Constraints are represented as must-link or cannot-link edges. Must-link edges have a short relaxed length and only exert a force when they are longer than the relaxed length. On the other hand, cannot-link edges have a long relaxed length and only exert a force when they are shorter than the relaxed length. The force exerted by constraint edges is also much higher than that of metric edges. Computing the embedding based on the spring system leads to an objective function that needs to be minimized, defined as the sum of the energies stored at each edge. The super-faces are then clustered using the k-means algorithm in the embedded space. Furthermore, the system can suggest pairs of super-faces that will likely improve the segmentation if a constraint is assigned between them. This is done by identifying low-confidence points that are farther away from their assigned cluster center and closer to the other clusters.

3.1.2 Alignment-based Methods

Another type of method uses proximity as a cue to obtain a consistent co-segmentation. The idea is to globally align the shapes and consider parts that are close to each other as having a higher probability to have the same label.

Golovinskiy et al. [GF09] first construct a graph that represents the shape collection. The nodes of the graph correspond to the faces of the shapes and the edges can be divided into two types. The intra-mesh adjacency edges connect faces that are adjacent in their respective shapes and their weight is determined by the dihedral angle and the length of the edge connecting the two faces. Additional adjacency edges are added if the shape contains disconnected components because it is possible that they should be assigned the same label. For this purpose, the two components are sampled and for each pair of sample points closer than a certain threshold, an adjacency edge is added between the faces containing these points, with the weight of the edge depending on the distance. The second type of edges, the inter-mesh correspondence edges, connect faces of different shapes that are close to each other. For each pair of shapes, the shapes are globally aligned and sampled. For each sample point on the first shape, the closest point on the other shape whose normal vector points roughly in the same direction is considered. If the distance between the two points is under a threshold, the two faces containing the points are connected with a correspondence edge, with the weight depending on the distance. An example can be seen in Figure 3.4, where adjacency edges are shown in green and correspondence edges are shown in red. Finally, a hierarchical clustering scheme is used to obtain the co-segmentation. Each node of the graph begins as its own segment and in each step two segments are merged based on their edge weights. To reduce computation time, only the intra-mesh adjacency edges are used for merging in the early stage of the algorithm, which corresponds to an individual oversegmentation of each shape. Furthermore, the boundaries between segments are refined every few steps to increase the quality of the segmentation.

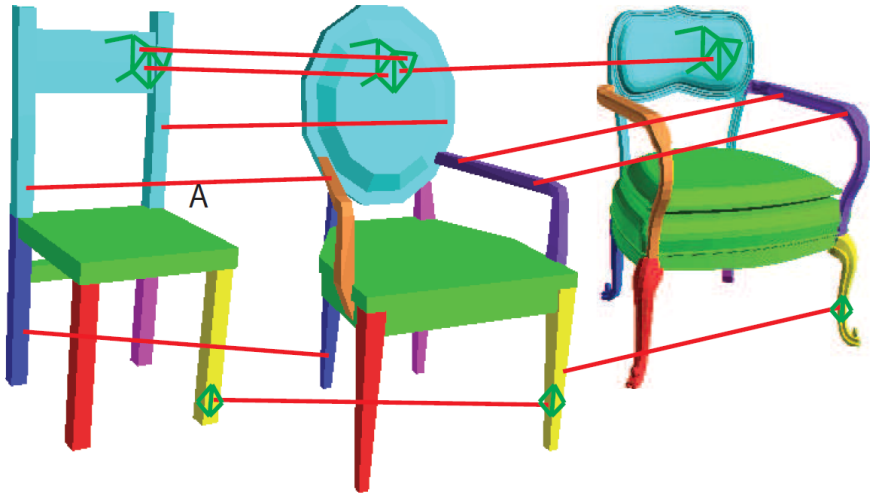

Figure 3.4: An example for alignment-based co-analysis [GF09]. Correspondences between faces of aligned shapes are created based on the distance between the faces. The correspondences, here depicted by red edges, are then used to assign a label to each face (here outlined by green edges).

However, because this method relies on rigid global alignment, it might fail in cases where there is a great variation of relative part scale across the shape collection. To solve this problem, Xu et al. [XLZ+10] first group the shapes according to their style based on the Anisotropic Part Scale (APS) of their parts. Each shape is represented by a graph where the nodes correspond to the Oriented Bounding Box (OBB) of the shape parts and the edges represent adjacencies between these OBBs. To cluster the shapes according to their styles, the APS style distance is used as a distance measure. For each possible part composition (which is any subset of parts of the shape), its APS style signature is computed. The signature is based on the scale of the part OBBs in each of the three major directions, as well as their linearity, planarity and sphericity (which are based on the ratios between their eigenvalues). A Laplacian matrix is constructed for each of these values and their eigenvalues are computed, yielding a total of $6n$ eigenvalues, where n is the number of parts in the part composition. Given two shapes, the distance between all possible pairs of part compositions with the same number of parts is computed, with the distance being the euclidean distance between the eigenvalues of the signature. The APS style distance between the two shapes is then simply the minimum distance between their part compositions. The actual co-analysis step is similar to the approach by Golovinskiy et al., but is only performed with shapes belonging to the same style cluster and using the part OBBs instead of faces as nodes of the graph. Finally, correspondences between parts of shapes belonging to different clusters are established using a deform-to-fit approach [ZSCO+08]. For a given pair of shapes, a search tree is constructed where each node represents a possible correspondence between a part of the first shape and a part of the

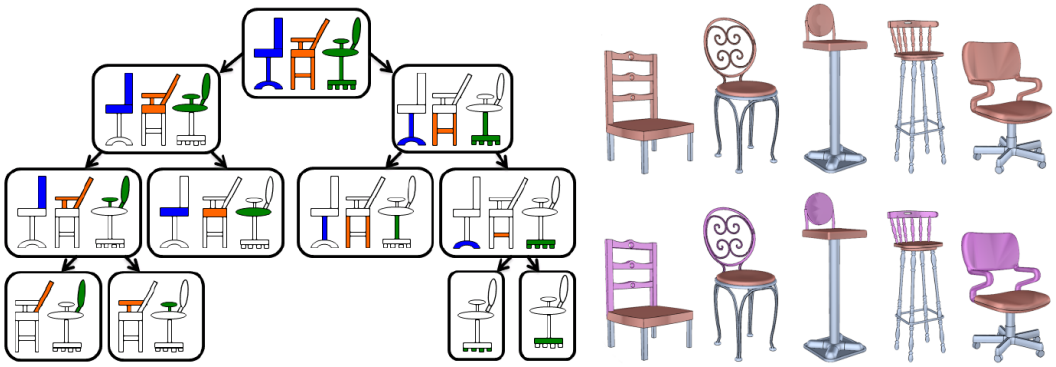

Figure 3.5: An example for structure-based co-analysis [VXZ+13]. A consistent co-segmentation is computed based on structural information, such as part hierarchy or relationships between parts.

second shape. When traversing the tree, at each node the first shape is transformed in such a way that the parts of the two shapes that are specified at the node match each other. For each subgraph the amount of overlap between the two shapes is computed to choose the best matching. Furthermore, to reduce the search space, the search tree is pruned based on the type of OBB and the kind of adjacency between adjacent parts.

3.1.3 Structure-based Methods

Structure-based methods try to solve the co-analysis problem by considering the hierarchies and relationships between individual parts of the shape. This is based on the idea that objects serving similar functions often share the same structure.

Van Kaick et al. [VXZ+13] first compute a graph representation of a shape where the nodes are the parts of the shape and the edges represent adjacency or symmetry relations between parts. The hierarchy of a shape is represented by a binary tree. The root node of a tree is the entire shape and contains all of its parts. For each lower level of the tree, the graph representing the parts of the current node is cut into two graphs, although parts connected by symmetry edges are considered a single atomic entity. Thus the leaves of the tree are either single parts or symmetry groups. A number of candidate hierarchies is created by taking samples of the possible trees for a shape: First, all binary decompositions of the root node's parts are computed and a number of samples is selected. This is done recursively for every node of the tree. Once that is done, all possible combinations of these partitions are considered to take a number of samples from to serve as the candidate hierarchies. These candidates are then clustered according to their tree-to-tree distance D_t which is defined as

$$\begin{aligned} D_t(h_i, h_j) = \min[&D_n(N_i, N_j) + \omega(D_t(h_{i1}, h_{j1}) + D_t(h_{i2}, h_{j2})), \\ &D'_n(N_i, N_j) + \omega(D_t(h_{i2}, h_{j1}) + D_t(h_{i1}, h_{j2}))] \end{aligned}, \tag{3.2}$$

where N_i and N_j are the root nodes of the trees i and j, h_{i1} and h_{j1} are their respective left children, h_{i2} and h_{j2} are their respective right children and D_n and D'_n are the node-to-node distances which correspond to the amount of translation and scaling necessary to align the axis-aligned bounding boxes enclosing all parts of the node.

For the clustering, the candidate trees are embedded into a normalized space using diffusion maps and clustered using agglomerative hierarchical clustering. After that, the medoid for each cluster is computed and for each shape one of two possible representative trees is selected. The first possibility is the tree that is closest to the medoid of its cluster, thus maximizing intra-cluster similarity. The second possibility is the tree that is closest to the cluster with the most central medoid, leading to increased inter-cluster similarity. The latter is only selected if the intra-cluster similarity remains above a certain threshold, otherwise the first possibility is chosen. Then, new candidate trees are sampled that are close to the selected representatives for each shape and the clustering and representative selection are repeated for a number of iterations. Finally, correspondences between the representative trees are computed by first matching the root nodes of two trees and then recursively matching their children according to the tree-to-tree distance. As an example, Figure 3.5 shows a number of chairs that have been matched based on their hierarchies.

Zheng et al. [ZCOAM14] also compute a graph representation of each shape where nodes are parts and edges signify either adjacency or symmetry relations. For the algorithm, groups of symmetric parts are once again treated as atomic entities. Any connected subgraph of this graph is considered a valid part composition (note that the authors refer to this simply as a *part* that consists of connected *segments* or *components*, but since this choice of terminology clashes with the meaning of *part* used in the majority of the referenced papers, we choose to call it a *part composition*). These part compositions are used to create a Pair Arrangement (PA) that consists of exactly two disjoint part compositions that are connected by at least one adjacency edge. The signature of a PA is defined as a normalized vector with 8 entries, each entry corresponding to the distance between the corner vertices of the axis-aligned bounding boxes of the part compositions. A similarity matrix $\mathbf{M_1}$ is constructed where the number of entries is the total number of PAs across all shapes. Each entry measures the similarity between two PAs based on their signature, but only if the two PAs belong to different shapes. $\mathbf{M_1}$ is then used to create another similarity matrix $\mathbf{M_2}$ that encodes the similarity between part compositions of different shapes. The similarity $\mathbf{M_2}(p_a, p_b)$ between two part compositions p_a and p_b is the sum of all similarities between PAs where p_a is part of one PA and p_b is part of the other PA. $\mathbf{M_2}$ is then used to create a third matrix $\mathbf{M_3}$ whose entries are the similarities between individual parts belonging to different shapes. The similarities are computed by considering every non-zero entry in $\mathbf{M_2}$. Given two part compositions p_a and p_b, p_a split into two sub-compositions a^1 and a^2, and p_b is split into b^1 and b^2. a^1 is then matched with b^1 and a^2 with b^2 (the ordering is determined by the maximum y-coordinate of the bounding boxes). There are 4 possibilities for a matching:

1. At least one sub-composition is not a connected graph, in which case the cut is discarded.

2. Both a^1 and b^1 are individual parts, then $\mathbf{M_3}(a^1, b^1) = \mathbf{M_3}(a^1, b^1) + \mathbf{M_2}(p_a, p_b)$.

3. If a^1 is an individual part while b^1 is a part composition (or vice versa), $\mathbf{M_2}(p_a, p_b)$ is distributed to $\mathbf{M_3}(a^1, b^{1i})$ according to the bounding box volumes of b^{1i}, with b^{1i} being the individual parts of b^1.

4. If both a^1 and b^1 consist of more than one part, they are both split based on the best possible match of sub-compositions (based on the entries of $\mathbf{M_1}$) and the procedure is continued.

Finally, a spectral clustering algorithm is used on $\mathbf{M_3}$ to obtain the co-segmentation.

Laga et al. [LMS13] combine geometric cues with structural information. They first compute a hierarchical graph of each shape. In the lowest level of the graph, nodes correspond to individual parts and edges signify adjacency and symmetry relations. The nodes of the next level of the graph are created by merging nodes of the lower level. Parts are merged if one of the following conditions is true, in the following order:

1. Parts A and B are merged to create a part C if C is geometrically similar to either A or B (thus merging a smaller part with a larger part).

2. A and B are symmetrical.

3. Parts are merged based on adjacency, with higher priority given to merge operations where the resulting part's geometric similarity with the parts that are merged is higher.

Each node is connected to its parent in the next graph level by an enclosure edge. In the next step, further edges are added between parts that correspond to different kinds of relationships: side-contact (the normal of the contact area is perpendicular to the axis of symmetry of one part), co-centricity (the axes of symmetry are parallel and close enough) and horizontal support (the normal of the contact area and the gravity vector of one part point in the same or opposite directions). To compute the similarity between two parts, both geometric and contextual information is considered. The geometric similarity K_{geo} depends the size of the parts, the three eigenvalues of the parts obtained by a PCA, and a histogram that measures the pairwise distances of uniformly sampled points on the surfaces of the parts. Given two shape graphs G_1 and G_2, the total similarity between nodes P_A and P_B is given by the p-order graph kernel $K^p(G_1, G_2, P_A, P_B)$:

$$K^p(G_1, G_2, P_A, P_B) = K_{geo}(P_A, P_B) \times \sum_{\substack{P_S \in \mathcal{N}_{G_1}(P_A) \\ P_Q \in \mathcal{N}_{G_2}(P_B)}} K_{rel}(e, f) K^{p-1}(G_1, G_2, P_S, P_Q), \tag{3.3}$$

with $K^0(G_1, G_2, P_A, P_B) = K_{geo}(P_A, P_B)$, $\mathcal{N}_G(x)$ being the neighboring nodes of node x on graph G, e and f being the edges connecting P_A to P_S and P_B to P_Q, and $K_{rel}(e, f)$ being 1 if e and f are the same type of edge and 0 otherwise. This kernel not only compares the geometric similarity between two nodes, but also all walks of length p

starting at the nodes. This measure of similarity is then used for supervised co-analysis. Based on a training dataset where each shape part has been assigned the correct label, a Support Vector Machine is trained for each label which is then used to classify the parts of the other shapes.

3.2 Parameterization of Shape Collections

The purpose of analyzing shape collections is to find the high-level semantic features which are shared by certain families of shapes while at the same time finding out in which ways individual shapes of a family differ. As an example, one can imagine a collection of seating furniture that contains chairs and benches. In this case, the distance between the legs of the shape might be a good feature for differentiating between chairs and benches since the legs of a bench are usually further apart than the legs of a chair.

Fish et al. [FAvK+14] compute a meta-representation of a shape family by looking at the relationships between parts. The input shapes are assumed to already be segmented and labeled. Each part of a shape is represented by an OBB. A set of unary and binary relations is defined for each part or pair of parts respectively. The unary relations encode the extends of the part OBB axes in relation to the axes of the entire shape OBB (the axes are ordered according to their alignment with the global x-, y- and z-axis). Binary relations between parts include scale (the relative difference between the axes of the two part OBBs), angle (the angle between the axes of the two part OBBs) and contacts (the relative placement of the intersection point between the two part OBBs). For each relation, a Probability Density Function (PDF) is computed using a 1D Kernel Density Estimator, with the bandwidth of the kernel estimated from the range of the data. The PDF for a relation shows the frequency of a certain setting for the relation within the shape family, as can be seen on the left side of Figure 3.6. The PDF can then be used in three ways. First, by starting with an initial shape and choosing one of the relations, the user can explore the shape collection by using the PDF as a guide. Clicking on a point in the PDF shows the shapes whose relations are closest to the chosen setting. Second, it can be used for constrained editing. If the user modifies a part by means of translation, scaling or rotation, the system can check whether the modified relations are valid based on the PDFs and can make adjustments if necessary. Finally, it is possible to use the PDF to edit multiple shapes at once by moving and scaling parts of the PDF itself.

Averkiou et al. [AKZM14] compute a 2D embedding of a shape family where similar shapes are clustered. First a number of templates is computed based on the method by Kim et al. [KLM+13]. These templates serve as a general approximation of the shapes where each shape part is represented by a box and they can be used to create a consistent co-segmentation of the shapes. This is done by aligning a shape with a template and minimizing an energy function based on which parts overlap. The unary term of the function encodes the amount by which the volume of the box increases if a face is assigned to it, while the pairwise term punishes adjacent faces being assigned to different boxes. Once each shape is segmented into parts, each part is represented by an axis-aligned bounding box. A configuration vector is computed for each shape

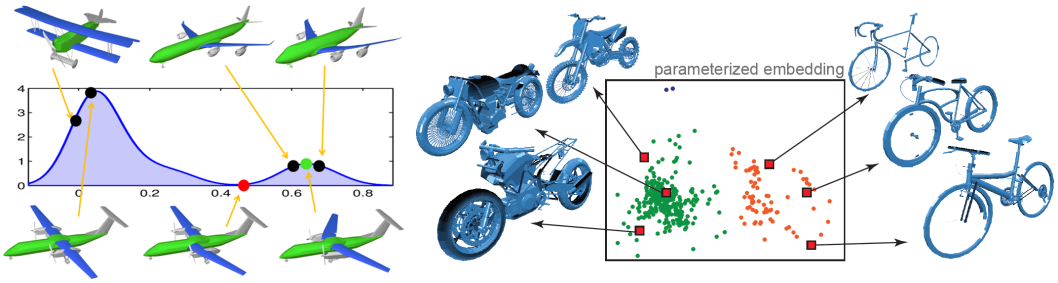

Figure 3.6: Analysis of shape collections. Left: The meta-representation of a shape family [FAvK+14] captures the distribution of defined relationships between parts (in this case the angle between the wings and fuselage of an airplane). Right: ShapeSynth [AKZM14] computes a 2D embedding of a shape family where similar shapes are clustered based on the arrangement of their parts.

that contains 6 entries for each part: the x-, y- and z-coordinates of the bounding box center relative to the center point of the shape, as well as the scale of each bounding box axis relative to the bounding box axes of the shape. The distance between two shapes is then the length of the difference between their configuration vectors. A pairwise distance matrix is computed and used to create a 2D embedding of the shapes with multi-dimensional scaling. To reduce computation time for very large shape collections, the distance matrix is only computed for a number of sample shapes and the embedding of the remaining shapes is interpolated based on its nearest neighbors. The data is then clustered using the mean-shift algorithm and each cluster is re-embedded on its own, creating a hierarchy of clusters. Such an embedding can be seen on the right side of Figure 3.6. The user can then explore the shape collection by clicking on a point in the embedded space, with preview models shown for each detected cluster. When clicking on a cluster, the system goes to the next level in the cluster hierarchy and shows the embedded space of that cluster. The system can also synthesize new models when clicking on an empty space. First, the configuration vector of the new shape is computed based on the embedding. The configuration vector is used to create an abstract shape made of boxes that satisfy a number of constraints, such as symmetry, contact and equal length in one or more axes. Finally, a full shape is synthesized by replacing each box with a part from the existing shapes that are most similar to that box.

Yumer et al. [YCHK15] encode the style of a shape within a family with semantic attributes. These attributes were found by conducting two user studies, using a number of training datasets from different shape families. In the first one, the 5 most prominent attributes were gathered for each shape family (e.g. fashionable, durable, comfy, feminine and active for a collection of shoes). In the second one, users were asked to rate these 5 attributes by comparing pairs of shapes. To find out how the attributes correlate to the style of a shape, each shape is abstracted by a number of handles [YK14]. These handles can be spheres, cylinders, cones or quadrics and are labeled so that handles of different shapes that are in correspondence have the same label. Each handle also has a number

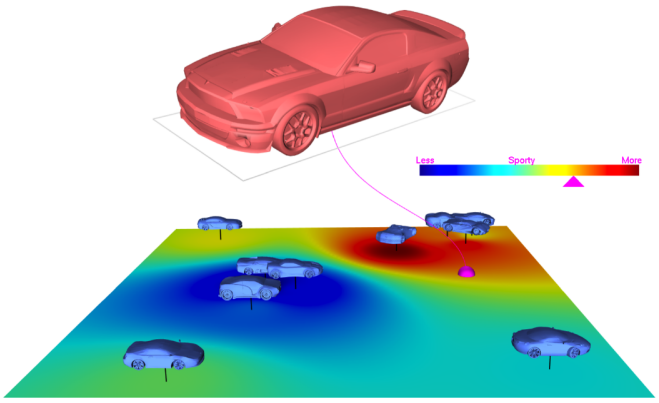

Figure 3.7: Exploration of a shape family with semantic attributes [YCHK15]. First, a correlation between semantic attributes (determined by a user study) and geometric attributes is computed. The shapes are then embedded into a 2D space based on the geometric attributes. A height map is used to show high and low values for a chosen semantic attribute (in this case the 'sportyness' of a car).

of features, including the position and orientation relative to the shape bounding box, the difference of these two features to every other handle of the shape, and also some features associated with the handle type, e.g. the radius of the sphere or the opening angle of the cone. The features are represented by a sparsely encoded vector, where each entry is preceded by an integer corresponding to the label of the handle to account for parts that are not present in all shapes. The mapping between the semantic attributes and the features happens in two steps. First, absolute values for the attributes need to be computed since the users in the study only rated the attributes of the shapes in comparison to other shapes. The attribute scores are modeled as a normal distribution and a system of linear equations is formed. By solving it in a least-squares sense, the absolute attribute scores are obtained. Second, a scoring function is constructed that allows the computation of the score $\tilde{f}_a(\mathbf{x_s})$ of attribute a for a new shape with features $\mathbf{x_s}$ that is not included in the training dataset:

$$\tilde{f}_a(\mathbf{x_s}) = \sum_{t \in \tau} \frac{\omega_t(\mathbf{x_s})}{\sum_j \omega_j(\mathbf{x_s})} f_a(\mathbf{x_t}), \tag{3.4}$$

with τ being the set of each training shapes, and $f_a(\mathbf{x_t})$ being the known attribute score of training shape t. $\omega_t(\mathbf{x_s})$ is a weight function that depends on the length of the difference between the feature vectors $\mathbf{x_s}$ and $\mathbf{x_t}$, excluding any features that are present in one shape, but not in the other. Finally, the system can be used for shape exploration. Each shape is embedded into a 2D space based on their feature vectors using locally linear embedding [RS00]. Then a scalar field is computed in the embedded space using the equation above. This can be visualized as a height map for a certain attribute, where each point on the map is colored according to the score of the attribute, as shown in Figure 3.7.

3.3 Shape Synthesis

Traditional modeling methods can be difficult to learn and the creation of detailed shapes often takes a lot of time even for experts. But with online model repositories growing larger every day, shape synthesis based on existing shapes has become a possible alternative for content creation.

Funkhouser et al. [FKS+04] allow the user to exchange parts between shapes by selecting a part and then searching for similar parts in a database. If the original shape is not already segmented into parts, the user can use an intelligent scissor tool that computes the best cut close the user's input, based on edge length (short cuts are preferred), dihedral angle between faces (cuts along concave regions are preferred) and face normal direction (a cut should preferably circle around the back side of the shape). The search for replaceable parts happens in two steps. First, the database is queried for shapes that are similar to the current shape. This is done by computing two voxel grids each for both the initial shape and the database shapes. For a shape A the voxel grid R_A is the rasterization of the boundary of the shape, i.e. a voxel is 1 if it contains part of the boundary or 0 otherwise. The second voxel grid E_A encodes the the squared euclidean distance to the closest point on the boundary for each voxel. The distance between two shapes A and B is then defined as $\langle R_A, E_B \rangle + \langle R_B, E_A \rangle$. Then the selected part is compared to parts of the matching shapes. This works in a similar manner as matching parts, only the first voxel grid is modified to have weight value w for voxels that contain a part of the shape boundary that belongs to the selected part. To exchange the selected part of the initial shape with a matching part from the database, the two parts are aligned using a variation of the Iterative Closest Point algorithm [BM92]. Finally, the new part is connected to the remainder of the shape using a greedy scheme that associates and connects each vertex along the cut boundary of the shape with a vertex of the cut boundary of the new part.

Kraevoy et al. [KJS07] first segment each shape into parts based on a measure of convexity and compactness. Convexity is defined as the average distance of each triangle to the convex hull of its part, while compactness is the area to volume ratio of the convex hull. At the beginning, each connected component of a shape is considered a part. For each part, a seed triangle is selected based on its distance to the convex hull of the part and the compactness of the tetrahedron formed by the triangle and convex hull center. A new part is grown from the seed triangle by adding adjacent triangles until adding another triangle increases the convexity error above a set threshold. The part growing step is repeated until there is no change in the number of parts created. A hierarchical segmentation can also be computed by using a low threshold for the convexity error and then merging adjacent parts based on an increased threshold.

Two shapes are then matched to see which parts can be exchanged. This is done by first finding a coarse one-to-one correspondence between components which consist of any number of parts of a shape. First, a midpoint graph is constructed for each shape by computing the midpoint of the boundary for each pair of adjacent parts and connecting them with straight edges. Given a one-to-one correspondence between components C_1 of

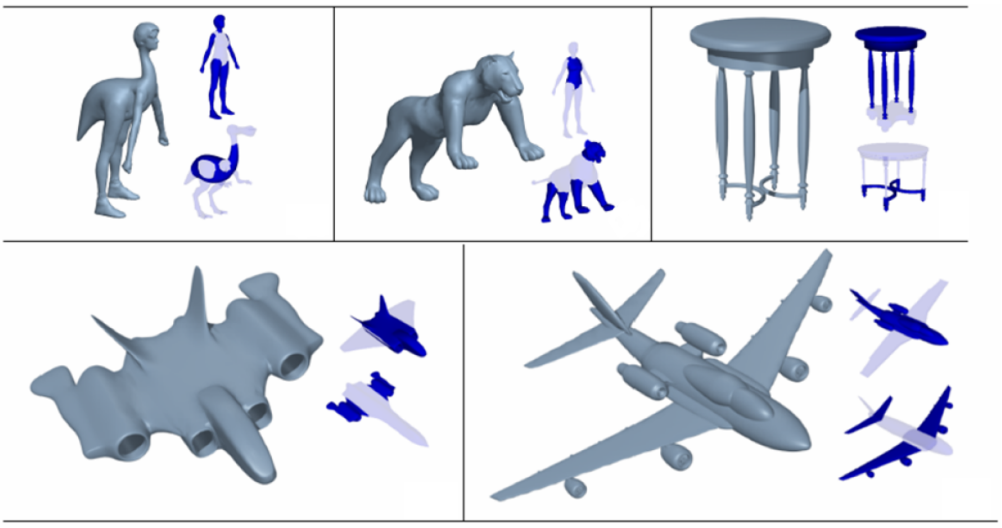

Figure 3.8: Model composition from interchangeable components [KJS07]. New shapes are created by starting with an existing shape and exchanging its parts with compatible parts from other shapes.

the first shape and C_2 of the second shape, their midpoint graph distance is defined as the sum of squared differences of the minimum midpoint distances between each pair of components (C_{1a}, C_{1b}) on the first shape to the corresponding pair of components (C_{2a}, C_{2b}) on the other shape. Together with a measure for component convexity (defined as the sum of the convex hull volumes of all components of both shapes) and a measure for volume similarity, a cost function is constructed and must be minimized to find the best matching of components. This is done using a stochastic local search scheme where the global minimum is found by first computing multiple local minima with a steepest descent minimization starting with different initial guesses. Two operations are allowed for each step of the minimization: the swap operation swaps the correspondences between two pairs of components and the regroup operation separates a part from one component and merges it with another component. Correspondences between smaller parts of the shape can then be found by recursively repeating the algorithm for the sub-components of each pair of matched components. Finally, to exchange two components the system uses the midpoints to align the swapped-in component with the remainder of the shape. A number of shapes created with this system can be seen in Figure 3.8.

Jain et al. [JTRS12] work with shapes that have already been segmented into parts. For every shape, each part is resampled into a point cloud and PCA is used to compute the eigentransformation from the global coordinate system to the local coordinate system of the part. Contacts between parts are detected and stored in an adjacency graph. For each shape, a part hierarchy is created, represented by a tree whose root node contains all parts. For each node, the major symmetry and the eigentransformation is computed and stored. To create child nodes, its parts are split by the major symmetry plane if it

exists, creating two child nodes (or three if the center of a part is close to the symmetry plane). In case there is no symmetry detected for a node, the parts are split based on the plane defined by the origin and x-, y- or z-axis of the local coordinate system. To ensure that a child node contains only connected parts, all n_k child nodes of the tree level k are split into their respective parts. The n_k largest parts are chosen as new child nodes and all smaller parts are merged with their smallest adjacent child node.

To exchange parts between two shapes, a matching between the shape hierarchies is computed. To to this, a new hierarchy for target shape is created. For a new level k, empty child notes are created for each parent node in the same way as on level k in the source hierarchy. Then the parts of each parent node are matched with the most similar part of the corresponding parent node of the source hierarchy, based on their positions in the local coordinate system. The part is then assigned to the child node in the target hierarchy corresponding to the child node of the matching part in the source hierarchy. Child nodes that remain empty are removed and the corresponding child node in the source hierarchy is merged with the closest child node. In the end, each part or connection of parts of the source shape has a direct correspondence with a part or connection of parts on the target shape. A new shape $S(w)$ is then created by blending between two shapes $S_1 = S(0)$ and $S_2 = S(1)$ with weight parameter $w \in [0, 1]$. Increasing from 0 to 1 causes the leaf nodes of S_1 to be exchanged one by one with the corresponding leaf nodes of S_2, with the nodes sorted by size so that the largest nodes are in the middle and the smallest nodes at the outsides. Aligning inserted nodes with the remainder of the shape is done using a spring system. Every node center and the contact points of every node are modeled as masses. Node centers are connected to its own contact points by springs that try to remain at their set length. Connections between nodes are modeled as zero-length springs between their contact points. Solving the spring system yields the best alignment of parts under the given contact constraints.

Zheng et al. [ZCOM13] find correspondences between parts that can be exchanged by looking at functional substructures of a shape. The term defined for the type of substructure that is used is Symmetry Functional Arrangement (SFARR). The sub-structure consists of a triplet of parts: two symmetric parts that are connected by a non-symmetric part. First, a graph abstraction is computed for each shape, where nodes correspond to parts and edges correspond to adjacency between parts. Edges can be either directed or undirected. Directed edges are called supportive edges and are used to connect parts where the first part is under the second part, thus supporting it. All other edges are undirected. SFARRs are found by detecting symmetries between parts and then consulting the shape graph to see whether they are connected to the same node, thus forming a valid SFARR. SFARRs can be one of three types depending on the edges of their respective sub-graphs. If the edges connecting the symmetric parts with the non-symmetric part are directed, the SFARR is of the placement type P if the edges point towards the symmetric parts, or of the support type S if the edges point towards the non-symmetric part. If the edges are undirected, the SFARR is of the embed type E. Two other attributes are also defined:

- A SFARR is stable if the projection of its center of gravity falls inside the convex

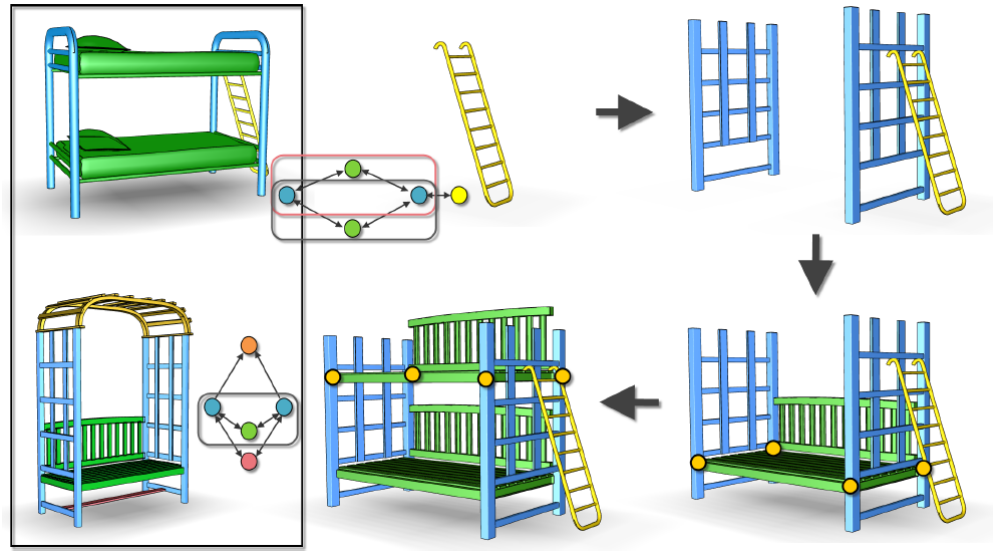

Figure 3.9: Smart Variations [ZCOM13]: symmetric substructures called SFARRs are detected and exchanged between shapes.

hull of its ground-touching vertices.

- A SFARR is coaxial if all of its parts are coaxial and the symmetric parts are cylindrical.

Two SFARRs are considered compatible for exchange if they are of the same type and shared the same attributes. Special consideration is given to parts that belong to more than one SFARR. Since it is possible that the common part serves a different function in each SFARR, a cluster is created which represents a connected sub-graph containing the SFARRs that share a common node. Additional attributes are assigned to those SFARRs. If the common part has both a placement and support role, it gets the $P + S$ attribute, and similarly a combination of embed and support yields $E + S$. A node with attributes Δ_i is then only replaceable by another node with attributes Δ_j if both of the following conditions are true:

- $\Delta_i \subseteq \Delta_j$ or $(\Delta_i \setminus \Delta_j) \bigcap S, P = \emptyset$
- Both nodes have the same number of contact slots (each cylindrical part has on slot at its contacting end and each cuboid part has two slots at the ends of its contacting edge).

The part replacement step begins with computing all SFARR clusters on the graph whose nodes are to be replaced. They are sorted according to their size and beginning with the first cluster, the SFARRs are replaced in a specific order, determined by their geometric compatibility with all the compatible SFARRs in the other graphs. The

geometric compatibility depends on the scale difference of corresponding nodes and the relative arrangement between the two SFARRs being compared, given by the difference of the subtended angle and perimeter of the triangle formed between the node centers of a triplet. SFARRs are replaced node by node, with the contact relations of a node to be replaced being used as constraints for the alignment (the contact slots mentioned above). Figure 3.9 shows how the parts of two shapes are combined to create a new shape.

CHAPTER 4

Framework for Shape Synthesis

This chapter describes our framework and the methods used to create a new shape from a collection of shapes that belong to the same family. Section 4.1 gives an overview for each stage of the system. The first stage, co-analysis, is described in more detail in Section 4.2. Section 4.3 describes how the collection is analyzed to find a small number of parameters that the user can then use to explore the collection as described in Section 4.4. Together they form the main contribution of this thesis. Finally, Section 4.5 explains how new shapes can be created by exchanging parts between existing shapes.

4.1 System Overview

The input is a shape collection $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ that contains m shapes M_i. The shapes are assumed to be of the same shape family (e.g. airplanes, chairs or vases). Although it is generally possible to use a collection that contains shapes from different families, it may result in unrealistic or implausible synthesized shapes. The process of creating a new shape from the input collection can be divided into four stages, which are also described in Figure 4.1:

1. Co-Analysis
2. Parameterization
3. Exploration
4. Shape Synthesis

To be able to exchange parts between shapes, it is first necessary to find correspondences between parts of different shapes. This is done in the co-analysis stage. Given the shape collection $\mathcal{M}$ as input, the co-analysis produces a co-segmentation of $\mathcal{M}$, which means that every part of each shape gets assigned a label l from a set of labels

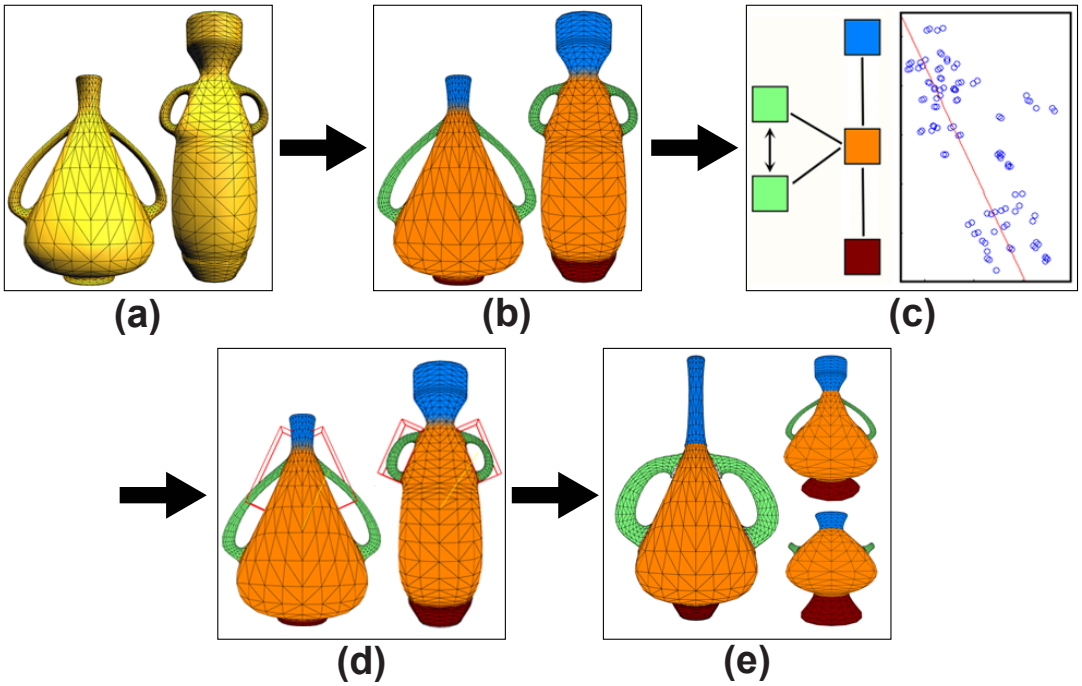

Figure 4.1: Overview of the proposed system. (a) The input is a set of shapes from the same family. (b) A consistent co-segmentation is obtained using co-analysis. (c) Each shape is abstracted as a graph, and the distribution of features between adjacent parts is analyzed. (d) The shape collection is explored by changing the features between adjacent parts. (e) New shapes are created by combining parts of existing shapes.

$\mathcal{L}$. Semantically, each label l corresponds to a common functionality inherent in every part labeled with l. As an example, the set of labels for an airplane might consist of the fuselage, wing, stabilizer and engine, although the name of each label can also be represented by a number. Practically, we assume all parts with the same label to be interchangeable.

Next we compute a small number of parameters that allow the user to explore the collection. We compute a graph representation of each shape, where nodes correspond to parts and edges connect parts that are adjacent. A number of relational features is computed for each edge, such as the distance, angle and relative scale between the adjacent parts. For each pair of labels (l, k), all edges connecting parts labeled l and k are embedded into a feature space based on the computed relational features. Principal Component Analysis (PCA) is performed to reduce the dimensionality, yielding one or two parameters that best describe how the relations between parts labeled l and k vary across the collection.

The computed parameters can then be used to explore the collection. Starting with an initial shape, the user can change the parameters for each relation, either directly by manipulating a slider, or by interacting with the part arrangements of the shape itself

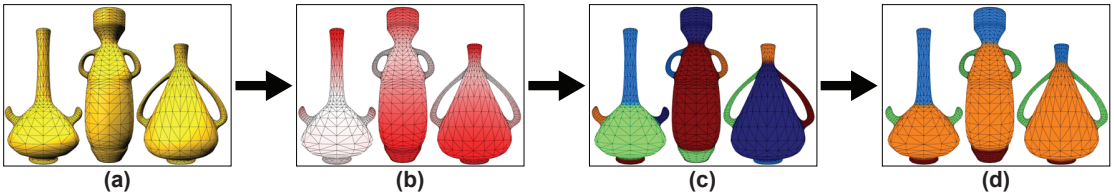

Figure 4.2: Overview of the co-analysis stage. (a) The input is a set of shapes from the same family. (b) Face-level features are computed for each shape. (c) Each shape is segmented individually. (d) Segments are clustered based on their feature distribution, producing a co-segmentation.

using a manipulator handle similar to those common in modeling programs. The two ways of interaction are coupled, so that changes are always reflected on both sides. The system then finds the shape whose relations most closely resemble the altered parameters, thus allowing the user to browse the collection.

Finally, a new shape is created by exchanging parts. Starting with an existing shape, the user can alter the parameters to find and select another shape that has the desired parts. The exchange operation removes all parts of the current shape with label l and replaces them with parts labeled l of the selected shape. Since the two shapes are not perfectly aligned, the new parts are transformed so they fit together with the remainder of the shape. The user can then proceed to replace other parts of the shape or even parts that have already been exchanged previously to create the desired shape.

4.2 Co-Analysis Stage

The aim of co-analysis is to produce a consistent co-segmentation of a collection of shapes. Given a set of shapes, each shape is segmented into parts, and each part is assigned a label, so that all parts across the collection with a certain label share the same functionality. Many different approaches have been proposed as a solution to the co-segmentation problem (see Section 3.1), but each one has its own advantages and disadvantages. Since there is no absolute best method and because co-analysis isn't the main focus of this thesis, we have instead opted for a more general three-step approach that allows the user to choose from a number of options for each step. The three steps are feature computation, individual segmentation and segment clustering. An overview of the co-analysis process can be seen in Figure 4.2.

4.2.1 Feature Computation

In the first step, a number of features is computed for each face of every shape. The chosen features are Average Geodesic Distance (AGD), Gaussian Curvature (GC), Conformal Factor (CF), Shape Diameter Function (SDF), Shape Context (SC), the face normal orientation and the geodesic distance from the base of the shape. Subsection 3.1.1

describes these features in more detail. Per default each feature has equal weight, but the user can assign different weights to each feature or even disregard some features entirely by giving them zero weight. Exporting them to and importing them from an external file is also possible. Since the computation of these features may take some time, especially for shapes with many faces, this option can be used to try out different algorithms in the following stages without needing to recompute the features.

4.2.2 Individual Segmentation

In the second step, each shape is segmented individually. While many shapes in online repositories already consist of multiple disconnected meshes, this is not always the case. For that reason we provide the user with a number of options to split a shape into parts. The first and second options are based on the 3D-NCuts approach [TLGG14]. First we compute the dual graph of the mesh, where nodes correspond to faces and adjacent faces are connected by an edge. Each graph edge is assigned a weight based on the dihedral angle between the faces and the length of the mesh edge connecting the faces so that short segment boundaries along concave regions of the shape are preferred. Then the Normalized Cuts algorithm [SM00] is used to create a partitioning of the graph, with each partition corresponding to a segment.

For the first option, the shape is immediately split into n segments based on the n largest eigenvectors of the graph Laplacian, while the second option applies the algorithm recursively to split each partition in two. The third and fourth options allow the user to import a segmentation from an external file, with the former taking the input as it is and the latter additionally splitting disconnected segments into separate segments. The fifth option is intended for shapes that already consist of multiple disconnected meshes and simply treats each of them as a separate segment. To account for cases where additional manual modifications are required, we provide a simple tool to create a manual segmentation, allowing the user to assign single faces or connected regions of faces to a segment by clicking on them. The segmentation resulting from any of these methods can also be exported to an external file.

4.2.3 Segment Clustering

In the last step, the segments are clustered to create the co-segmentation. Strictly speaking this is the step where the actual co-analysis takes place, since in the first two steps each shape is processed individually. Three options are provided for this step.

The first option is based on the work of Sidi et al. [SvKK$^+$11] and Meng et al. [MXLH13], although without the optimization step. For each segment, one histogram per feature is computed by adding the surface area of each segment face to the histogram bin corresponding to its feature value. Then a $n \times n$ affinity matrix $\mathbf{A}$ is constructed, where n is the total number of segments. To compute the distance between two segments, we must first compute the distance between their feature histograms. For that purpose

we use the Bhattacharyya distance [Bha43] d_B, which is defined as

$$d_B(h_i^f, h_j^f) = -\ln\left(\sum_{k=1}^{nbins}\sqrt{h_i^f(k)h_j^f(k)}\right), \tag{4.1}$$

with h_i^f and h_i^f being the histograms of segments i and j for feature f, and $nbins$ being the number of histogram bins, which is the same for each histogram. The distance d_S between a pair of segments is then given by the euclidean distance between their respective histograms for all features f:

$$d_S(i,j) = \sqrt{\textstyle\sum_f d_B(h_i^f, h_j^f)^2}\,. \tag{4.2}$$

By applying a Gaussian Kernel on the distance, the affinity A_{ij} between the two segments i and j is obtained:

$$A_{ij} = e^{\frac{-d_S(i,j)}{2\sigma^2}}, \tag{4.3}$$

with σ^2 being the variance of the distances d_S. The Gaussian Kernel works as a filter that results in a high affinity when the distance is small and a low affinity when the distance is large. The filter is modeled after a Gaussian function so that very large distances result in an affinity close to zero. The entries of the affinity matrix $\mathbf{A}$ are then the affinities A_{ij} between each pair of segments.

Then we use the Normalized Cuts algorithm on the affinity matrix to obtain a clustering of the segments. In the original algorithms by Sidi et al. and Meng et al., an additional optimization step is performed where a statistical model based on the computed features and clustered segments is created. This model is then used to re-estimate the probabilities of each face belonging to a certain cluster in order to refine the co-segmentation. This step is not included in our implementation since the individual co-analysis algorithms are not the focus of this thesis, although it would certainly improve the results obtained by this particular algorithm.

The second option is an extension of the first option based on the work by Laga et al. [LMS13], where we also incorporate structural information for computing the affinity matrix. In addition to the steps of the first option, we construct a hierarchical structural graph of each shape with nodes corresponding to segments (or groups of segments) and edges within the same level of the graph corresponding to adjacency and symmetry relations. The lowest level of the graph contains a node for each individual segment, while the nodes of the higher levels correspond to groups of adjacent or symmetric parts. Each upper level node also has an enclosure edge connected to each node of the next lower level whose corresponding parts it contains. The affinity between two segments is then based on the geometric similarity (using the distances between the feature histograms as in the previous option), as well as the structural similarity (based on the similarity of their neighbors in the graph and the type of edge they are connected with). A more comprehensive explanation can be found in Subsection 3.1.1. The third option is not an actual co-analysis algorithm, as it simply accepts the segmentation obtained from the

previous step as-is. This can be used in combination with loading the segmentation from an external file for quick use of the following stages.

As mentioned before, co-analysis is not the main focus of this thesis. As such we only provide a small number of options for each step. However, the system is designed to be easily extensible, allowing the development of additional methods for each step of the process, which is further explained in Section 5.2.

4.3 Parameterization Stage

The aim of the parameterization stage is to find a small number of parameters that allow the user to explore the shape collection. The idea is to use the spatial features, such as vertical and horizontal distance, angle and relative scale between adjacent parts as a guide, similar to the work of Fish et al. [FAvK+14]. However, using every spatial feature as a parameter would result in a large number of parameters, and some of them might not vary much across the collection, making them a bad choice for exploring the collection. Furthermore, correlations might exist between different features, resulting in a lot of empty regions in the exploration space if these features are used as separate parameters. Thus we use PCA to reduce the dimensionality of the parameter space, resulting in a small number of exploration parameters. The whole stage can be divided into three steps. First we create a graph abstraction of each shape, followed by computing the spatial features between each pair of adjacent parts. In the last step, the exploration parameters are found by analyzing how these spatial features vary for similar pairs of parts.

4.3.1 Shape Graph

The input is a set of shapes that has been co-segmented, meaning that every face of a shape has been assigned a label l from a set of labels $\mathcal{L}$ that is the same for the entire collection. First, we compute a structural graph for each shape. Nodes correspond to parts or groups of parts and edges correspond to relationships between them, specifically adjacency, symmetry and enclosure. The graph is a hierarchical graph with two levels. The top level contains one node for each label present in the shape, with each node corresponding to the union of all faces with that specific label. To create the bottom-level nodes, each top-level node is split into multiple nodes based on the connectedness of its faces. A breadth-first search is performed to find connected regions of faces with the same label. Each of these connected regions forms a part, and thus a node in the bottom level of the graph which is connected to its parent with an enclosure edge. An example of such a graph can be seen in Figure 4.3.

Problems might arise with shapes where there are many small disconnected parts that are close to each other and should really be treated as a single larger part. We use the bounding boxes of the parts to test whether or not they intersect. Since the bounding box is the smallest box that contains all points of its associated part, it is a useful approximation for the part itself. An axis-aligned bounding box, whose edges have the property that they are aligned with the axes of the global coordinate system, can

be easily computed based on the maximum and minimum coordinates of the contained points. The disadvantage is that the dimensions of the axis-aligned bounding box depend on the orientation of its corresponding part, which can be a problem since we do not assume the shapes to be globally aligned.

To use an Oriented Bounding Box (OBB) would a better solution for our purposes. The advantage of OBBs is that their edges do not have to be aligned with the axes of the global coordinate system, making them orientation-invariant. Computation of an OBB is a little more complex. We use PCA on the points of the part to compute their principal components. The first principal component corresponds to the direction in which the point coordinates have the greatest variance, while the second principal component maximizes the variance within the two-dimensional subspace that is orthogonal to the first principal component. The third principal component is orthogonal to both the first and second principal components, and thus they form an orthogonal basis which can be used as the local coordinate system of the OBB. The dimensions of the OBB can then be computed using the maximum and minimum coordinates of the points in the local coordinate system.

However, using the entire OBB might still be too inaccurate for parts that are not convex, so we perform an intersection test by creating a binary OBB tree for each part. The root node of the tree is the entire OBB of the part. Since it is a binary tree, each node has at most two child nodes, each corresponding to roughly one half of the OBB of its parent. To create the child nodes, the OBB of the parent node is split in half along the longest axis of the local coordinate system of the OBB and each face is assigned to one half based on its centroid position. The OBB of the new child node is then computed from the assigned faces. If one half does not contain a face, no child node is created for that half. The nodes are recursively split until the leaf nodes of the tree only contain a single face each. To test whether or not two parts intersect, an intersection test is performed between their trees, starting with the root nodes. If the two nodes intersect, the first tree is traversed down, testing for intersection at each node. Once a leaf node has been reached, the second tree is traversed, again checking for intersection at each node. If there is an intersection between two leaf nodes, then the two parts are considered to be intersecting and are merged to form a single part.

Next we test for adjacency between parts with different labels. First we iterate over all mesh edges to find faces that are adjacent and have different labels. When such a pair of faces is found and an adjacency edge between the nodes of their corresponding labels does not yet exist, such an adjacency edge is created. Furthermore, we take the boundary vertices between the two segments, meaning all mesh vertices that are incident to at least one face of each label, to compute the contact center and a number of contact vectors.

The contact center is computed by taking the average of all boundary vertices, while the contact vectors are obtained by performing PCA on the boundary vertices. The contact vectors are sorted according to their eigenvalues in descending order. We will also refer to the eigenvector corresponding to the smallest eigenvalue as the *contact normal* since it serves as the normal vector of the plane that best approximates the contact area.

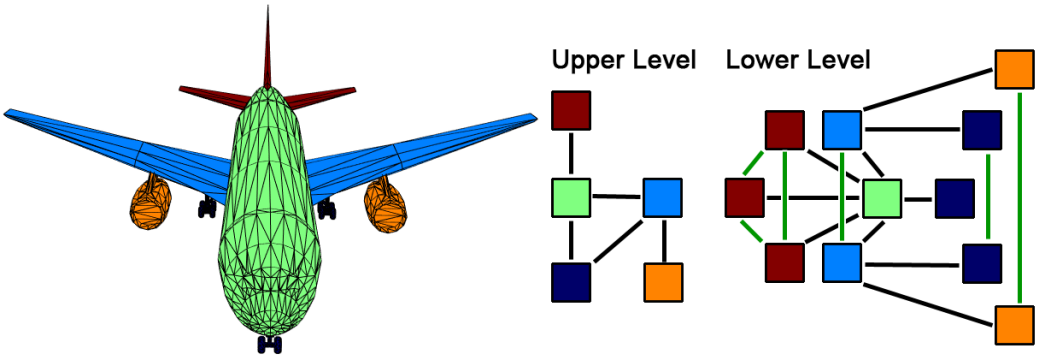

Figure 4.3: A co-segmented mesh with its corresponding shape graph. Each node of the upper level of the graph corresponds to the set of all parts with a specific label, here represented by the color. The lower level of the graph has one node for each part of the shape, each with the same color as its parent node in the upper level. Black edges correspond to adjacencies between parts, while green edges represent symmetries between parts.

Additionally, we compute the diameter of the contact area based on the range of the boundary vertex coordinates in the subspace formed by the first and second contact vectors. The contact center, vectors and diameter are stored with the adjacency edge and will be useful later in aligning parts of different shapes.

We also must consider shapes that consist of multiple meshes. To test for adjacency, we once again use OBB trees, but in addition to the intersection tests between the nodes of the trees, there is one more intersection test between the faces of the leaf nodes, using Möller's triangle intersection algorithm [Möl97]. A single intersection is enough to create an adjacency edge between the nodes corresponding to the parts, but we continue to search for all intersections between the parts and use the intersection points to compute the contact center and contact vectors for the adjacency edge. The adjacencies in the bottom level of the shape graph are then transferred to the top level – if there is an adjacency edge between two bottom-level nodes and there is no adjacency edge connecting their parent nodes, such an edge is created between the parents.

To complete the shape graph, symmetry relations are computed between nodes of the bottom level, but only between nodes that share the same parent node. Symmetries are detected using an approach similar to the one in [JTRS12]. A translational symmetry transform between two parts can be created using the vector between their centers. To test for a reflective symmetry, the vector between the centers is used as a normal vector to create a symmetry plane at the midway point between the part centers. To create a rotational symmetry transform, a third part is selected to fit a circle to the center of the three parts, thus defining a rotation. To check whether two parts are symmetrical in regards to a transform, one part is transformed with the symmetry transform. If the distance between the center of the transformed part and the center of the other part is under a threshold, and their eigenvectors are similar in length, a symmetry edge is

created between the nodes corresponding to the two parts.

4.3.2 Spatial Features

Next we compute a feature vector $\mathbf{x}$ for each bottom-level adjacency edge. We have chosen four features to capture the spatial arrangement between each adjacent pair of parts p and q: the vertical distance d_y and horizontal distance d_{xz} between part centers, the angle θ between parts and the bounded relative scale s between parts. Let $\mathbf{d} = (d_x, d_y, d_z)^T$ be the distance vector between part centers $\mathbf{p_c}$ and $\mathbf{q_c}$. Then the vertical distance d_y is simply the y-coordinate of $\mathbf{d}$. The horizontal distance is the euclidean distance between the part centers in the xz-plane: $d_{xz} = \sqrt{d_x^2 + d_z^2}$. Both d_y and d_{xz} are normalized by dividing by the diameter of the bounding box obtained by merging the bounding boxes of the two parts. We have chosen not to use the distance in x- and z-direction as separate features because it would necessitate the assumption that the orientation of all shapes is globally aligned, which is not always the case with models taken from an online database. We do, however, assume that the shapes are in upright position. To compute the angle θ between parts, we select the OBB axis of each part that is best aligned with the contact normal of the adjacency edge and compute the angle between them.

The relative scale $s_r(p, q)$ between two parts p and q is obtained by dividing the OBB diameter of p by the OBB diameter of q. However, the scale is not symmetric since $s_r(p, q) = 1/s_r(q, p)$, so we define a consistent ordering based on the part labels. Let L_p and L_q be the labels of parts p and q, then the ordered relative scale s_o is given by

$$s_o(p, q) = \begin{cases} s_r(p, q), & \text{if } L_p < L_q \\ s_r(q, p), & \text{otherwise} \end{cases}. \tag{4.4}$$

Furthermore, we want to limit the range of the feature. We define a mapping s with

$$s(p, q) = \begin{cases} 1 - \dfrac{1}{s_o(p, q)}, & \text{if } s_o(p, q) > 1 \\ s_o(p, q) - 1, & \text{otherwise} \end{cases}. \tag{4.5}$$

This mapping limits the range of the feature to the interval $[-1, 1]$ such that $s(p, q) = 0$ means that the OBB diameter of the two parts have equal length, positive values mean that p is larger than q and negative values mean that p is smaller than q.

4.3.3 Exploration Parameters

For the last parameterization step, we define a set of relations $\mathcal{R}$ for the shape collection. A relation R_{lk} is contained in the set only if there exists at least one shape in the collection whose graph contains an adjacency edge between nodes labeled l and k. Since the adjacency edges are undirected, the relations are symmetrical, i.e. $R_{lk} = R_{kl}$. For each relation, we want to compute one or two exploration parameters, the number depending on the amount of correlation between the spatial features. This is done by using PCA to reduce the dimensionality of the feature space spanned by the spatial features. For each

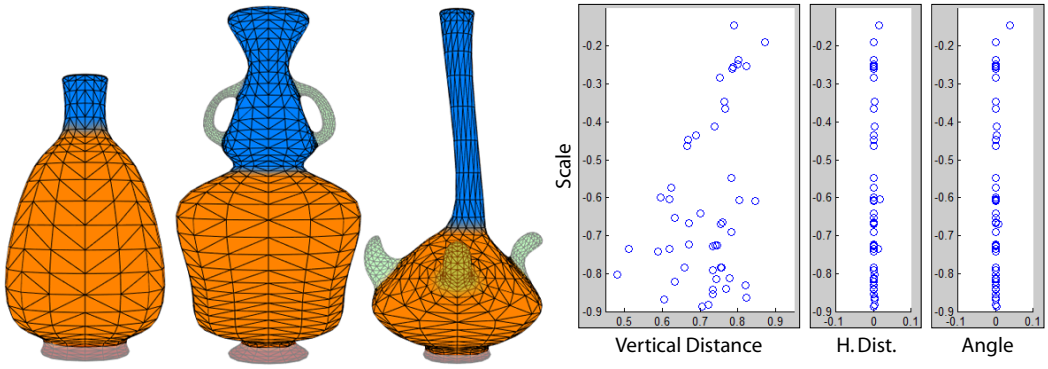

Figure 4.4: The distribution of the shapes in the vase datset based on the spatial features between the neck and body of the shapes. The scale is plotted on the y-axis, while the vertical distance, horizontal distance and angle are plotted on the x-axis. There is almost no variation in the latter two features, thus they are not very useful as parameters for exploring the shape collection.

relation R_{lk}, all corresponding adjacency edges are embedded into a feature space based on the centered feature vectors $\hat{\mathbf{x}}_{\mathbf{pq}} = \mathbf{x}_{\mathbf{pq}} - \bar{\mathbf{x}}_{\mathbf{lk}}$, where $\mathbf{x}_{\mathbf{pq}}$ is the feature vector of the adjacency edge between parts p and q, and $\bar{\mathbf{x}}_{\mathbf{lk}}$ is the mean of the feature vectors of all adjacency edges between parts labeled l and k.

Then we compute each feature's variance for every relation and remove the features with a variance smaller than a threshold. The reasoning is that for any given relation there might be some features that do not significantly contribute to the variability within the shape collection. An example of this can be seen in Figure 4.4, where the horizontal distance and angle between the neck and body of the shapes in the vase dataset are the same for most shapes. Thus they are not useful parameters for browsing the collection. We set the threshold to 0.005 which was chosen empirically.

Next we perform PCA for each relation since there is the possibility of correlation between features. If two features are strongly correlated, it is possible to use a single parameter for these features instead of using one parameter for each. Figure 4.5 shows an example in which the horizontal distance and relative scale between the legs and seat of the shapes in the chair dataset are correlated. Since the legs of a chair are usually placed at the corners of the seat, their horizontal distance to the center of the seat depends on the size of the seat. On the other hand, for chairs that only have a single leg, the leg is usually placed at the center for balance and needs to be larger to carry the weight of the seat.

Performing the PCA yields the principal component coefficient matrix $\mathbf{V}_{\mathbf{lk}}$ and the variances $\boldsymbol{\lambda}$ for each relation, as well as a principal component score $\mathbf{y}_{\mathbf{pq}}$ for each adjacency edge between parts p and q. The rows of $\mathbf{V}_{\mathbf{lk}}$ can be seen as the axes of the principal component space expressed as vectors in the feature space. The scores $\mathbf{y}_{\mathbf{pq}}$ express the coordinates of the adjacency edge in the principal component space and are used as our

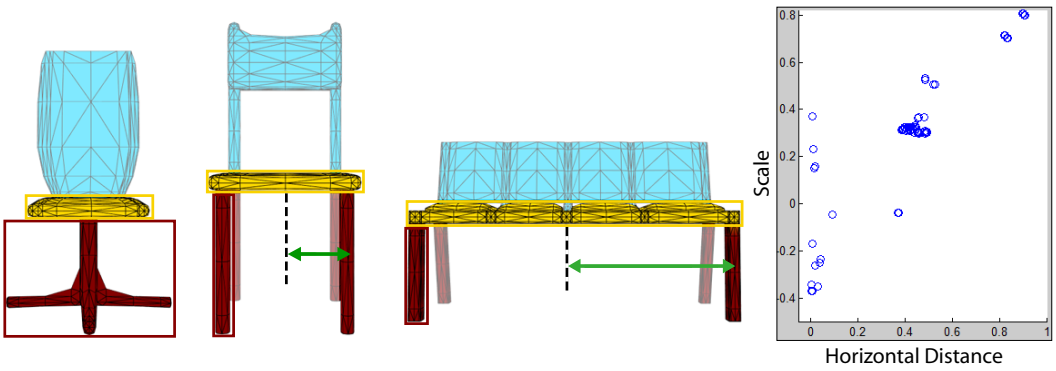

Figure 4.5: The spatial features between adjacent parts can be correlated. In this case there is a correlation between the horizontal distance and scale between seat and legs of the chair.

parameters for the exploration of the shape collection. They are obtained by transforming the centered feature space coordinates $\hat{\mathbf{x}}_{\mathbf{pq}}$ with the corresponding transposed coefficient matrix $\mathbf{V}^T_{\mathbf{L_p L_q}}$:

$$\mathbf{y_{pq}} = \mathbf{V}^T_{\mathbf{L_p L_q}} \hat{\mathbf{x}}_{\mathbf{pq}} \,. \tag{4.6}$$

The principal component variances $\boldsymbol{\lambda}$ are simply the variances of the data for each axis of the principal component space. They are used to decide the number of parameters used for the exploration. We choose to use the two principal components with the largest variances if the variance of the second component is larger than half the variance of the first component. Otherwise we choose only the first principal component since the second component does not significantly contribute to the variation of the relation. In cases where we need two principal components but two features have been discarded because their variance is too small, we do not use the principal components as parameters. Instead we use the two remaining features directly since it is more intuitive to control them separately. Since we only need one or two principal components, we only store at most two columns of $\mathbf{V_{lk}}$ and $\mathbf{y_{pq}}$ with each relation and adjacency edge respectively. Finally, we also store the mean feature vector $\bar{\mathbf{x}}_{\mathbf{lk}}$, as well as the minimum and maximum of the features and scores of each relation for later use.

4.4 Exploration Stage

Using the coefficient scores $\mathbf{y}$ as parameters, it is now possible to explore the shape collection. Starting with an existing shape, the user selects an adjacency edge a_{pq} by picking a pair of adjacent parts p and q. It is then possible to alter the parameters $\mathbf{y_{pq}}$ of the chosen edge a_{pq}. This can be done in two ways. First, it is possible to change the parameters directly by interacting with one or two sliders, each corresponding to an entry of $\mathbf{y_{pq}}$. The second way of altering the parameters is by interacting with the shape itself, using a manipulator tool to transform the selected part and then recomputing the

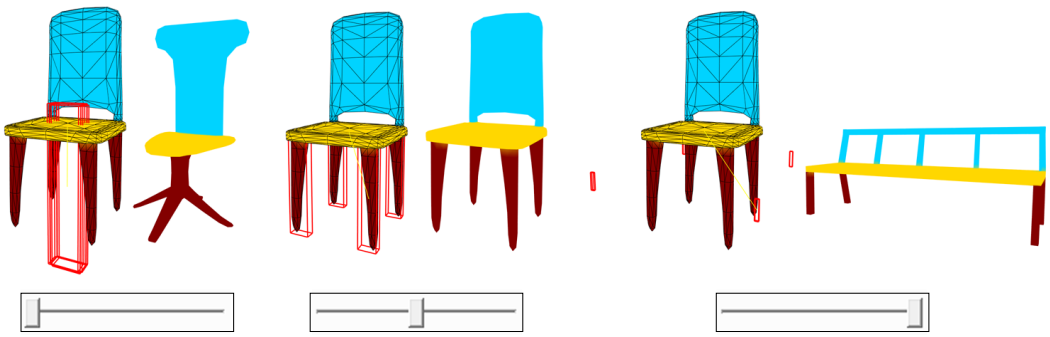

Figure 4.6: Exploration of the shape collection by directly changing the parameters of the chosen adjacency edge. In each image, the original shape is shown on the left, while the shape that best fits the altered parameters is shown on the right. In this example, the parameters between the legs and seat of the chair are altered by interacting with the slider, and the bounding boxes of the chair's legs are updated accordingly.

parameters from the new transformation. A coupling between the parameter sliders and the visual representation ensures that any changes are always reflected on both sides. This is explained in Subsection 4.4.1. Once the altered parameters are computed, we search for the shape in the collection that best fits these parameters, which is described in Subsection 4.4.2.

4.4.1 Visual Representation

Interacting with a slider is a simple and quick way of changing a parameter value. The problem with this kind of interaction is that the effect of increasing or decreasing the parameters is not immediately apparent to the user. Because of that, we have designed a visual aid in the model viewer that is coupled to the parameters. The selected part p as well as all parts that are connected to p by a symmetry edge are visually represented by their OBBs. Additionally, a line connecting the OBB centers of p and q is shown to represent the selected adjacency edge. To show the effect of altering the parameters, the position, rotation and scale of the OBB of p is updated accordingly. An example of this can be seen in Figure 4.6. The spatial arrangements between the legs and seat of the shown chair are are altered by interacting with a slider. The OBBs of the legs are updated according to the altered parameters, and reflect the variations of this particular relation shown in Figure 4.5. The shape containing the adjacency edge that best fits the altered parameters is also shown next to the original shape. To apply the parameter changes to the visual representation, it is first necessary to compute the altered spatial features from the altered parameters.

Let $\mathbf{y'_{pq}}$ denote the altered parameters of the adjacency edge a_{pq}, and let $\mathbf{\Delta y_{pq}} = \mathbf{y'_{pq}} - \mathbf{y_{pq}}$ be the difference between the altered and original parameters. Using $\mathbf{\Delta y_{pq}}$ and the coefficient matrix $\mathbf{V_{L_pL_q}}$ we can compute the change in terms of the original

spatial features:

$$\mathbf{\Delta x_{pq}} = \mathbf{V_{L_p L_q}} \mathbf{\Delta y_{pq}}, \tag{4.7}$$

with $\mathbf{\Delta x_{pq}} = (\Delta d_y, \Delta d_{xz}, \Delta\theta, \Delta s)^T$. We can apply these changes to the original arrangement between p and q to represent the change visually. With $\mathbf{d} = (d_x, d_y, d_z)^T$ again as the distance vector between part centers $\mathbf{p_c}$ and $\mathbf{q_c}$, we can get the translation vector $\mathbf{\Delta d}$ to the new position of $\mathbf{p_c}$ by

$$\mathbf{\Delta d} = \Delta d_y \begin{pmatrix} 0 \\ \frac{d_y}{|d_y|} \\ 0 \end{pmatrix} + \Delta d_{xz} \begin{pmatrix} \frac{d_x}{d_{xz}} \\ 0 \\ \frac{d_z}{d_{xz}} \end{pmatrix}. \tag{4.8}$$

To apply the change of the angle between the parts, it is first necessary to determine the axis of rotation. Let $\mathbf{u_p}$ and $\mathbf{u_q}$ denote the OBB axes of p and q that are best aligned with the contact vector of a_{pq}. Since we want to transform the OBB of p in relation to the OBB of q, the axis of rotation can be obtained by the cross product $\mathbf{u_q} \times \mathbf{u_p}$. Of course this is only possible when $\langle \mathbf{u_q}, \mathbf{u_p} \rangle \neq 0$, in which case the OBB of p is rotated by $\Delta\theta$ around the computed axis of rotation using the contact center as the pivot. Otherwise we do not visualize the rotation, since we have no means to determine the axis of rotation.

Finally, to apply the change in scale, we first compute the new relative scale s'_r from $s' = s + \Delta s$:

$$s'_o = \begin{cases} \frac{1}{1-s'}, & \text{if } s' > 0. \\ 1 + s', & \text{otherwise} \end{cases}, \qquad s'_r = \begin{cases} s'_o, & \text{if } L_p < L_q. \\ \frac{1}{s'_o}, & \text{otherwise} \end{cases}. \tag{4.9}$$

Since s'_r denotes the new relative scale of p in relation to q, it is first necessary to scale p to be of the same size as q before applying the new scaling s'_r. Thus the OBB of p is scaled by $\frac{s'_r(p,q)}{s_r(p,q)}$.

To keep the symmetry between symmetric parts, we also apply any parameter change $\mathbf{\Delta y}$ to the parameters of relevant adjacency edges of symmetric parts. Let P and Q denote the set of parts symmetric to p and q respectively. $\mathbf{\Delta y}$ is then added to the parameters of all adjacency edges $a_{p_i q_j}$ where $p_i \in P, q_j \in Q$. The OBBs of the parts p_i displayed in the model viewer are also transformed based on the changed parameters. Naturally it is also possible to change the order of p and q when selecting an adjacency edge. Since the adjacency edge is undirected, the altered coordinates $\mathbf{y'}$ stay the same. The only thing that changes is the visualization: because p and q are swapped in the equations above, the visualization shows the transformed OBB of q in relation to p.

It is important to note that the altered features can possibly take on values that exceed the minimum or maximum features present in the shape collections, or even result in invalid values such as a negative distance or scale. This can happen because at most the first two principal component scores are changed, while the others remain the same. For this reason we store the minimum and maximum features for each relation during the parameterization so we can restrict the visual representation to those values.

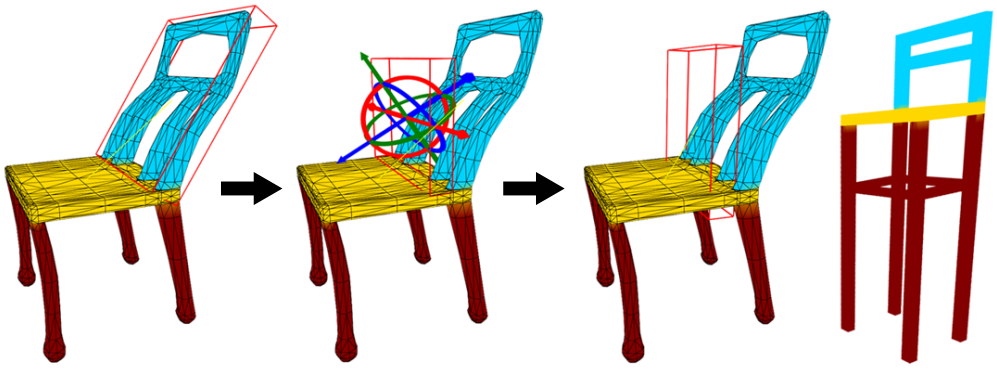

Figure 4.7: It is also possible to change the parameters by interacting with the shape itself. The bounding box of the selected part can be transformed using a manipulator tool. Once the transformation has been accepted, the altered parameters are computed from the transformed bounding box and the visualization is updated to stay in line with the new parameters. The shape shown on the right is the shape which best fits the new parameters.

The other method of changing the parameters is by interacting with the shape in the model viewer. After selecting an adjacency edge by picking a pair of adjacent parts p and q, the user can activate edit mode to freely translate, rotate and scale the OBB of p. This can be done with a manipulator tool similar to those in conventional modeling programs. Once the user accepts the transformation of the OBB, a feature vector $\mathbf{x}_{\mathbf{p'q}}$ is computed, with p' denoting the transformed OBB. We center the feature vector by subtracting the mean feature vector $\bar{\mathbf{x}}_{\mathbf{L_{p'}L_q}}$ of the corresponding relation $R_{L_{p'}L_q}$ and compute the new principal component score $\mathbf{y}_{\mathbf{p'q}}$ using Equation 4.6. If the new score falls outside the range of scores of the relation, the values are clamped so they do not exceed the minimum and maximum values. Finally, the OBB is transformed once more based on $\mathbf{y}_{\mathbf{p'q}}$ to ensure that the visual representation stays in line with the parameters. Figure 4.7 shows an example of this.

4.4.2 Finding The Best Fit

The altered parameters are then used to find shapes in the collection that best fit these parameters. There are two options for this search: finding the pair of parts in the collection that best satisfy the parameters of the currently selected adjacency edge, or incorporating all adjacency edges of the current shape in the search to find the most fitting shape in the collection. The first case is simple. Let a_{pq} be the currently selected adjacency edge with altered parameters $\mathbf{y}'_{\mathbf{pq}}$, and let $A_{L_pL_q}$ denote the set of adjacency edges between parts labeled L_p and L_q (note that a_{pq} is also included in this set). Then we can find the best fitting adjacency edge a^* by

$$a^* = \underset{a \in A_{L_pL_q}}{\arg\min} \left\| \mathbf{y}'_{\mathbf{pq}} - \mathbf{y}_{\mathbf{a}} \right\| . \tag{4.10}$$

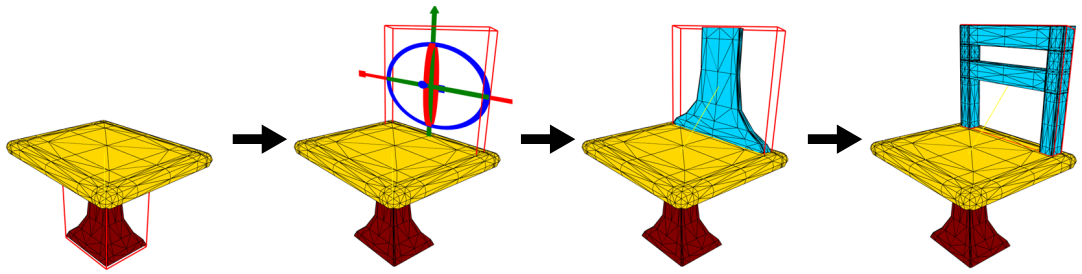

Figure 4.8: Existing parts can be copied and assigned a different label. Such a new part can then be used as a proxy to explore the collection or to swap in parts from other shapes.

In order to take all adjacency edges into account when searching for the shape that best fits the altered parameters, we must first consider how to deal with cases where the cardinality of parts varies across the collection. For every shape M_i we first compute the average parameter vector $\bar{\mathbf{y}}^{\mathbf{i}}_{\mathbf{lk}}$ for each relation R_{lk} given by

$$\bar{\mathbf{y}}^{\mathbf{i}}_{\mathbf{lk}} = \begin{cases} \frac{\sum_{a \in A^i_{lk}} \mathbf{y_a}}{|A^i_{lk}|}, & \text{if } |A^i_{lk}| > 0 \\ \frac{\mathbf{y^{max}_{lk}} - \mathbf{y^{min}_{lk}}}{2}, & \text{otherwise} \end{cases}, \tag{4.11}$$

with A^i_{lk} being the set of adjacency edges of shape M_i between parts labeled l and k, and $\mathbf{y^{max}_{lk}}$ and $\mathbf{y^{min}_{lk}}$ being the maximum and minimum values for $\mathbf{y_{lk}}$. Furthermore, we extend each vector $\bar{\mathbf{y}}^{\mathbf{i}}_{\mathbf{lk}}$ by adding another entry z^i_{lk}, which is the weighted number of adjacency edges between parts labeled l and k given by

$$z^i_{lk} = \begin{cases} \omega \frac{|A^i_{lk}| - A^{min}_{lk}}{A^{max}_{lk} - A^{min}_{lk}}, & \text{if } A^{max}_{lk} - A^{min}_{lk} > 0 \\ 0, & \text{otherwise} \end{cases}, \tag{4.12}$$

with $A^{min}_{lk} = \min_{i=1\ldots n} |A^i_{lk}|$, $A^{max}_{lk} = \max_{i=1\ldots n} |A^i_{lk}|$ and $\omega \geq 0$ being a weight parameter that decides how important the number of adjacency edges is in finding similar shapes. The average of the altered parameters $\bar{\mathbf{y}}'_{\mathbf{lk}}$ is computed in the same manner. The shape M^* that is closest to the altered parameters can then be found by

$$M^* = \arg\min_{M_i \in \mathcal{M}} \sum_{R_{lk} \in \mathcal{R}} \|\bar{\mathbf{y}}'_{\mathbf{lk}} - \bar{\mathbf{y}}^{\mathbf{i}}_{\mathbf{lk}}\| \,. \tag{4.13}$$

Since we can also take the number of adjacency edges into account when searching for the best fit, it is necessary to provide the user with a way to add or remove adjacency edges of the current shape. Either operation is only allowed when the number of adjacency edges $|A^i_{lk}|$ remains within the interval $[A^{min}_{lk}, A^{max}_{lk}]$ after increasing or decreasing it. The first step is to select an adjacency edge by picking two adjacent parts p and q. An

adjacency edge can be deleted by pressing the corresponding button. Creating a new adjacency edge is done by creating a new part p' that is then connected to q. This can be achieved by copying the OBB of p and using the manipulator tool to move it to a new location. Once confirmed, the vertices and faces of p are copied and placed at the new location using the transformation of the OBB. To obtain an approximation for the contact center and contact vectors of the new adjacency edge, we intersect the edges of the OBB of p' with the planes of the OBB of q and vice versa. The points of intersection are then treated as the boundary vertices between the two parts to compute the contact center and vectors. If the OBB of p' is placed so that there are less than 3 intersections, the new location is not accepted and the user can either alter the transformation of the OBB or cancel the creation of the new part.

Finally, it is also possible to change the label of a copied part, thus allowing the creation of adjacency edges that do not occur in the current shape. While the new part still has the geometry of the part that was copied, it can be used as a proxy to find shapes with similar part arrangements and to swap in parts with the same label from other shapes. Figure 4.8 shows an example of this process. The shown shape from the chair dataset only consists of a seat and a leg. The leg part is copied, transformed, placed on top of the seat and assigned the label of a back part. It is then used as a proxy to find and swap in a back part from a different shape.

4.5 Shape Synthesis Stage

The final stage of the shape synthesis process is the exchange of parts between shapes. Once the shape M^* with the desired parts has been found by using the process explained in Section 4.4, the user can replace the part p of the currently selected adjacency edge a_{pq} with a corresponding part of M^*. First, the faces and vertices of p are removed from the shape. The system then picks a part p^* from M^* with the same label as p as a replacement and adds its vertices and faces to the current shape. Since the two shapes are not aligned, it is necessary to transform p^* so it fits together with the remainder of the shape. This is done with the help of the contact centers and contact vectors of the adjacency edge a_{pq} and its corresponding edge $a_{p^*q^*}$ in M^* that connects p^* with a part q^* that has the same label as q. The aim is to find the transformation that best aligns the two contact areas, which can be done by aligning the contact vectors.

As the first step, scaling is applied to p^* based on the relative scale between the contact diameters of a_{pq} and $a_{p^*q^*}$. Next we make sure that the contact normals of a_{pq} and $a_{p^*q^*}$ point in the same direction as the the vectors pointing from the contact center to the OBB of p and p^* respectively. If this is not the case, we reverse the direction of the contact vectors. Then we check if the two contact vectors point in the same direction. If not, q^* is mirrored across the plane defined by the contact center and contact normal of $a_{p^*q^*}$ (in this case the order of the vertex indices of the faces of p^* also needs to be reversed to preserve the direction of the face normals). To align the contact vectors, we first compute the angle between the contact normals and use their cross product as the axis of rotation to align them. Next we align the transformed first contact vector of $a_{p^*q^*}$

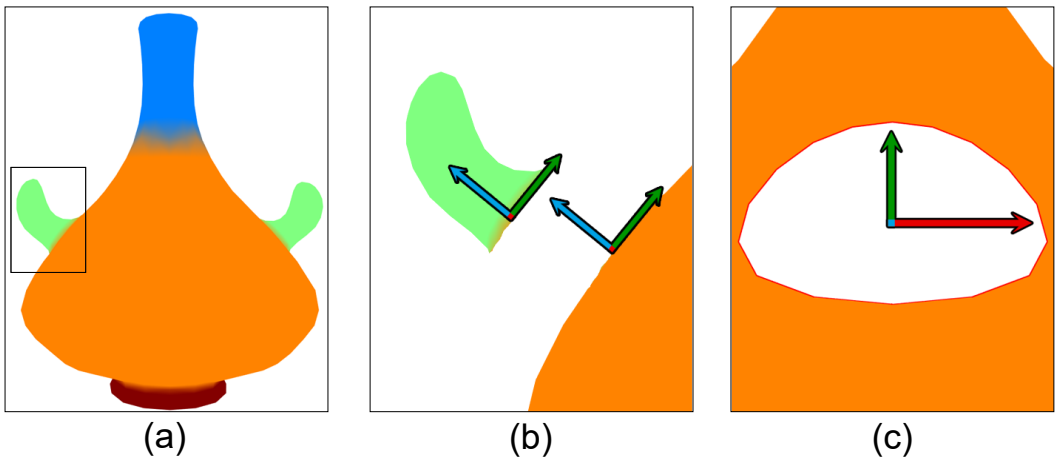

Figure 4.9: Part alignment between the handle and body of a shape from the vase dataset. The arrows represent the contact vectors, with the first and second vectors shown in red and green respectively, and the contact normal shown in blue. (a) The full shape, with the area of interest highlighted. (b) Side view of the contact area. (c) Frontal view of the contact area.

with the axis corresponding to the first contact vector of a_{pq}. This is done by using the aligned contact normal as the axis of rotation and rotating the first contact vector of $a_{p^*q^*}$ so that it aligns with either the first contact vector of a_{pq} or its inverted vector, whichever requires the smaller rotation. Finally, p^* is translated by the vector between the contact centers of a_{pq} and $a_{p^*q^*}$. An illustration of this process can be seen in Figure 4.9. Also, to keep symmetry between parts, the process is repeated for all parts that are symmetric to p.

Once the parts have been exchanged, the user can repeatedly go through the exploration and shape synthesis stages, by selecting another adjacency edge, exploring the shape collection and exchanging further parts. Naturally, it is also possible to exchange or explore with parts that have been swapped in from other shapes.

This system serves as a simple demonstration of how new shapes can be created by swapping in parts from other shapes. Depending on how well the swapped-in parts fit with the remainder of the shape, the results might look far from perfect without further refinements. For example, when the parts of the initial shape are connected, removing these parts results in holes which would need to be filled (an example of how this can be achieved can be found in [FKS+04]). Other considerations include how to handle the exchange between parts where the topology of the contact areas differs, or constraints of plausibility like requiring the shape to be stable with regards to gravity. But since shape synthesis is not the main contribution of this thesis, the proposed system suffices for purposes of demonstration.

CHAPTER 5

Implementation

This chapter gives an explanation about the implementation of our proposed framework. Section 5.1 details the base framework and libraries that were used. Section 5.2 explains how the co-analysis stage was designed to be modular and easily extensible. In Section 5.3 we explain the data structure we use to store the shape graphs of each shape. Finally, Section 5.4 contains details about the user interface and methods of interaction with the application.

5.1 Base Framework and Libraries

The application is written in C# , using Helix 3D Toolkit with SharpDX as a base. Shapes are loaded from files containing vertex positions and faces of a triangle mesh. The supported file types are OBJ and OFF, but the application can be easily extended to support further file extensions as long as the file contains the necessary information to create a triangle mesh. The loaded shapes are stored as objects of the `TriangleMesh` class [Kol05], which uses a halfedge data structure. This data structure makes it easier and more efficient to navigate through the neighborhood of a vertex, edge or face of the mesh, which is needed in many operations present in the application. One disadvantage is that the data structure is very strict about the layout of the mesh. For example, an edge can never have more than two adjacent faces. This can be a problem when using shapes from a model repository, since such standards are not enforced there.

The application also uses MATLAB for several operations, thus a working copy of MATLAB is required to run the application. Whenever necessary, data is passed to MATLAB where it is processed and then the results are queried. Most notably, the application uses the k-means algorithm of MATLAB for computing the SC feature and the PCA algorithm of MATLAB to compute the contact areas and exploration parameters for each adjacency edge. The co-analysis stage also uses a MATLAB implementation of the Normalized Cuts algorithm [CYS04].

For the contact and symmetry detection, as well as the computation of OBBs and OBB trees of parts, we use an existing implementation developed during a previous student project. Finally, the application uses an open source implementation of a C# priority queue, primarily used in Dijkstra's algorithm for the computation of the geodesic distances on the triangle mesh.

5.2 Co-Analysis Extendability

As mentioned in the previous chapter, the co-analysis stage of the application is developed to be both modular and easily extensible. The co-analysis process consists of three steps: feature computation, individual segmentation and segment clustering. The classes and methods used in this stage are contained in the `CoAnalysis` library.

During the first step, various features are computed for each face of every shape. The features used in this application are explained in Subsection 3.1.1. The `FeatureDescriptor` class contains all methods for feature computation. With the exception of SCs, each face-level feature is represented by a `double`-type value and the features for each shape are stored in a jagged array. The first dimension contains an entry for each feature, while the second dimension contains an entry for each face of the shape. Since the size of the array depends on the number of features, it is possible to add more features without changing any of the other modules of the co-analysis stage. SCs are the one exception to the way features are represented in the application. Since a SC is a 2D histogram, additional processing is needed to bring it into the same format as the other features. We use the same method as Hu et al. [HFL12] by clustering the SCs of all shapes using k-means and then using the cluster index of each SC as the new feature.

During the individual segmentation, the shapes are segmented into regions of connected faces, with each face being assigned a label corresponding to their segment. The algorithm used for the segmentation can be chosen during runtime from a number of options. Further segmentation algorithms can be developed to increase the number of options available. For this purpose, the `CoAnalysis` library contains an interface with the name `ISegmentationAlgorithm`. It contains a single method

```
int[] Segment( TriangleMesh mesh,
               int nSegments
             );
```

which must be implemented when creating a class that extends the interface. The two inputs are the `TriangleMesh` to be segmented and the number of segments that should be created. The output is an integer array containing the corresponding segment label for each face of the shape. To include the new algorithm as an option for the segmentation step, an instance of the class only needs to be added to the list of options at application startup.

The third step, segment clustering, is implemented in a similar manner. During this step, the segments of all shapes are clustered, assigning a new label to each segment so that that all segments with the same label belong to the same category.

Once again, a number of options is available for this step and more can be added by extending the `IClusteringAlgorithm` interface. This interface has two methods. `bool RequiresFeatures()` returns true if the implemented algorithm requires the computation of face-level features. This is used during first step of the co-analysis stage. Since not all co-analysis algorithms depend on face-level features, the feature computation can be skipped in those cases. The second method is

```
IList<int[]> Cluster( IList<TriangleMesh> meshList,
                      IList<double[][]> featureList,
                      IList<int[]> segmentList,
                      int nClusters,
                      double[] featureWeights,
                      bool autoClusters
                     );
```

which performs the clustering. The inputs are, in order: the list of shapes as objects of the `TriangleMesh` class, the list of features in the format described earlier, the list containing the labels of each shape computed in the previous step, the desired number of clusters, an array containing the weight for each feature and a boolean that determines whether the number of clusters should be estimated by the algorithm or not. Of course it is not required for the implementation of a clustering algorithm to use all of these inputs during the algorithm. The output is a list containing an integer array for each shape, with each array containing the new segment labels of each face.

5.3 Shape Graph Data Structure

In the parameterization stage we compute a graph abstraction of each shape. Such a graph is represented by the `MeshGraph` class. The parts of a shape correspond to nodes in the graph, which are represented by the `MeshGraphNode` class. Relationships between parts, such as adjacency or symmetry, correspond to edges in the graph and are represented by the `MeshGraphEdge` class. The data structure works as follows: A `MeshGraph` stores a list of `MeshGraphNode` objects for each level of the graph. For the parameterization stage we only ever use two levels, but the class is also used in one of our co-analysis algorithms where it can have more levels. Each `MeshGraphNode` stores the index of the shape it belongs to, its OBB, the indices of its faces and the corresponding label. A list of `MeshGraphEdge` objects that are incident to the node can be used to quickly find adjacent or symmetric nodes. For each `MeshGraphEdge` we store the two incident `MeshGraphNode` objects and the type of the edge which can be adjacency, symmetry or enclosure. An enclosure edge represents a parent-child relationship between nodes of different levels. In this case, the parent node is always saved as the second node of the edge. For adjacency edges of the lower level we also store the contact center, contact normals and the diameter of the contact area, as well as the exploration parameters $\mathbf{y}$.

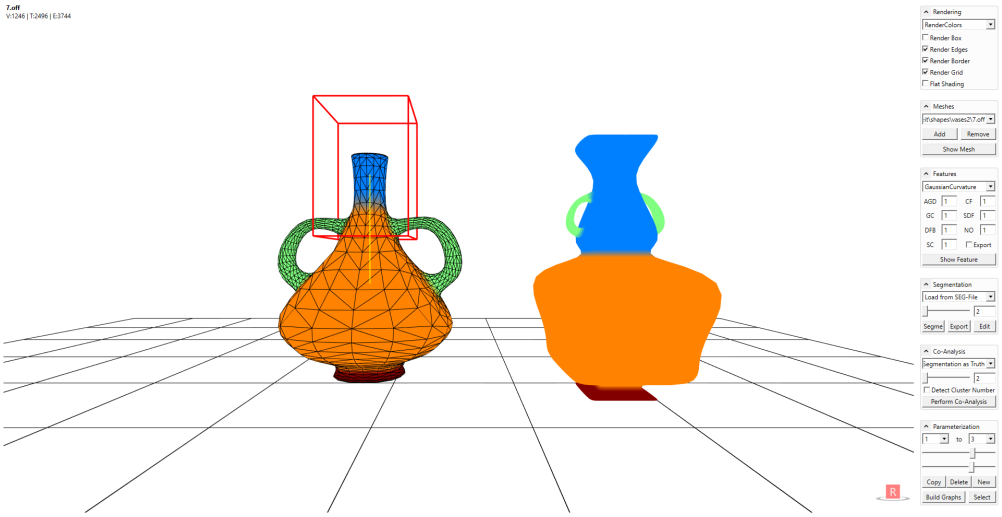

Figure 5.1: The user interface of the application. It consists of two regions. The left side is the model viewer which displays the currently selected shape and a preview of the shape that best fits the altered parameters. The right side contains controls for the different stages of the system.

This allows us to quickly find specific nodes and edges for different purposes. For example, disconnected parts with the same label can be found by iterating over the enclosure edges of the corresponding node in the upper level of the graph, while parts that are adjacent or symmetric to a specific part can be found by iterating over its edges of the corresponding type. Since the number of edges incident to each node is quite small in general, this is fast and efficient. Also, since we allow the user to make changes to the shape graph of the currently selected shape in order to explore the collection, a copy of the shape graph is created when selecting a shape from the collection. All changes by the user are only applied to this copy to ensure that the original shape remains the same and can be chosen when searching for the shape that best fits the altered parameters.

5.4 User Interface

The user interface, which can be seen in Figure 5.1 is divided into two regions, with the model viewer on the left side and the user controls on the right side. The model viewer displays the currently selected shape and during the exploration stage also a preview of the shape that best fits the exploration parameters altered by the user. The controls are organized into panels corresponding to the different stages of the system. The first panel contains some options related to the rendering, allowing the user to choose which elements to display in the model viewer. The second panel contains the files of the shape collection. Here, files can be added or removed from the current collection and the shape to be displayed in the model view can be selected.

The third, fourth and fifth panel are relevant for the co-analysis stage. The *Feature* panel is used for setting the weights of the face-level features and to visualize the features on the currently displayed shape. It is also possible to export the computed features which are saved in a file with the extension FTR and placed in the same folder as the shape file. If such a file is present during the feature computation step, it will be automatically used to import the features instead of computing them. The *Segmentation* panel allows the user to choose a segmentation algorithm from a number of options. The chosen algorithm can be applied to the displayed shape in order to visualize, export or edit the resulting segmentation. The *Co-Analysis* panel contains a list of available clustering algorithms for the third step of the co-analysis process. When performing the co-analysis, the parameters of the previous two panels are taken into account, using the chosen feature weights and the selected segmentation algorithm.

The sixth panel is used for the parameterization, exploration and shape synthesis stages. Once the co-analysis stage has concluded, the parameterization can be performed by pressing the corresponding button. Two drop-down boxes allow the selection of an adjacency edge by first picking a shape part from the first box and then picking an adjacent part from the second box. With the sliders it is possible to alter the parameters of the adjacency edge directly. The panel also offers controls for copying and deleting parts, as well as creating new proxy parts by copying a part and assigning a new label to it. Finally, the *Select* button exchanges the part currently selected for a corresponding part of the shape that best fits the altered parameters, allowing the creation of a new shape.

Interaction is also possible in the model viewer by clicking on the displayed shape. The `MeshGeometryModel3D` class provided by Helix 3D Toolkit with SharpDX automatically performs a hit test when the user clicks somewhere in the model viewer. However, that functionality is not enough for our purposes. For that reason we extended its functionality, creating the `HittableMeshGeometryModel3D` class. This class additionally stores the index of the last triangle hit by clicking on the shape, also taking into account whether the Shift or Ctrl keys were pressed during the click.

The result of such an interaction depends on the stage of the system. While in the exploration stage, left-clicking allows the selection of the shape part the hit triangle belongs to. Holding Shift while clicking on the shape selects the adjacency edge between the selected first part and the part that was hit, provided that these two parts are actually adjacent. When editing the current segmentation, simply left clicking changes the label of the hit triangle. Clicking while holding the Shift key changes the label of the whole connected segment the hit triangle belongs to. Finally, holding Ctrl while clicking works similar to a color picker, allowing the user to select the label of the hit triangle to use for the other two operations.

CHAPTER 6

Results

In this chapter we perform a number of test. All of the tests are performed on a machine with an Intel Core i5 processor with four 3.40GHz cores, 8 GB RAM and a AMD Radeon HD 6950 graphics card. The test datasets are taken from the Princeton Shape Benchmark [SMKF04], the Shape COSEG Dataset [Wan] and 3D Warehouse [Tri].

We use two different versions of the large COSEG chair dataset which consists of a variety of chairs and benches. The *test set* contains the shapes numbered 0 to 188 and 400 to 632, for a total of 275 shapes (note that not all numbers exist in the dataset). The number of faces of the shapes in this set range from 336 to 20928, with an average of 3522. The *full set* also contains the remaining 125 shapes in addition to the shapes of the test set. Some of these shapes have a higher face count, resulting in a range of 336 to 31456 faces for the set, with an average of 4611.

The large COSEG vases dataset contains a total of 300 shapes. The types of shapes in the set include vases, cups and buckets. The average face count of the set is 2533, ranging from 832 to 6528. Finally, we also use two smaller datasets. The candelabra set from the COSEG benchmark consists of 28 shapes, with 9952 to 10004 faces, the average being 9996. The plane dataset consists of 15 plane and fighter jet shapes that have been put together from shapes found on 3D Warehouse and the Princeton Shape Benchmark. The face count for this set ranges from 480 to 28592, with 5482 faces on average. Images of the used datasets can be found in Appendix A.

The labeling of the shapes is obtained using our co-analysis stage. For the actual tests we use the option to load an existing segmentation from a file, usually provided together with the dataset. The shapes of the plane dataset do not come with an existing labeling, so we created our own segmentation using our manual segmentation tools. We also tested different combinations of our implemented segmentation and clustering algorithms. While the resulting co-segmentations can be used successfully in the following stages of the framework, our implementation lacks the refinement step of the original algorithm and thus leads to a higher error rate in the labeling. This can lead to unsatisfactory

results, so to demonstrate the full capabilities of the other stages, we use an existing segmentation.

6.1 Parameterization and Exploration Results

This section presents the results of a number of tests that were made for the exploration stage. We perform tests for both single-relation and multi-relation exploration. Note that for these tests, we assume that each shape is already correctly segmented and labeled. Numbers for the parameterization of the shape collections can be found in Appendix B.

The parameterization of the shape collection needs to be done only once before exploration of the collection becomes possible. The performance depends on the number of shapes and the number of faces for each shape. We recorded the duration of the parameterization for the test datasets. On the test system, the parameterization of the test chair set takes 1 minute and 16.91 seconds. The full chair set contains more shapes including some shapes with a higher face count and thus the time it takes to parameterize the set is higher, taking 2 minutes and 32.48 seconds. The parameterization of the vases dataset takes 1 minutes and 14.48 seconds. Finally, it takes 26.79 seconds for the candelabra dataset and 11.14 seconds for the plane dataset.

Since the computation of the shape that best fits the altered parameters is fast, exploration of the shape collection can then be done in real time even for large sets. As a test, the exploration of the combined collection of the vases and full chair datasets (for a total of 700 shapes) results in a framerate of $30 - 40$ frames per second on the test system even when considering all relations in the computation of the shape that best fits the altered parameters.

6.1.1 Single-relation Exploration

In this subsection we present the results of our single-relation exploration tests, meaning that only the currently selected relation is taken into account when searching for the shape that best fits the altered parameters. Figure 6.1 shows the results for the relation between the backrest and the seat of the shapes in the test chair dataset. The additional shapes of the full set are not used in order to demonstrate how the parameters change depending on the variability of the set, which will be shown later. We use the provided segmentation files for the labeling of the parts.

The analysis of this dataset yields the first two principal components as exploration parameters. The shapes shown in Figure 6.1 are ordered roughly according to their position in the exploration parameter space. This means that change to the first parameter is shown on the horizontal axis, while change to the second parameter is shown on the vertical axis. Increasing the first parameter results in an increase for the relative scale and vertical distance, but a decrease for the angle and horizontal distance of the backrest relative to the seat. Increasing the second parameter results in an increase of all features.

As a result, similar shapes are closer to each other than dissimilar shapes. For example, angled backrests can be found mostly on the left side of the parameter space,

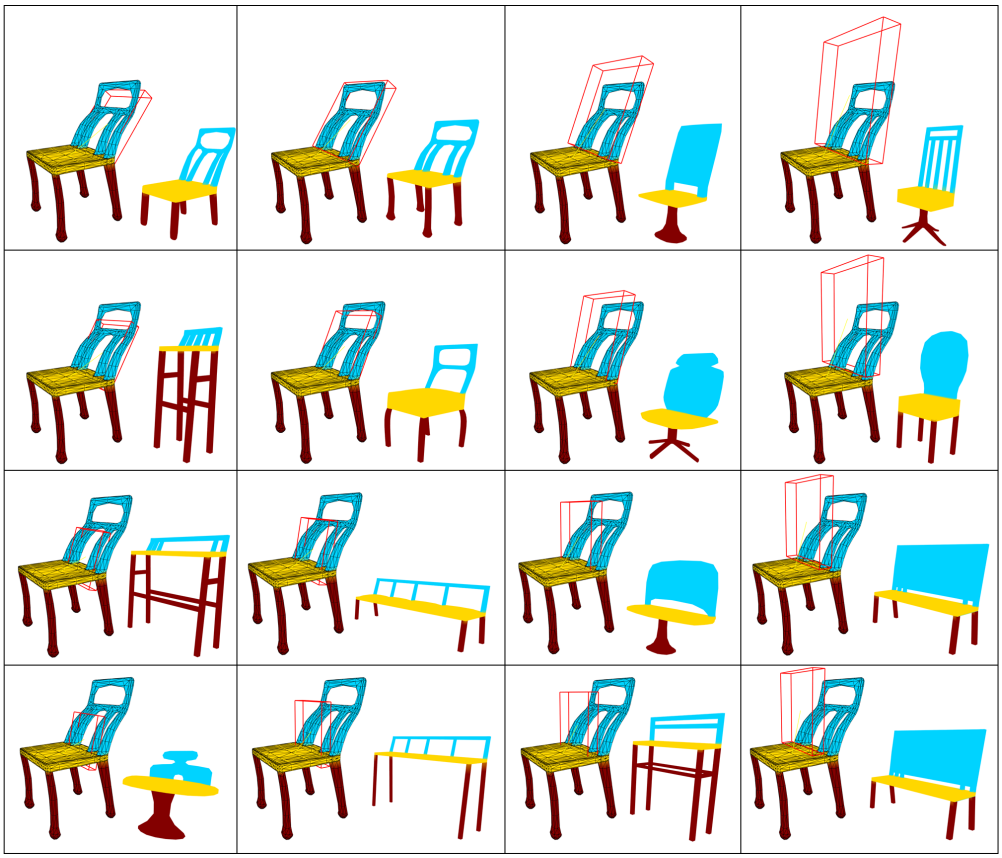

Figure 6.1: The analysis of the relation between the backrests and seats of the chair dataset results in the first two principal components being used as exploration parameters. In each box, the original shape used for the exploration is shown on the left, while the preview of the shape that best fits the altered exploration parameters is shown on the right. The altered parameters are represented visually by the bounding box of the backrest.

while large backrests can be found near the top and right sides of the parameter space. Furthermore, most benches are closer to the bottom side of the parameter space since the horizontal distance between their backrest and seat is small in relation to their combined bounding box diameter.

The reason for why we exclude some shapes for the test is because it is a good example for how the parameterization can change depending on the dataset. The excluded part of the set consists of many shapes that are scaled in direction of the y-axis. For the backrests and seats in particular this means that stretching the shape causes the vertical distance and relative scale of the backrest to increase, while the horizontal distance (relative to the combined bounding box diameter) becomes smaller. Vertically shrinking the shape on the other hand decreases the vertical distance and relative scale while at the

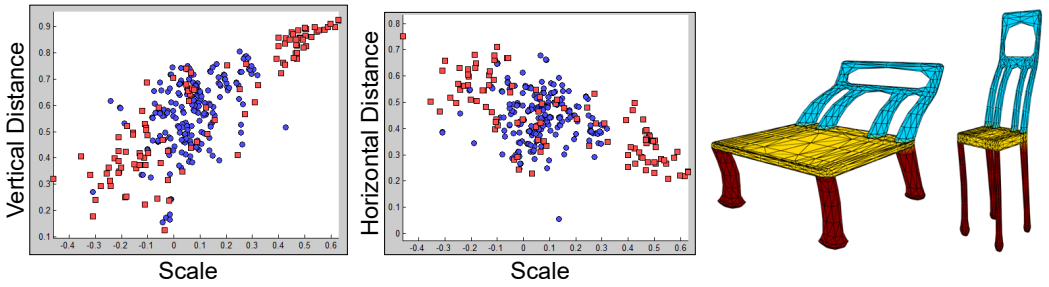

Figure 6.2: A plot of the relative scale and vertical and horizontal distance between the backrests and seats of the chair dataset. Blue circles represent the shapes of the test set, while red squares represent the excluded shapes. For the excluded shapes, there is a strong correlation between the shown features. This is because most of the excluded shapes have been scaled vertically as can be seen on the right.

same time increasing the horizontal distance in relation to the combined bounding box diameter. As a result, the correlation between the scale, horizontal and vertical distance becomes stronger, which can also be seen in Table B.1.

Figure 6.2 shows a plot of the datasets and two examples for the shapes of the excluded set. The relative scale is shown on the x-axis and the distances are shown on the y-axis, with test shapes represented by blue circles and the excluded shapes represented by red squares. As can be seen, the excluded shapes show a strong correlation between the scale and the vertical and horizontal distance of the backrests – as the scale increases, the vertical distance also increases while the horizontal distance becomes smaller. The consequence is that when using the whole dataset in the analysis, the first principal component has a larger variance than when only using the shapes in the test dataset and thus only the first principal component is used for exploration.

We also analyze the relation between the legs and seat of the chairs. The results are similar to the observations made previously in Section 4.3. The legs of the shapes mostly vary in scale and horizontal distance, and these two parameters are strongly correlated. Because of that, chairs with a single large leg are clustered on one side of the parameter space, while benches can be found on the other side because their legs are further apart horizontally and smaller in relation to the size of the seat. An example can be seen in Figure 4.6.

Another test is performed using the vases dataset. Again we use the provided segmentation files for the labeling of the parts, although we correct two of the cup shapes that have their insides incorrectly labeled as a neck part instead of a body part. Three relations are inspected in the following tests: neck to body, base to body and handle to body.

For both the neck and the base of the shapes, there is almost no variability in horizontal distance and angle between parts. There is also no strong correlation between the two remaining features vertical distance and scale and thus the analysis yields two principal components for the exploration. Since there are only two relevant features

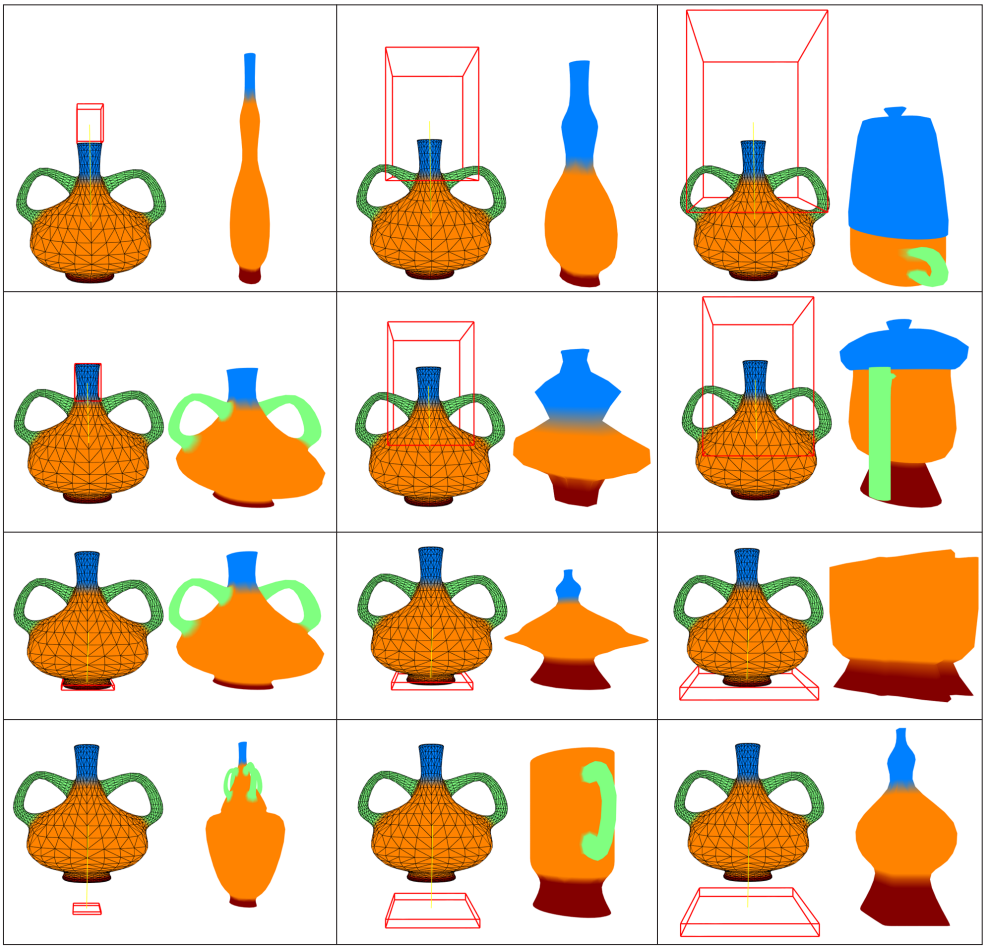

Figure 6.3: Exploration of the vase dataset using the relation between neck and body (top two rows), and base and body (bottom two rows). In both cases, only two features are relevant (horizontal distance and scale) and they are not strongly correlated. Thus the horizontal distance and scale are used directly as parameters for the exploration.

anyway, the two exploration parameters are the individual features, with the scale being the feature with the larger variance in both cases. Figure 6.3 shows some example shapes found by exploring the collection using the relations of the neck and base parts to the body of the shapes.

Exploration using the relations of the handles to the body of the vases also yields interesting results. Both the first and second principal components have high variance and thus both are used in the exploration. Increasing the first exploration parameter results in an increased vertical distance while the horizontal distance decreases. The angle also decreases slightly, but the scale remains almost unchanged. Increasing the second parameter mostly results in the scale of the handle becoming larger and the angle

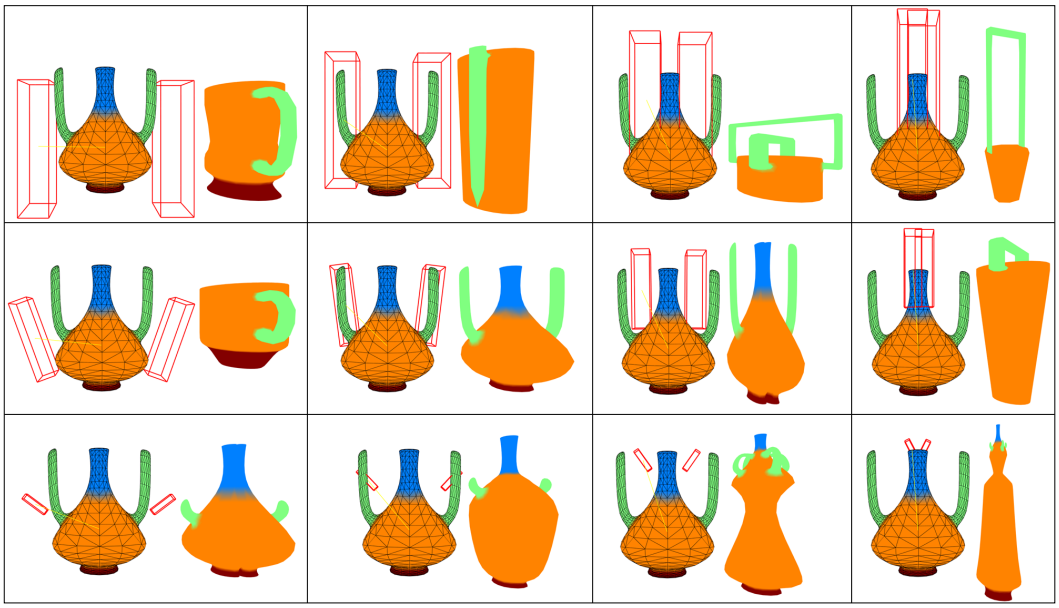

Figure 6.4: Exploration of the vase dataset using the relation between handle and body. Changes to the first principal component are shown on the x-axis, while changes to the second principal component are shown on the y-axis.

becoming smaller, though the vertical distance also decreases slightly.

Once again this results in similar shapes being clustered. Handles that are orthogonal to the body can be found on the top of the parameter space, while angled handles can be found near the bottom. Furthermore, on the left side of the parameter space one can find handles that are connected to the side of the body, while the handles on the right side are mostly connected to the top of the body. Since the handle of cup shapes are orthogonal and on the side of the body, these types of shapes are clustered near the top left of the parameter space. Bucket shapes on the other hand can be found near the top right of the parameter space since the handles are also orthogonal to the body, but placed on top of it.

Regular vases can be found near the bottom since their handles are generally smaller compared to the body. Again the shapes on the left side have handles that are attached to the side of the body while the right side has shapes with handles that are connected to the top of the body. As a result, vertically long shapes can be found mostly on the right side, while short shapes can be found on the opposite side.

Finally, we also test two small datasets. The candelabra set from the COSEG benchmark consists of 28 shapes whose parts can be categorized as candle, flame, base and handle. The plane dataset consists of 15 plane and fighter jet shapes that have been put together from shapes found on 3D Warehouse and the Princeton Shape Benchmark. Its parts consist of fuselage (or body), wheel, wing, engine and stabilizer. Note that we assign engines of passenger planes and missiles of fighter jets the same label.

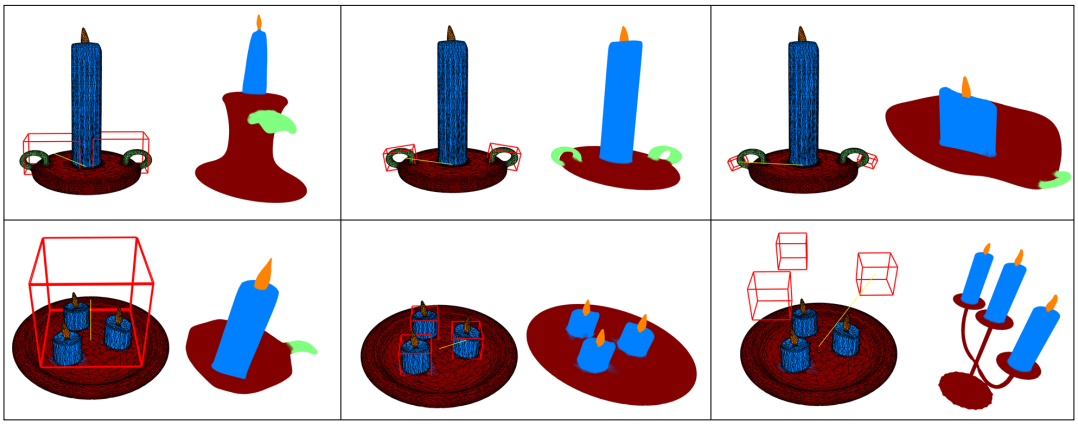

Figure 6.5: Exploring the candelabra dataset using different relations. In the first row, the relation between the handle and the base of the candelabra is changed. In the second row, the dataset is explored by changing the relation between the candles and the base.

First we look at the relation of the handles to the base of the candelabra dataset. The first two principal components are used since they both have a high variance. The first exploration parameter mostly controls the scale and horizontal distance – as the scale decreases, the horizontal distance increases. This is because the handles are mostly attached to the side of the base and thus its relative size is strongly correlated with the horizontal distance between the part centers. The second parameter mostly controls the other two features, with a decrease in the vertical distance causing the angle between the parts to increase. The reason for this is that most handles that are orthogonal to the base are attached closer to the top, while most handles that are angled are attached closer to the bottom of the base. An example of this can be seen in the top row of Figure 6.5.

For the candle-base relation of the candelabra dataset, the angle accounts for almost no variation within the dataset and is thus ignored in the PCA. The first principal component shows a strong correlation between scale and distance. As the scale increases, the vertical distance also does, but the horizontal distance decreases. This is because dataset contains shapes with up to three candles connected to the base. For shapes with a single candle it is usually placed in the center of the base. On the other hand, the candles are more spread out if there is more than one and the base needs to be larger to accommodate the additional candles in those cases. Increasing the second parameter results in an increase in vertical distance and a slight increase in horizontal distance. In this relation it is less useful for finding different candles, but it can be used to find differently shaped bases like plates, bowls or stands. Examples can be seen in the bottom row of Figure 6.5.

For the plane dataset we test three different relations. The wings of the planes in relation to the fuselage mostly differ in angle and scale. Vertical and horizontal distance are discarded for the PCA. Since both principal components show a high enough

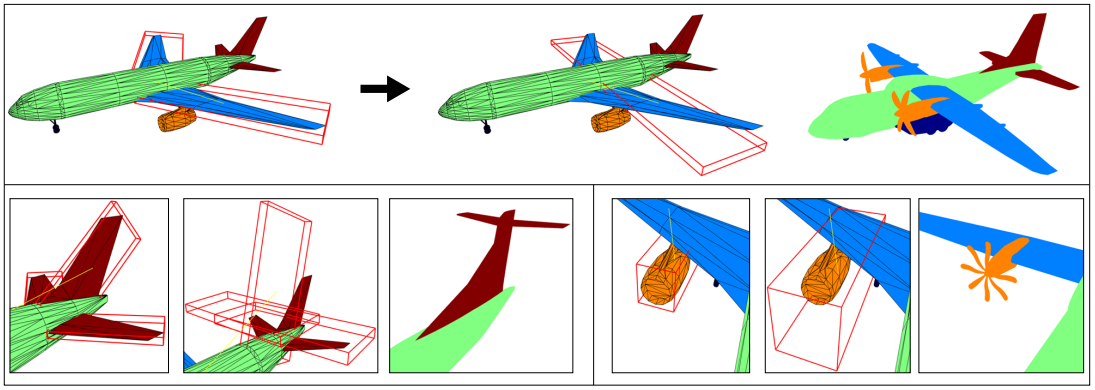

Figure 6.6: Exploring the plane dataset using different relations. The top row shows how changing the relation between the wing and fuselage can be used to find other shapes in the collection. In the bottom row, the relation between stabilizer and fuselage (left), as well as the relation between engine and wing (right) are used in the exploration.

variance, both remaining features are used as parameters directly. The scale is useful for differentiating passenger planes from fighter jets, while the angle is useful in finding differently shaped wings. The stabilizer-to-fuselage relation mostly varies in angle, though the other three features still show enough variance to be kept. The variation can be explained well by the first principal component – increasing the angle results in an increase in horizontal distance, but a decrease in vertical distance and scale. Finally, the engine-to-wing relation mostly differs in scale and horizontal distance, with angle and vertical distance having almost no variance. The first principal component is enough to describe the variability of the relation as scale and horizontal distance are positively correlated. This is because the engines and rockets have the same label – most of the fighter jets have small wings with rockets attached to the end of the wings, while the passenger plane have large wings and the engines are closer to the center of the wings. Figure 6.6 show examples of these relations.

6.1.2 Multi-relation Exploration

In this section we explore the shape collections by taking multiple relations into account. By changing the parameters of multiple relations it is possible to find more specific results. The tests are performed using the same datasets as before.

For the first test, we use the chair dataset, again with some of the shapes excluded. The testing procedure is as follows: We first set the parameter for the leg-seat relation of the chair. Then we alter the parameters for the relation between the backrest and the seat. The results can be seen in Figure 6.7. The left column shows the parameters for the legs of the chair. We choose three settings for the test with parameter values near the ends and middle of the parameter space. This corresponds to the different kind of chair shapes in the dataset – benches whose legs are far apart, chairs with a single large

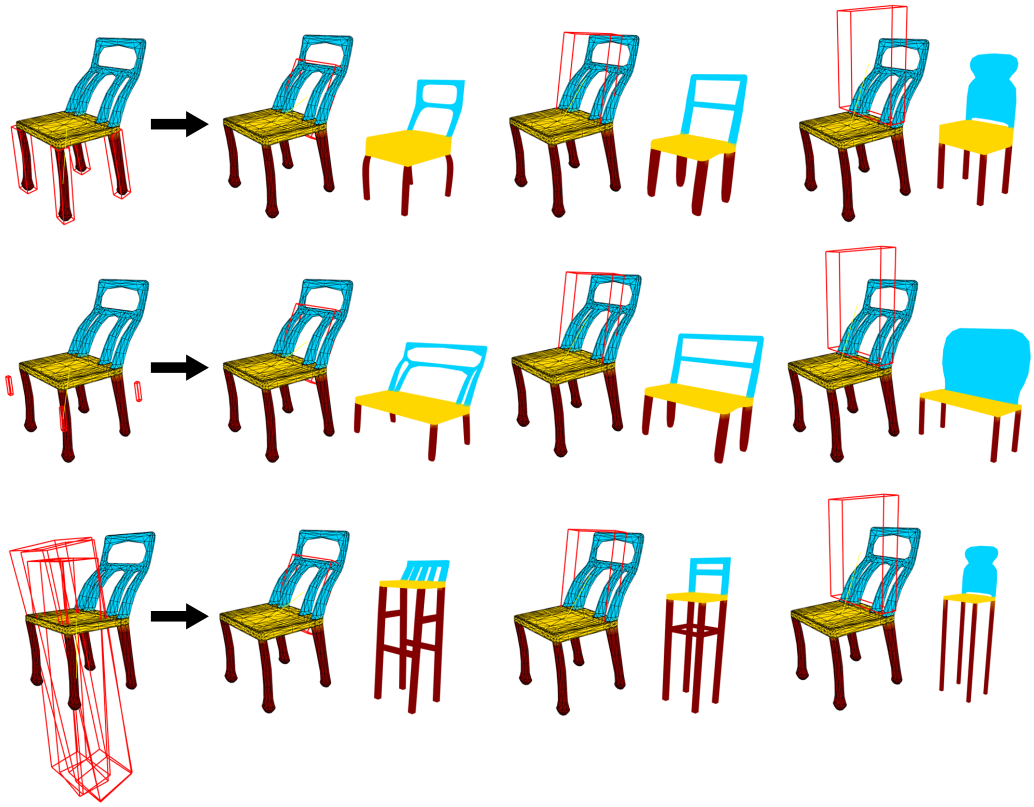

Figure 6.7: Exploration of the chair dataset using multiple relations. The column on the left shows the parameters of the leg-seat relation that is used for the other shapes of the corresponding row. Changing the parameters of the backrest-seat relation then yields different results by also taking into account the setting of the leg-seat relation.

leg and conventional chairs.

The rows of Figure 6.7 then show the exploration of the dataset using the relation between the backrest and seat of the chairs, taking into account the altered parameter for the leg-seat relation shown in the left column. For every other column, the three shapes in the column use the same parameters for the backrest-seat relation, which can be seen in the visualization of the bounding box. However, the shape that best fits the altered parameters is not the same as the parameters for the legs are also taken into consideration. As a result, the shapes in the first row are conventional chairs, the shapes in the second row are more like benches and the shapes in the third row all have a single large leg.

For the next test we also consider the number of parts as a parameter. Finding a good weight ω to determine how much influence the difference in part numbers has depends on the use case. For this test we intend to only search for shapes that have the exact same number of parts. For that purpose, ω is set to a large value, in this case 100.

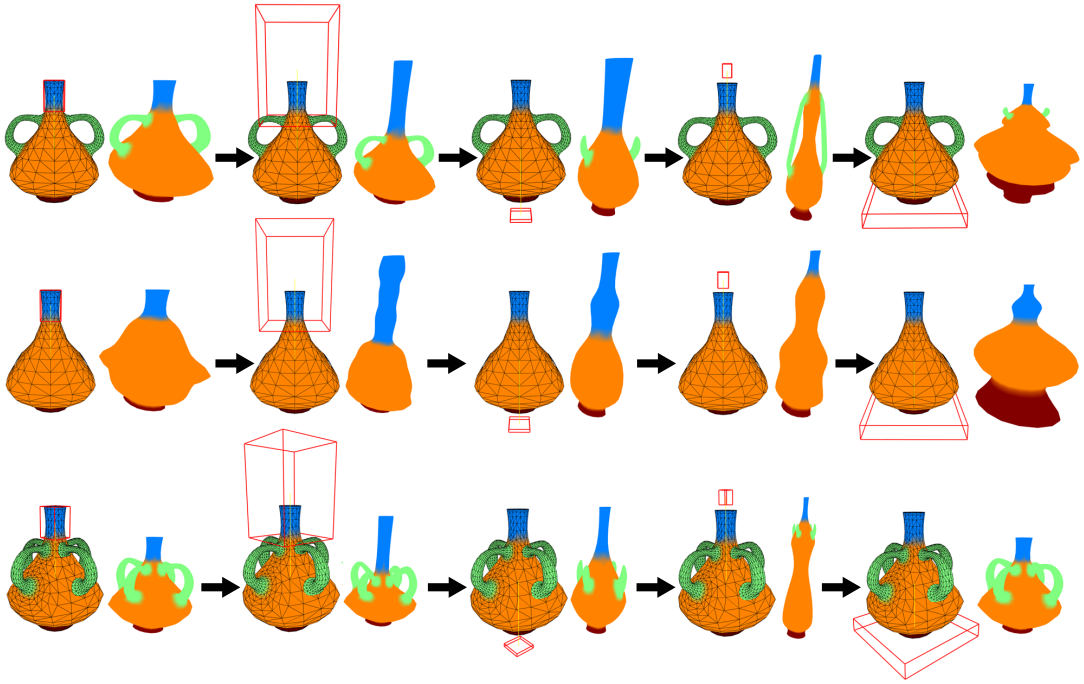

Figure 6.8: Exploration of the vases dataset using multiple relations and different numbers of parts. The original shape is shape shown in the top left, with the number of handles altered for the starting shape of the second and third row. Exploring the collection using the same changes to the parameters results in similar shapes for each starting shape, with the major difference being the number of handles.

We choose a shape from the vase dataset as the starting shape to search for other shapes in the dataset. This starting shape contains a neck, a base and two handles which are all connected to the body of the shape. Then we explore the collection by alternately changing the parameters of the neck and the base of the shape. The exploration consists of the following four steps:

1. The vertical distance and scale of the neck are increased.
2. The vertical distance of the base is increased, while the scale of the base is decreased.
3. The scale of the neck is decreased.
4. The vertical distance of the base is decreased, while the scale of the base is increased.

We perform this exploration process three times, each time with a different number of handles. In the first test, the number of handles is unchanged. In the second test, both handles are removed. Then in the third test each handle is copied and placed so that the four handles form a rotational symmetry. This specific placement isn't strictly

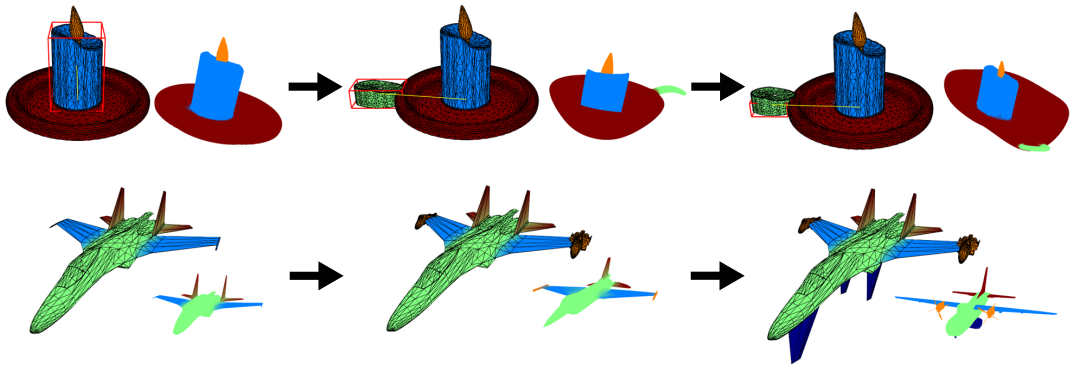

Figure 6.9: Exploration of the candelabra and plane datasets by adding parts that do not exist in the starting shapes shown on the left. By copying a part and assigning it a different label, it is possible to find shapes of the collection with different parts than the starting shape. In this example, a handle is added to the candelabra in the top row, and engines and wheels are added to the plane in the bottom row.

necessary – it would also be possible to place the copied handles at the exact location of their original counterpart since we only measure the horizontal distance as a whole and not separately for the x- and z-directions. However, the resulting visualization is closer to the types of shapes that are contained in the dataset, making the visualization more accurate. The exploration parameters for the handles are not changed in any of the steps.

The results can be seen in Figure 6.8. The first row shows the exploration with two handles, the second row without handles and the third row with four handles. The shapes shown in each column show a lot of similarities, reflecting the change in parameters. The major difference is the number of handles. The first column shows shapes with a body that is roughly as wide as it is long, with both the base and neck being small in comparison. The body and base of the shapes in the second column are similar, but the neck is longer for all three shapes. the change of the base results in the body of the shapes becoming a little longer and less wide, as shown in the third column. The fourth column has shapes with a very long and thin body, resulting from the neck decreasing in size.

Finally, the fifth column is the only one with a shape that is very different from the others. The first two rows show shapes with a large base and body, but a small neck, while the third row contains the same shape as in the first column. As can be seen, its base is significantly smaller than the bases of the other two shapes. This is because there is no shape with four handles in the dataset that fits the altered parameters for all the parts.

In the last exploration test, we show how to deal with cases where the starting shape does not contain all possible relations. For this we use the candelabra and plane datasets. The top row of Figure 6.9 shows a candelabra shape that consists of a candle, flame and

base. We would like to find a shape with a handle, so we copy the candle part, assign it the label of a handle and place it to the side of the base. Immediately the best fitting shape in the collection changes and now shows a candelabra that also has a handle, as shown in the second column. The newly placed handle can then be used to explore the collection as usual, which can be seen in the third column.

Lastly, a similar test is performed with the plane dataset, which is demonstrated in the second row of Figure 6.9. The starting shape, shown on the left side, is a fighter jet that contains the fuselage, wings and stabilizers. We first make two copies of the fuselage, assign the labels of an engine and place them at the end of both wings, resulting in the shape shown in the second column. Then we copy the stabilizers, turn them into wheels by assigning the corresponding label and then place them under the fuselage. The best fit changes again, now to a plane that has both wheels and engines, as shown in the third column.

6.2 Synthesis Results

This section shows the results of creating new shapes in the shape synthesis stage. We use the same datasets as in the exploration tests and create some new shapes by combining parts of different existing shapes. This is done by first picking a starting shape and using the parts of the shape to explore the shape collection. Once a shape with the desired parts has been found, the corresponding parts of the current shape are replaced by them.

Figure 6.10 shows some new chairs created by combining parts of the chair dataset. For most of shapes, the existing parts are merely replaced by other parts. The second shape in the top row and the fourth shape in the bottom row originally do not contain a backrest, so in both cases we add one by copying the leg, placing it on top of the seat, changing the label to that of a backrest and replacing it with a backrest from another shape. Similarly, the starting shape of the fifth column in the bottom row only has a single leg, but we create a chair with three legs by copying it twice.

Shape synthesis results for the vase dataset can be seen in Figure 6.11. Once again we choose a starting shape, explore the collection using the part relations and exchange parts between shapes. For each shown shape we exchange all parts except the body of the vase. Note that the second and fourth shapes in the bottom row originally do not have a neck and base part respectively. For these parts we again copy a different part and changed the label to find corresponding parts in other shapes.

Figure 6.12 shows a few examples using the candelabra dataset. The two shapes on the left side are created in a continuation of the exploration process shown in Figure 6.9. Using the created parts to explore the collection, we then exchange these parts with corresponding parts from the shapes that best fit the altered exploration parameters. In this case, two different handles are swapped in to create a new shape.

Finally, Figure 6.13 shows three synthesized plane shapes. Similarly to the previous example, the two shapes on the bottom row display a continuation of the exploration from Figure 6.9. To create these shapes, two different kinds of engine parts, another stabilizer and some wheels are swapped in.

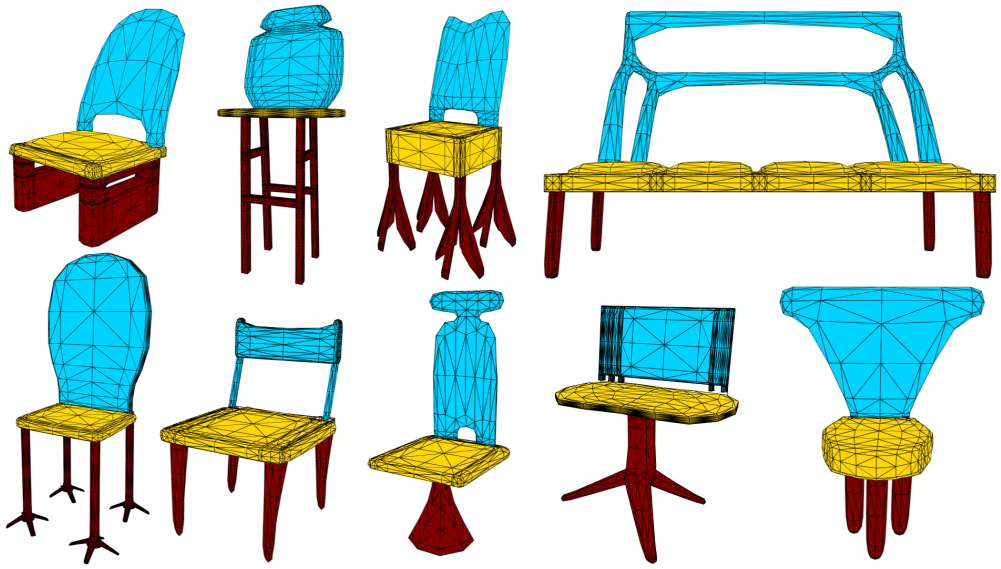

Figure 6.10: Some examples for shapes created in the shape synthesis stage, in this case using the chair dataset.

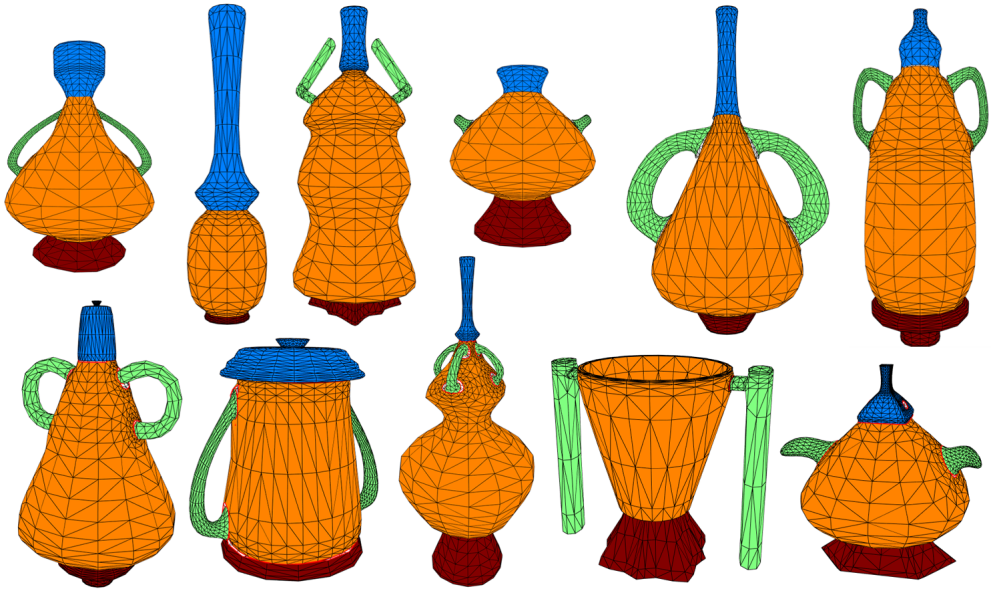

Figure 6.11: Some examples for shapes created in the shape synthesis stage, in this case using the vase dataset.

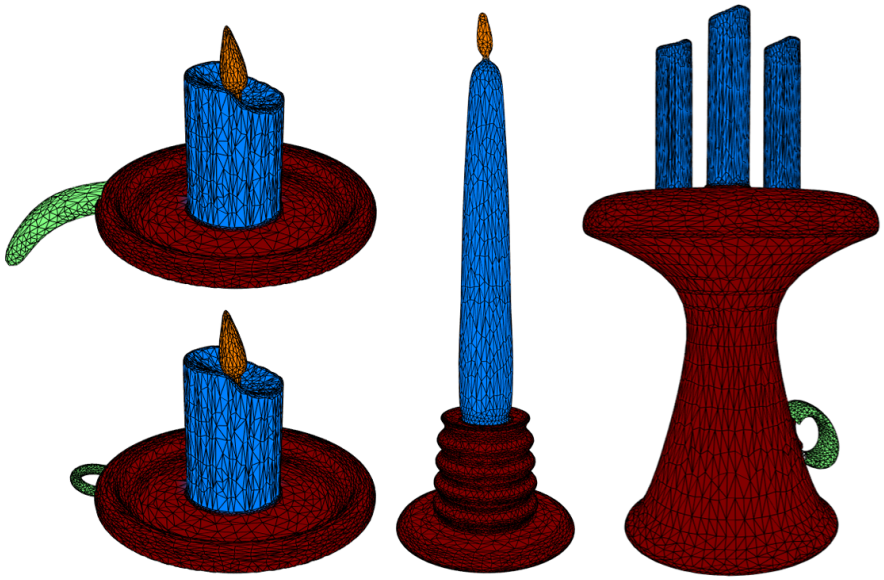

Figure 6.12: Some examples for shapes created in the shape synthesis stage, in this case using the candelabra dataset.

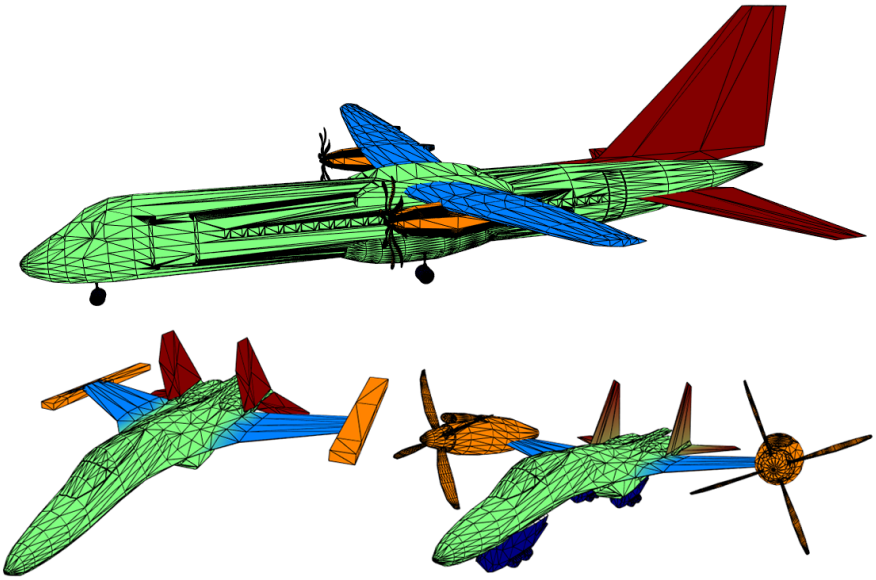

Figure 6.13: Some examples for shapes created in the shape synthesis stage, in this case using the plane dataset.

6.3 Limitations

In this section, we take a look at some of the drawbacks of our proposed system. One problem is that the center of the parameter space is usually more densely populated than the outer regions. Since altering parameter values directly is done using a slider, this can make it difficult to find shapes that only differ slightly in their parameters. At the same time, a small change to a parameter with a value near the borders of the parameter space might not give a different result for the best fitting shape when the distance between the nearest neighbors is large. A possible solution could be to let the user choose from the k best fitting shapes instead of just the single shape that best fits the altered parameters. Alternatively, it would be possible to set the parameter controls in a way such that the changes to the parameter are smaller when there exist many shapes near its current value. Furthermore, incorporating a depiction of the density into the parameter controls similar to the methods described in Section 3.2 could give the user a better understanding of the parameter space prior to the actual exploration.

Another possible issue is that our visual representation of parameter changes is just an approximation, so the results are not always exact. Since we do not assume the shapes to be globally aligned, distance can only be displayed by translating the bounding boxes based on their current displacement. Similarly, a change of angle can only be visualized based on the current angle between the parts – if the angle is zero, it is not possible to determine the correct axis of rotation. The relative scale of parts is also only determined by their bounding box diameter, thus there is no clear distinction between thin long shapes and large boxy shapes. Some of these problems can be alleviated by assuming that the shapes are globally aligned, but even then it is possible that there may be no perfect alignment between a pair of shapes.

In the shape synthesis stage, parts are aligned based on the contact area between the parts. While this results in the parts fitting together more naturally, the transformation for the alignment is applied globally to the whole part. This can cause the swapped-in parts to become much larger or smaller compared to the rest of the shape. An example of this can be seen in the fourth shape in the bottom row of Figure 6.11. The initial shape does not contain a base part and because of the alignment process the swapped-in handles are larger than the main body of the shape. As a consequence, the body is not touching the ground, so we need to add a base part to make the shape more realistic.

CHAPTER 7

Conclusion

In this final chapter we first provide a summary of our proposed framework, condensing the most important information into Section 7.1. Then we discuss the results obtained from our tests, highlighting the goals we achieved as well as some problems with our approach. This can be found in Section 7.2. Finally, we take a look at how some of these problems could possibly be alleviated, suggesting topics for further research topics in Section 7.3.

7.1 Summary

In this thesis, we presented a framework for shape synthesis from shape collections, introducing a new method of parameterization and exploration of those collections. With ever growing online repositories for 3D models becoming more prevalent, creating new shapes by combining parts of already existing shapes has become a real possibility for content creation that is not only easy to use, but also faster than traditional modeling methods.

Our proposed framework consists of four stages: co-analysis, parameterization, exploration and synthesis. In the co-analysis stage, shapes are segmented into parts which are then grouped into categories according to their function. Instead of focusing on a specific co-analysis algorithm, we use a modular and extensible approach that allows different algorithms to be developed and used in this stage.

The parameterization and exploration stages form the other main contribution of this thesis. Relations between parts are analyzed in how their spatial arrangements vary across the collection, yielding a small number of parameters that can be used to explore the collection. The parameters can be altered either directly or by interacting with the shape itself. A visual representation of how the spatial arrangements change when altering the parameters provides a better understanding of the kind of shapes that can be found in the exploration process, making it more intuitive to search for specific

shapes. To account for shapes containing a varying number of parts, the system includes a copy-and-paste operation for parts that also allows for the addition of parts that are not present in the currently displayed shape.

In the last stage, new shapes are assembled by combining parts from the shapes found in the exploration process. Since the shapes in the collection are not necessarily aligned, the new parts are transformed to fit together with the remainder of the shape. Once swapped in, the new parts can then be used to explore the collection as usual, in order to replace further parts or even parts that have already been exchanged before. This makes it possible to create a large variety of new shapes in a short time.

7.2 Discussion

Our test results for the parameterization and exploration stages show that the shapes found while browsing the collection are mostly in line with our expectations. Spatial features with low variation within the collection – which are not very useful for distinguishing between shapes – are eliminated successfully, as is the case with the neck and base parts of the vases dataset where there is almost no variation in angle and horizontal distance. Similarly, our analysis correctly detects correlations between spatial features as evidenced by the different results for the two chair datasets. As a result, the parameters for each relation reflect the way in which the relation varies the most across the collection.

The visual representation of parameter changes proves a useful aid for understanding the effect of altering the parameters, making it more intuitive to search for specific kinds of shapes when interacting with the parameters directly. Representing the selected parts by their bounding boxes serves as a useful approximation and gives a rough preview of the shape that best fits the altered parameters.

Exploring the shape collection by focusing on a single relation is a quick way to browse through the available shapes containing the parts of the chosen relation. On the other hand, it is possible to search for shapes with more specific part arrangements by taking all relations into account. The ability to alter the parameters by interacting with the shape itself is also useful in cases where the user has a concrete idea of the desired shape. The disadvantage of this multi-relation exploration is that for shapes that have many different relations it can be tedious to set the parameters for all relations. Letting the user choose a subset of relations instead of taking all relations into account could be an improvement in these cases.

Our system also gives special consideration to cases where the number of parts differs between shapes, which is something that is usually treated with less importance in existing approaches. Finding a good weight for how much differing part numbers influence the computation of the best fit can be difficult and depends on the use case. A low number like 0.2 can help distinguish similar shapes that mostly differ by the number of parts, while a value like 0.5 is more likely to yield shapes with the desired part number unless there are no similar shapes with the same number of parts. High weights with values larger than 1 can be used to restrict the resulting shapes to contain a specific number of parts, but this can greatly reduce the number of possible fits if there is a large variation

in part numbers within the collection, making the exploration parameters themselves less relevant.

Altering part numbers for a given shape is made simple by a copy-and-paste system for parts. If one wants to only increase the part numbers without changing the associated parameters for the purpose of exploration, the copied parts can be quickly placed at the exact location of its original part. On the other hand, the copied parts can also be transformed and placed at a different location, for example to later exchange them for other parts. Changing the label of the copied part allows the user to add parts that are not present in the current shape, making it possible to explore the entire collection from every initial shape.

Our framework allows us to work with a variety of shape collections from different sources. The modular design of the co-analysis stage makes it possible to deal with different problems that may come up when working with different shapes. For example, the co-analysis algorithms we implemented do not function well when the shapes consist of multiple disconnected surfaces or contain geometrical defects. In these cases we were able to create a manual segmentation when there was no segmentation provided with the shapes, as is the case with the plane dataset. Of course, the manual segmentation tools are also useful for improving the results of other co-analysis methods which can then be exported for later use.

The results of our shape synthesis tests show a nice variety of created shapes. With our alignment method we try to optimize the alignment between the contact areas of the parts, which yields a better transition between adjacent parts. While there is certainly much more work required to produce high-quality shapes in this manner, we believe our shape synthesis stage is sufficient in demonstrating the capabilities of the framework as a whole.

7.3 Future Work

There are still a lot of open questions when it comes to the problem of parameterization and exploration of shape collections. We specifically look at relations between pairs of parts, but naturally it is also possible to consider the individual properties of parts or their arrangements with regards to the whole shape. These ideas have also been explored individually in existing approaches, but to our knowledge there is no method utilizing all of these viewpoints simultaneously. Also, while our system detects correlations between spatial features for a single relation, it would be interesting to extend this approach to finding possible correlations between different relations. Furthermore, to produce better results when working with a large number of features, it might be necessary to consider non-linear methods of dimensionality reduction [LV07] for the computation of the exploration parameters rather than the linear method of PCA.

Incorporating varying part numbers is also something that still needs more research. Existing approaches tend to ignore the number of parts present in shapes or simply use the average when performing computations. However, part numbers can be an important aspect when distinguishing between shapes and thus should not be ignored. Searching for

correlations between part numbers and spatial arrangements between parts (an example of which can be seen in the chair dataset, as described previously) is an interesting idea, but since part numbers are given by a discrete value while other features are continuous, the best way of doing so is yet to be determined.

The large quality differences for shapes from online repositories are a general problem with these methods. Many existing approaches make assumptions for the state of the shapes, such as the shapes being globally aligned or there being intersections between the triangles of adjacent parts. Some algorithms might even require manual cleanup of the 3D models before they can be used. The result is that the actual practical use of these methods is greatly diminished. Considering all possible problems regarding the quality of the shapes is one of the major challenges when working with shapes taken from online repositories.

APPENDIX A

Datasets

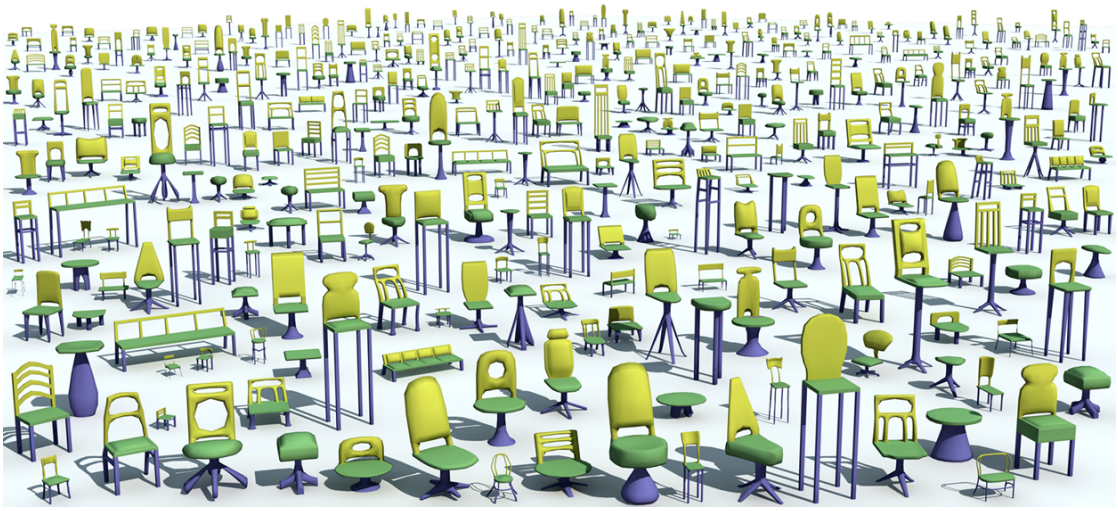

Figure A.1: The chair dataset of the Shape COSEG Dataset. Image adapted from [Wan].

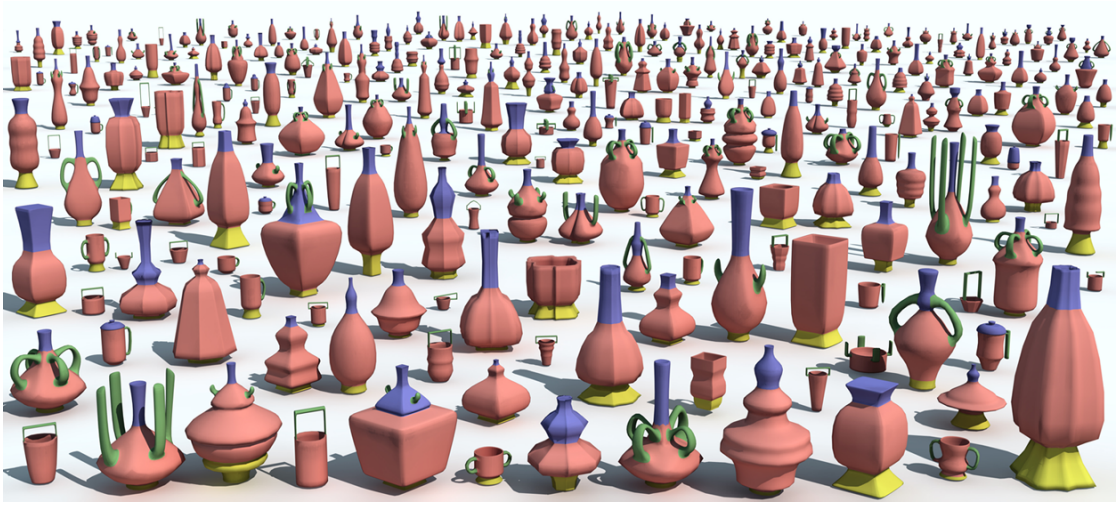

Figure A.2: The vase dataset of the Shape COSEG Dataset. Image adapted from [Wan].

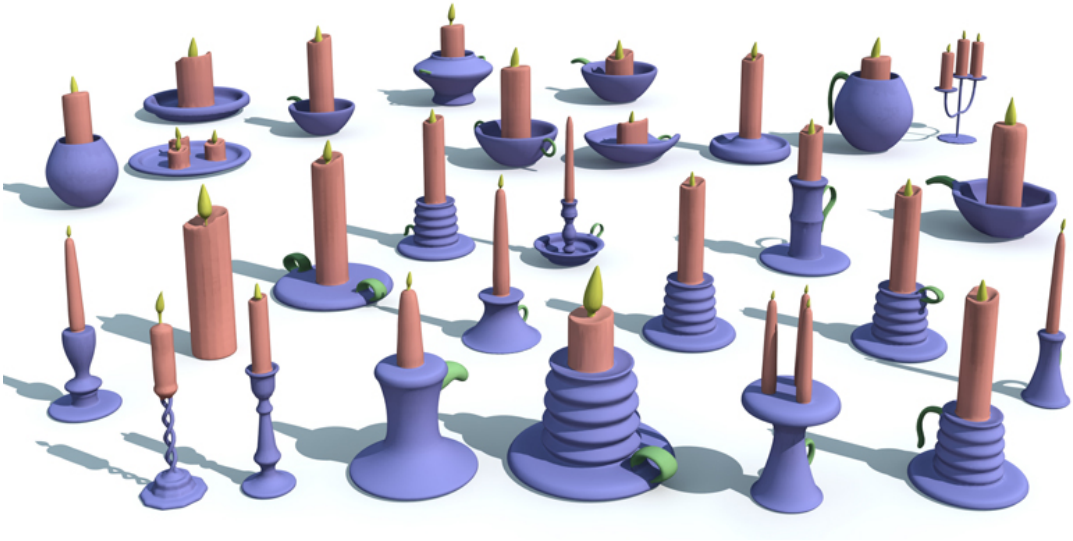

Figure A.3: The candelabra dataset of the Shape COSEG Dataset. Image adapted from [Wan].

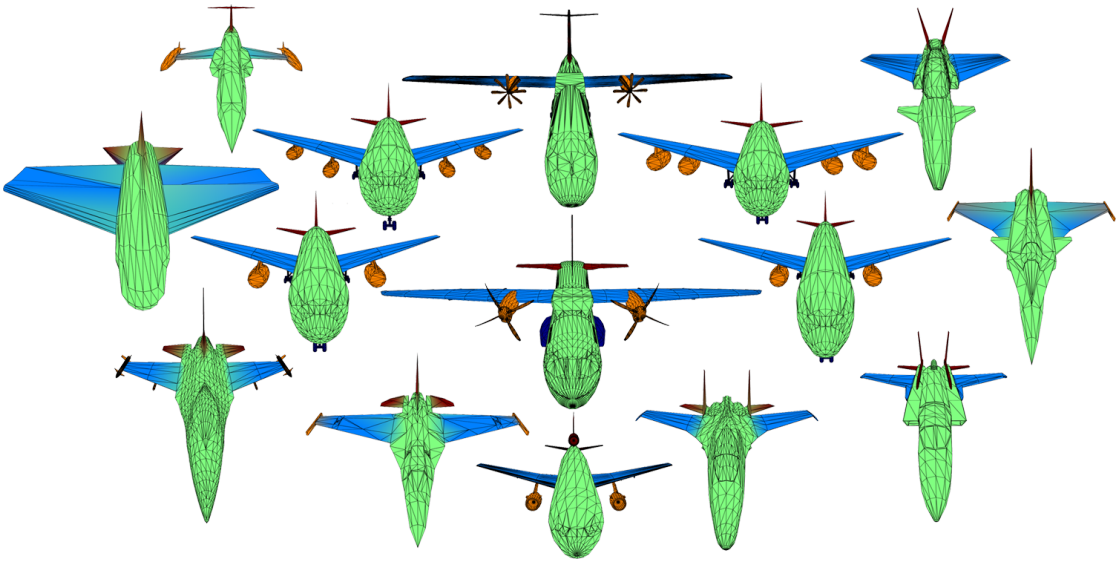

Figure A.4: The plane dataset with shapes taken from 3D Warehouse [Tri] and the Princeton Shape Benchmark [SMKF04].

APPENDIX B

Parameterization Data

Covariance Matrix of Chair Dataset				
Test Set	V.Dist.	H.Dist.	Angle	Scale
V.Dist	0.0172	0.0010	-0.0046	0.0068
H.Dist	0.0010	0.0086	0.0039	-0.0014
Angle	-0.0046	0.0039	0.0102	0.0013
Scale	0.0068	-0.0014	0.0013	0.0130
Full Set	V.Dist.	H.Dist.	Angle	Scale
V.Dist	0.0290	-0.0067	-0.0069	0.0261
H.Dist	-0.0067	0.0120	0.0057	-0.0119
Angle	-0.0069	0.0057	0.0133	-0.0026
Scale	0.0261	-0.0119	-0.0026	0.0410

Table B.1: The covariance matrices of the two chair datasets. The test set contains 275 shapes, while the full set contains all 400 shapes. In the full set, the scale is correlated more strongly with the vertical and horizontal distance.

Dataset	**Relation**	**Variance**			
		V.Dist.	H.Dist.	Angle	Scale
Chair Test	Back-Seat	0.0172	0.0086	0.0102	0.0130
	Leg-Seat	0.0287	0.1013	0.0012	0.1479
Chair Full	Back-Seat	0.0290	0.0120	0.0133	0.0410
	Leg-Seat	0.0362	0.0901	0.0018	0.1582
Vases	Neck-Body	0.0109	0.0000	0.0011	0.0516
	Base-Body	0.0179	0.0001	0.0011	0.0281
	Handle-Body	0.0535	0.0354	0.0679	0.0571
Candelabra	Candle-Base	0.0763	0.0182	0.0000	0.0570
	Handle-Base	0.0096	0.0155	0.0140	0.0076
Plane	Wings-Body	0.0009	0.0023	0.0277	0.0140
	Stabilizer-Body	0.0064	0.0082	0.0448	0.0110
	Engine-Wing	0.0018	0.0388	0.0036	0.0895

Table B.2: The variances of each feature for the relations of the test datasets. Grey text means that the variance of the feature is too low and is thus discarded.

Dataset	**Relation**	**PC Variance**	**PC Coefficients**			
			V.Dist.	H.Dist.	Angle	Scale
Chair Test	Back-Seat	1.5988	0.0784	-0.0365	-0.0546	0.0501
		1.3071	0.0526	0.0504	0.0497	0.0624
	Leg-Seat	2.1831	-0.0920	0.1684	-	0.1224
		0.6945	0.1168	0.2307	-	-0.0059
Chair Full	Back-Seat	2.3185	0.0941	-0.0552	-0.0424	0.1121
		0.9632	-0.0067	0.0120	0.0057	-0.0119
	Leg-Seat	2.2294	-0.1052	0.1564	-	0.2584
		0.6696	0.1253	0.2254	-	-0.0163
Vases	Neck-Body	1.1176	0.0737	-	-	0.1606
		0.8824	-0.0737	-	-	0.1606
	Base-Body	1.2552	0.0945	-	-	0.1186
		0.7448	-0.0945	-	-	0.1186
	Handle-Body	1.8515	0.1566	-0.1320	-0.0578	-0.0067
		1.5020	0.0464	0.0019	0.1744	-0.1710
Candelabra	Candle-Base	1.7364	0.1230	-0.0803	-	0.1598
		0.8688	0.2336	0.0708	-	-0.0231
	Handle-Base	1.6291	0.0374	0.0711	0.0065	-0.0630
		1.4880	-0.0632	0.0325	0.0843	-0.0071
Plane	Wing-Body	1.2957	-	-	0.1177	0.0837
		0.7043	-	-	0.1177	-0.0837
	Stabilizer-Body	2.5328	-0.0399	0.0488	0.0939	-0.0540
		0.7089	0.0056	-0.0050	0.1719	-0.0605
	Engine-Wing	1.3380	-	-0.1393	-	-0.2116
		0.6620	-	0.1393	-	0.2116

Table B.3: Principal Component variances and coefficients for each relation of the test datasets. Missing entries mean that the feature has a low variance and was discarded. Grey text means that the second principal component has low variance and isn't used in the exploration.

Bibliography

[AKZM14] Melinos Averkiou, Vladimir G. Kim, Youyi Zheng, and Niloy J. Mitra. ShapeSynth: Parameterizing model collections for coupled shape exploration and synthesis. *Computer Graphics Forum*, 33(2):125–134, 2014.

[BBV+01] Yuri Boykov, Yuri Boykov, Olga Veksler, Olga Veksler, Ramin Zabih, and Ramin Zabih. Fast Approximate Energy Minimization via Graph Cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.

[BCG08] Mirela Ben-Chen and Craig Gotsman. Characterizing Shape Using Conformal Factors. *Proceedings of the 1st Eurographics Conference on 3D Object Retrieval*, pages 1–8, 2008.

[Bha43] Anil K. Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society*, 35:239–258, 1943.

[BM92] Paul Besl and Neil McKay. A Method for Registration of 3-D Shapes, 1992.

[BMP02] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape Matching and Object Recognition Using Shape Contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(24):509–522, 2002.

[Chu97] Fan R. K. Chung. *Spectral Graph Theory*. 1997.

[CKGK11] Siddhartha Chaudhuri, Evangelos Kalogerakis, Leonidas Guibas, and Vladlen Koltun. Probabilistic reasoning for assembly-based 3D modeling. *ACM Transactions on Graphics*, 30(4):35:1–35:10, 2011.

[CM02] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[CYS04] Timothee Cour, Stella Yu, and Jianbo Shi. Normalized Cut Segmentation Code, 2004.

[FAvK+14] Noa Fish, Melinos Averkiou, Oliver van Kaick, Olga Sorkine-Hornung, Daniel Cohen-Or, and Niloy J. Mitra. Meta-representation of shape families. *ACM Transactions on Graphics*, 33(4):34:1–34:11, 2014.

[FKS+04] Thomas Funkhouser, Michael Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, and David Dobkin. Modeling by example. *ACM Transactions on Graphics*, 23(3):652–663, 2004.

[GF08] Aleksey Golovinskiy and Thomas Funkhouser. Randomized cuts for 3D mesh analysis. *ACM Transactions on Graphics*, 27(5):145:1–145:12, 2008.

[GF09] Aleksey Golovinskiy and Thomas Funkhouser. Consistent segmentation of 3D models. *Computers & Graphics*, 33(3):262–269, 2009.

[GG] Sven Grundberg and Chase Gummer. "Ubisoft Spends 5,000 Man-Hours Recreating Notre Dame". *The Wall Street Journal*. http://blogs.wsj.com/digits/2014/08/15/ubisoft-spends-5000-man-hours-recreating-notre-dame/ (accessed November 18, 2015).

[GG06] Timothy D. Gatzke and Cindy M. Grimm. Estimating Curvature on Triangular Meshes. *International Journal of Shape Modeling*, 12(1):1–28, 2006.

[HFL12] Ruizhen Hu, Lubin Fan, and Ligang Liu. Co-segmentation of 3D shapes via subspace clustering. *Computer Graphics Forum*, 31(5):1703–1713, 2012.

[HK92] Lars Hagen and Andrew B. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(9):1074–1085, 1992.

[HS01] Masaki Hilaga and Yoshihisa Shinagawa. Topology matching for fully automatic similarity estimation of 3D shapes. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 203–212, 2001.

[Jol02] Ian Jolliffe. *Principal Component Analysis, Second Edition*. 2002.

[JTRS12] Arjun Jain, Thorsten Thormählen, Tobias Ritschel, and Hans-Peter Seidel. Exploring Shape Variations by 3D-Model Decomposition and Part-based Recombination. *Computer Graphics Forum*, 31(2):631–640, 2012.

[KCKK12] Evangelos Kalogerakis, Siddhartha Chaudhuri, Daphne Koller, and Vladlen Koltun. A probabilistic model for component-based shape synthesis. *ACM Transactions on Graphics*, 31(4):55:1–55:11, 2012.

[KHS10] Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. Learning 3D mesh segmentation and labeling. *ACM Transactions on Graphics*, 29(4):102:1–102:12, 2010.

[KJS07] Vladislav Kreavoy, Dan Julius, and Alla Sheffer. Model composition from interchangeable components. *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, pages 129–138, 2007.

[KLM+13] Vladimir G. Kim, Wilmot Li, Niloy J. Mitra, Siddhartha Chaudhuri, Stephen DiVerdi, and Thomas Funkhouser. Learning part-based templates from large collections of 3D shapes. *ACM Transactions on Graphics*, 32(4):70:1–70:12, 2013.

[Kol05] Alexander Kolliopoulos. A Generic Halfedge Mesh Data Structure for .Net. 2005.

[KPNK03] Marcel Körtgen, Gil-Joo Park, Marcin Novotni, and Reinhard Klein. 3D Shape Matching with 3D Shape Contexts. *The 7th Central European Seminar on Computer Graphics*, 2003.

[KT03] Sagi Katz and Ayellet Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics*, 22(3):954–961, 2003.

[LMS13] Hamid Laga, Michela Mortara, and Michela Spagnuolo. Geometry and Context for Semantic Correspondences and Functionality Recognition in Manmade 3D Shapes. *ACM Transactions on Graphics*, 32(5):150:1–150:16, 2013.

[Lux06] Ulrike Von Luxburg. A Tutorial on Spectral Clustering. *Statistics and Computing*, 17(4):395–416, 2006.

[LV07] John A. Lee and Michel Verleysen. *Nonlinear dimensionality reduction*. Springer New York, 2007.

[Mey00] Carl D. Meyer. *Matrix Analysis and Linear Algebra*. 2000.

[Möl97] Tomas Möller. Fast triangle-triangle intersection test. *Journal of Graphics Tools*, 2(2):25–30, 1997.

[MWZ+13] Niloy J. Mitra, Michael Wand, Hao (Richard) Zhang, Daniel Cohen-Or, Vladimir Kim, and Qi-Xing Huang. Structure-aware shape processing. *SIGGRAPH Asia 2013 Courses*, pages 1:1–1:20, 2013.

[MXLH13] Min Meng, Jiazhi Xia, Jun Luo, and Ying He. Unsupervised co-segmentation for 3D shapes using iterative multi-label optimization. *CAD Computer Aided Design*, 45(2):312–320, 2013.

[NJW02] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems 14*, pages 849–856, 2002.

[NLCK05] Boaz Nadler, Stephane Lafon, Ronald R. Coifman, and Ioannis G. Kevrekidis. Diffusion Maps, Spectral Clustering and Eigenfunctions of Fokker-Planck operators. *Advances in Neural Information Processing Systems 18*, pages 955–962, 2005.

[Ray04] Vikas Chandrakant Raykar. Spectral clustering and kernel principal component analysis are pursuing good projections. Technical report, 2004.

[RS00] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[SM00] Jianbo Shi and Jitendra Malik. Normalized Cuts and Image Segmentation. *Ieee Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[SMKF04] Philip Shilane, Patrick Min, Michael Kazhdan, and Thomas Funkhouser. The Princeton Shape Benchmark. In *Shape Modeling International*, 2004.

[SSCO08] Lior Shapira, Ariel Shamir, and Daniel Cohen-Or. Consistent mesh partitioning and skeletonisation using the shape diameter function. *The Visual Computer*, 24(4):249–259, 2008.

[SvKK+11] Oana Sidi, Oliver van Kaick, Yanir Kleiman, Hao Zhang, and Daniel Cohen-Or. Unsupervised co-segmentation of a set of shapes via descriptor-space spectral clustering. *ACM Transactions on Graphics*, 30(6):126:1–126:10, 2011.

[TB97] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. 1997.

[TLGG14] Zahra Toony, Denis Laurendeau, Philippe Giguère, and Christian Gagné. 3D-NCuts : Adapting Normalized Cuts to 3D Triangulated Surface Segmentation. *Proceedings of the 9th International Conference on Computer Graphics Theory and Applications (GRAPP)*, pages 144–152, 2014.

[Tri] Trimble. 3D Warehouse. https://3dwarehouse.sketchup.com/ (accessed November 18, 2015).

[Vid11] René Vidal. Subspace Clustering. *IEEE Signal Processing Magazine*, 28(2):52–68, 2011.

[VTS+11] Oliver Van Kaick, Andrea Tagliasacchi, Oana Sidi, Hao Zhang, Daniel Cohen-Or, Lior Wolf, and Ghassan Hamarneh. Prior knowledge for part correspondence. *Computer Graphics Forum*, 30(2):553–562, 2011.

[VXZ+13] Oliver Van Kaick, Kai Xu, Hao Zhang, Yanzhen Wang, Shuyang Sun, Ariel Shamir, and Daniel Cohen-Or. Co-hierarchical analysis of shape structures. *ACM Transactions on Graphics*, 32(4):69:1–69:10, 2013.

[Wan] Yunhai Wang. The Shape COSEG Dataset. http://web.siat.ac.cn/~yunhai/ssl/ssd.htm (accessed November 18, 2015).

[WAV+12] Yunhai Wang, Shmulik Asafi, Oliver Van Kaick, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. Active co-analysis of a set of shapes. *ACM Transactions on Graphics*, 31(6):165:1–165:10, 2012.

[XLZ+10] Kai Xu, Honghua Li, Hao Zhang, Daniel Cohen-Or, Yueshan Xiong, and Zhi-Quan Cheng. Style-content separation by anisotropic part scales. *ACM Transactions on Graphics*, 29(6):184:1–184:10, 2010.

[XZCOC12] Kai Xu, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. Fit and diverse: Set Evolution for Insiring 3D Shape Galleries. *ACM Transactions on Graphics*, 31(4):57:1–57:10, 2012.

[YCHK15] Mehmet Ersin Yumer, Siddhartha Chaudhuri, Jessica K. Hodgins, and Levent Burak Kara. Semantic Shape Editing Using Deformation Handles. *ACM Transactions on Graphics*, 34(4):86:1–86:12, 2015.

[YK14] Mehmet Ersin Yumer and Levent Burak Kara. Co-Constrained Handles for Deformation in Shape Collections. *ACM Transactions on Graphics*, 33(6):187:1–187:11, 2014.

[ZCOAM14] Youyi Zheng, Daniel Cohen-Or, Melinos Averkiou, and Niloy J. Mitra. Recurring part arrangements in shape collections. *Computer Graphics Forum*, 33(2):115–124, 2014.

[ZCOM13] Youyi Zheng, Daniel Cohen-Or, and Niloy J. Mitra. Smart variations: Functional substructures for part compatibility. *Computer Graphics Forum*, 32(2):195–204, 2013.

[ZSCO+08] Hao Zhang, Alla Sheffer, Daniel Cohen-Or, Qingnan Zhou, Oliver Van Kaick, and Andrea Tagliasacchi. Deformation-driven shape correspondence. *Proceedings of the Symposium on Geometry Processing*, pages 1431–1439, 2008.

[ZVD10] Hao Zhang, Oliver Van Kaick, and Ramsay Dyer. Spectral mesh processing. *Computer Graphics Forum*, 29(6):1865–1894, 2010.

Acronyms

AGD Average Geodesic Distance. 19, 20, 22, 23, 39

APS Anisotropic Part Scale. 25

CF Conformal Factor. 20, 22, 23, 39

GC Gaussian Curvature. 20, 22, 39

OBB Oriented Bounding Box. 25, 26, 29, 43–45, 48–50, 52, 56, 57

PA Pair Arrangement. 27

PCA Principal Component Analysis. 9, 12, 28, 33, 38, 42, 43, 46, 55, 67

PDF Probability Density Function. 29

SC Shape Context. 21–23, 39, 55, 56

SDF Shape Diameter Function. 20–23, 39

SFARR Symmetry Functional Arrangement. 34–36