FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Parameter Spaces of Cups

Cluster-based Exploration of a Geometry Generator's Parameter Space

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Michael Beham
Matrikelnummer 0726417

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Mitwirkung: Dipl.-Ing. Johannes Kehrer, PhD

Wien, TT.MM.JJJJ

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Parameter Spaces of Cups

Cluster-based Exploration of a Geometry Generator's Parameter Space

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Michael Beham
Registration Number 0726417

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Assistance: Dipl.-Ing. Johannes Kehrer, PhD

Vienna, TT.MM.JJJJ

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Michael Beham
Kölblgasse 14 Tür 11, 1030 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.

(Ort, Datum) (Unterschrift Verfasser)

Danksagung

In den folgenden Zeilen möchte ich mich bei allen Menschen bedanken, welche mich während der Diplomarbeitet unterstützt haben. Ohne deren Unterstützung wäre die Durchführung der Arbeit nicht möglich gewesen.

Zuerst möchte ich mich bei meinem Betreuer **Johannes Kehrer** bedanken. Seine ermunternde Art, sein großer Einsatz, sowie sein umfangreiches Wissen haben mich umfangreich während der Diplomarbeit unterstützt. An dieser Stelle möchte ich ihm auch nochmals zu seiner Vermählung gratulieren und im viel Erfolg für seine wissenschaftliche Laufbahn wünschen. Außerdem möchte ich mich bei **Meister Eduard Gröller** bedanken. Er lektorierte nicht nur die Diplomarbeit, sondern hat mit zahlreichen Ratschlägen die Diplomarbeit umfangreich bereichert. Desweiteren hat er auch mich ermuntert, diese Arbeit bei der IEEE VIS 2014 einzureichen, wo ich diese Arbeit vor der internationalen Visualisierungs-Community im Bereich Visual Analytics Science and Technology (VAST) präsentieren konnte [8].

Diese Diplomarbeit wurde gemeinsam mit dem **AIT - Austrian Institut of Technology** durchgeführt. Besonders möchte ich mich bei meinem Projektleiter **Wolfgang Herzner** bedanken, welcher mir die Chance gab, in seiner innovativen Forschungsgruppe zu arbeiten. Gemeinsam mit den Arbeitskollegen **Markus Murschitz** und **Oliver Zendel**, welchen ich ebenfalls meinen Dank ausspreche, gab mir Wolfgang nicht nur sehr wertvolles Feedback, sondern hat meine Anwendung auch evaluiert. Ohne der kollegialen Arbeit in der Forschungsgruppe hätte ich weder ausreichend Wissen über das behandelte Anwendungsgebiet, noch das Wissen über die besonderen Erfordernisse erwerben können.

Außerdem möchte ich mich an dieser Stelle bei meiner Familie bedanken. Ohne deren Unterstützung hätte ich weder das Studium absolvieren, noch die Diplomarbeit durchführen können. Besonders möchte ich aber an dieser Stelle meine Eltern **Regina** und **Leopold Beham** hervorheben. Sie finanzierten nicht nur meine akademische Ausbildung, sondern haben mich über das gesamte Studium, besonders in schwierigen Situationen, unterstützt. Ihre umfangreiche Unterstützung ermöglichte überhaupt meine akademische Ausbildung. Auch möchte ich mich bei meiner Schwester **Birgit** bedanken. Sie diente nicht nur als Vorbild für schulische Leistungen, sondern hat mir auch Mut in schwierigen Situationen zugesprochen und mich mit guten Vorschlägen unterstützt.

Nicht vergessen möchte ich meine Freunde und Studienkollegen, welche mich während des Studiums begleitet haben. Besonders hervorheben möchte ich aber meine Mitbewohner **Elisabeth Staudigl**, **Fabian Ladurner**, **Dominik Hörner** und **Myriam Boubaker**. Sie sind nicht nur meine Mitbewohner und Freunde, sondern waren auch immer mit Rat zur Seite.

Abstract

Geometry generators are commonly used in video games and evaluation systems for computer vision to create geometric shapes such as terrains, vegetation or airplanes. The parameters of the generator are often sampled automatically which can lead to many similar or unwanted objects. In this thesis, we propose a novel visual exploration approach that combines the abstract parameter space of the generator with the resulting geometric shapes in a composite visualization. Similar 3D shapes are first grouped using hierarchical clustering and then displayed in an illustrative parallel coordinates or scatterplot matrix visualization. This helps the user to study the sensitivity of the generator with respect to its parameter space and to identify invalid regions. Starting from a compact overview representation, the user can iteratively drill-down into local shape differences by clicking on the respective clusters. Additionally, a linked radial tree gives an overview of the cluster hierarchy and enables the user to manually split or merge clusters. We evaluate our approach by exploring the parameter space of a cup generator and provide feedback from domain experts.

Kurzfassung

Geometriegeneratoren werden häufig in Videospielen und Evaluierungssystemen für Maschinelles Sehen eingesetzt um 3D Geometrie wie Terrains, Vegetation und Flugzeuge zu erstellen. Die Generatoren werden durch Parameter gesteuert. Diese werden oft automatisch abgetastet, um verschiedenste Variationen eines Objektes zu erzeugen. Dies führt aber oft zu sehr ähnlichen oder unerwünschten Objekten. In dieser Diplomarbeit wird ein neues Visualisierungssystem vorgestellt, welches die Analyse des abstrakten Parameterraumes eines Geometriegenerators gemeinsam mit den resultierenden Geometrieobjekten mithilfe neuer Visualisierungen darstellt. Ähnliche Objekte werden zunächst mittles Hierarchischem Clustering gruppiert und anschließend in illustrativen Parallelen Koordinaten dargestellt. Dies ermöglicht dem/der BenutzerIn die Sensitivität einzelner Parameter zu analysieren und Regionen im Parameterraum zu identifizieren, welche unerwünschte Ergebnisse liefern. Die Visualisierung startet mit einer kompakten übersichtlichen Darstellung und der/die BenutzerIn kann interaktiv die lokalen Unterschiede durch klicken auf den gewünschten Cluster analysieren. Zur erhöhten Übersichtlichkeit wird auch eine radiale Baumdarstellung der Cluster angeboten. Desweiteren können Cluster vereinigt und gesplittet werden. Das System wurde von Fachleuten evaluiert, wobei ein Tassengenerator analysiert wurde.

Contents

CHAPTER 1

Introduction

In the recent years, computer vision (CV) applications have found their way out of science laboratories and have become part of everybody's life. Cars are equipped with advanced driver assistance systems, drones observe public places to ensure the public safety, and robots are used for quality control in factories. Computer vision applications act in the real world, and a safe operation mode is very important to prevent accidents and malfunctions of the robots. So, testing of the CV systems is of particular importance. If every test case takes place in the real world, this task is very expensive and time intensive. The obvious idea is to use computer-generated simulations to reduce the amount of time to test and the costs.

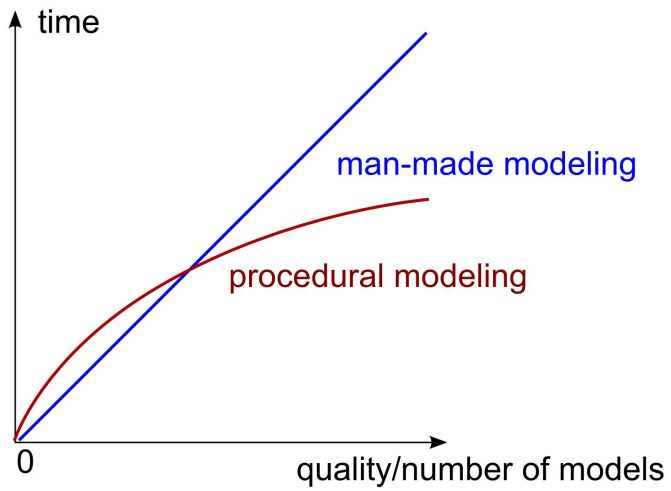

Figure 1.1: The function plot compares hand-made and procedural modeling. If a high quality or many shapes is needed, procedural modeling is useful [71].

Figure 1.2: Categories of airplanes.

To test a CV system, usually a computer generated 3D scene is created. This scene consists of different geometric shapes, which illustrate the simulated world. Optionally, the shapes are animated. Then, a test-case can be performed using the generated scene. Generating each test case by hand is time intensive. To speedup this process a test case simulator is often used. For evaluation systems of computer vision a common test case is testing different variations of an object (e.g., different cars). The objects can be created by an artist. However, creating many different variations of an object is time-consuming. A more reasonable approach is to create the shapes with procedural modeling.

Procedural modeling is commonly used for the automatic creation of 3D models [91]. Such techniques are used to create models, which are too complex to build for a person like natural objects such as clouds [42], trees [79] and landscapes. However, human-made objects can also be created like cities, buildings, and cars [75]. As shown in Figure 1.1 procedural modeling is useful, if many different variations of an object, or very complex models have to be generated. So, this technique is also used in other application areas. It is commonly used in video games, and all other field, where photorealistic worlds should be generated (movies, commercials, etc.).

Procedural modeling refers to a wide variety of techniques [91]. For creating landmarks or clouds usually a noise generator [84] or simulation techniques are used [42]. Flowers and trees are created using L-Systems, which consist of some basic shapes and a grammar that describes the overall property of the model [79]. Similar techniques are used to create houses and cities [75]. In this thesis, an application using procedural modeling is called a geometry generator.

Even through the shapes are generated with simulations, noise generators, or L-Systems, a parameter space is often used to control the algorithm. Changing of the parameter results in different variations or complexities of the resulting shapes. In this thesis, we present a novel visualization system to analyze the parameter space of a geometry generator. Using our new

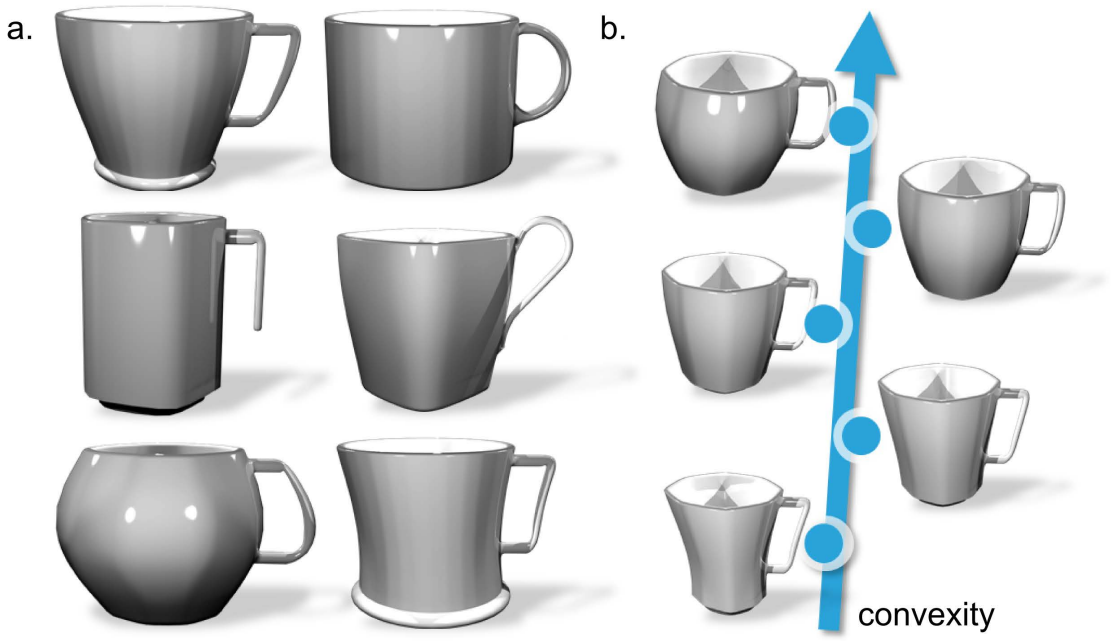

Figure 1.3: (a) Some categories in the cup data set. (b) some cups with different parameter settings of parameter *convexity*

tool enables the user, to get new visual insights of the geometry generator's behaviour. In the next section, we give a detailed problem statement.

1.1 Problem Statement

Geometry generators create polygonal meshes that approximate a specific domain of geometric shapes, e.g., cars, airplanes or vegetation. Therefore, they are often used in video games to generate realistic worlds. For example, every tree in a 3D scene has a different shape by varying the parameters of the generator [79]. Computer vision (CV) evaluation systems also use geometry generators to create virtual test cases, for example, to evaluate the recognition of different shapes of an object [104].

Geometry generators usually have a set of parameters that determine the shape of a generated object. A parameter can be a categorical (e.g., engine type) or continuous attribute (e.g, size of the wing). From a mathematical viewpoint, a geometry generator is a function f: $\mathbf{R}^n \rightarrow \mathbf{R}^m$, where n is the number of parameters, and m is the dimensionally of the resulting shapes. In our case, we use 3D shapes, but geometry generators can also create 2D or time-varying shapes [79]. For creating shapes, the parameter space of a generator is usually sampled automatically. A general goal in this context is to understand the relations between the parameter space and the resulting geometric shapes [11]. In particular, we identify three tasks for analyzing geometry generators:

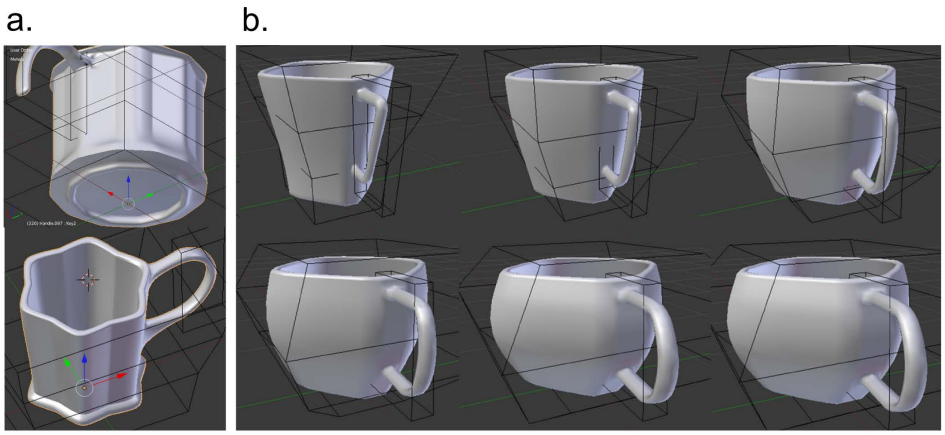

Figure 1.4: Some examples of the cup data set: (a) shows examples of unwanted cups. The top cup looks like a beer mug. The bottom cup has a very big handle, which results in an unstable cup. (b) shows the result of the *convexity side* parameter. A constant change of the parameter does not result in a constant change of the shape.

Task 1: Find similar 3D shapes and the corresponding parameter settings:

The user wants to identify which regions of the parameter space create certain types of similar 3D shapes. In the case of airplanes, for example, different types of airplanes would be military jets, airliners, and small airplanes. Variations within the type military jet can be MIG-15, MIG-29, and Sukhoi Su-27 (see Figure 1.2). However, we are typically not interested in small variations of the geometric shape, for instance, whether an airplane has round or squared windows. Examples of different cup categories are shown in Figure 1.3a.

Task 2: Find errors and implausible 3D shapes:

Geometry generators can produce (physically) implausible or unwanted results. For example, the appropriate profile of a wing is important for the ability of the plane to fly. The generated meshes may also contain errors such as holes or self intersections of the surface. However, not only one parameter region can produce unwanted results but a combination of parameters too. Our approach helps the user to identify such problematic regions in the parameter space by directly relating the parameters of the generator and the created 3D shapes in the visualization. Examples of invalid cups are shown in Figure 1.4a.

Task 3: Determine the sensitivity and influence of parameters on the result:

It is often difficult to predict how changes in one or more of the parameter values influence the resulting geometric shapes. Changes may be global or local, depending on the parameter (e.g., changing the type of an airplane vs. changing the number of windows). Additionally, the sensitivity of a parameter may not be linear. That is, for certain regions of the parameter space

even small variations may have a drastic effect on the resulting 3D shapes, while changes in less sensitive regions may have little to no effect on the created geometry. Examples of different cups of parameter *convexity* are shown in Figure 1.3b and Figure 1.4b.

1.2 Contribution

In their previous workflow, our domain experts of testing computer vision systems (CV) would manually study thumbnails of the generated 3D shapes in an image gallery and look up the corresponding parameter settings in a table. This is cumbersome and time-consuming, especially in the case of many generated 3D shapes.

Motivated by the tasks described above, we introduce a new approach called *Cupid*. We want to efficiently explore the results of a geometry generator. We propose a novel visual exploration approach that combines the abstract parameter space of the geometry generator with the resulting geometric shapes in a composite visualization. The abstract parameter space is visualized using a composite parallel coordinates or a composite scatterplot matrix approach. We nest the geometric information using icons. This combination helps the user to study the sensitivity of the geometry generator with respect to its parameter space and to identify invalid regions. We use a hierarchical clustering approach for preventing visual clutter. Starting from a compact overview representation the user can interactively drill-down a cluster of interest to explore all details. Additionally, we use a composite radial tree approach to visualize the hierarchical clustering.

In this context, this thesis represents a close collaboration between visualization researchers and domain experts (one of the paper authors has developed both a geometry generator for cups and *Cupid*). The contributions of this work are as follows:

- We propose a composite visualization that combines both the abstract parameter space of the geometry generator and the resulting 3D shapes in a single visualization. This enables the user to study relations between the parameter space and the created geometry.
- Adapting hierarchical clustering, the user can interactively drill-down into groups of similar 3D shapes (compare to Task 1).
- We evaluate our approach according to Task 1-3 by exploring the results of a cup generator and provide feedback from domain experts.

1.3 Thesis Structure

This thesis is organized as follows: In the next chapter the *state of the art* of the *exploration of a set of geometric shapes* and *parameter studies* is presented. In Chapter 3 the basic *methods*, which are used, are presented, and their implementation is explained. In Chapter 4 a conceptional *overview of Cupid* is given and the preprocessing steps are presented. The *composite visualizations* are in focus of Chapter 5 and the *geometric properties* in Chapter 6. Chapter 7 gives an overview of the system architecture, and all relevant *implementation* details are discussed. The *results* are presented in Chapter 8. In Chapter 9, we *discuss* our solution in detail. In Chapter 10 a *summary* is given, as well as future perspectives of research.

Parts of these thesis are based on: M. Beham, W. Herzner, M.E. Gröller and J. Kehrer: Cupid: Cluster-Based Exploration of Geometry Generators with Parallel Coordinates and Radial Trees, IEEE Transactions on Visualization and Computer Graphics, 2014 [8].

CHAPTER 2

State of the Art

This chapter covers the most sophisticated works of the visualization and exploration of a set of geometric shapes and parameter studies. Related work to the exploration of a set of geometric shapes is presented in Section 2.1. In Section 2.2, several works, which enable parameter studies, are discussed. A conclusion is given, and open problems are discussed in Section 2.3.

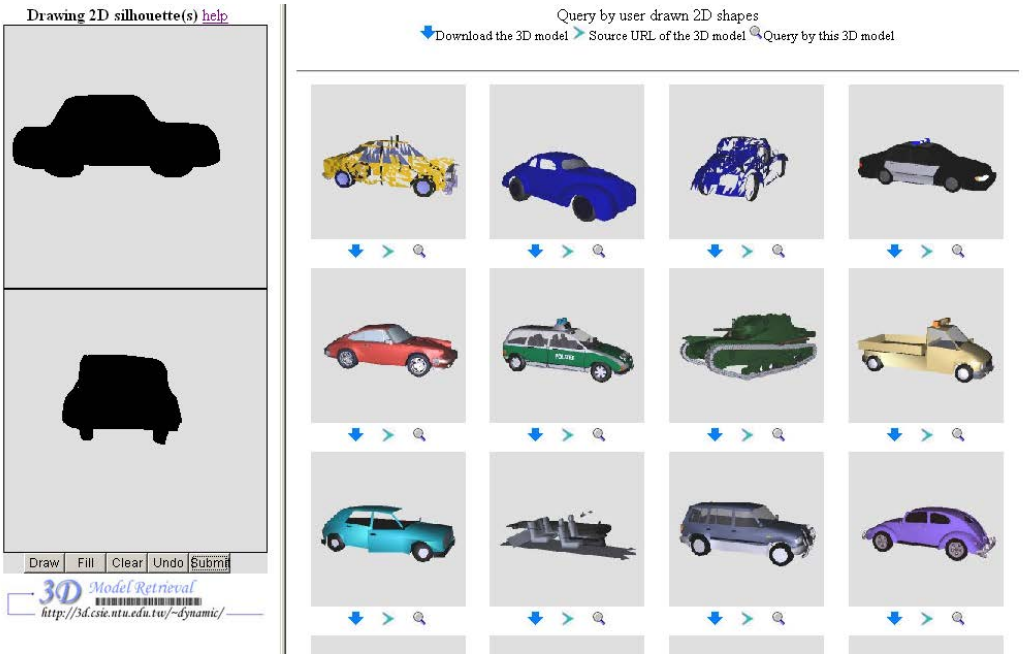

Figure 2.1: An example of a retrieval system: The resulting 3D shapes are similar to the user-drawn 2D shape [27].

Figure 2.2: Example of context-based searching [37]. The results depend on the location and size of the query box.

2.1 Visualization and Exploration of a Set of Geometric Shapes

The visualization and exploration of a set of geometric shapes are still an active area of research. While some works try to improve the creation process of a 3D scene, other techniques support the user by creating 3D shapes or provide techniques for analyzing simulations. So, recent works focus primarily on the exploration of many heterogeneous geometric shapes of different categories (for creating 3D scenes), and the exploration of geometric shapes within a category (for creating 3D shapes, analyzing simulations).

In the following, several querying techniques for the exploration of a set of 3D shapes are presented, as well as techniques that enable to explore a set of shapes by manipulating a template object. Then, some approaches, which deal with the parameter space of a set of geometric shapes, are presented, and in the last section some open problems are discussed.

Querying-based Exploration

Several researchers have proposed methods for exploring of a set of geometric shapes by querying. A common method is to (automatically) collect text keywords to the 3D shape and offer a keyword search (see Fisher and Hanrahan [37] for an overview). Such applications enable to enter a text query. Then, a set of geometric shapes is returned. The geometric shapes are ranked based on how close the query matches a set of tags associated with the object [37].

If the user looks for a particular shape, keywords, describing the shape are hard to find. Some research proposes methods using sketches of 3D shapes [27]. The user gives an example (sketch or 3D shape) and a similarity-based query returns a list of similar geometric shapes.

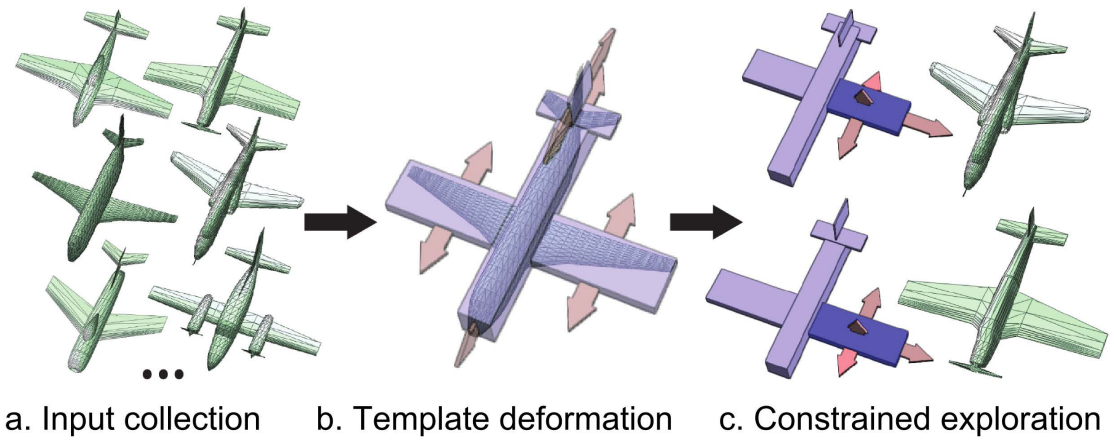

Figure 2.3: Exploring collections of 3D shapes [72]: (a) The database of similar 3D shapes. A analysis automatically extracts a deformation model that characterizes variability in a template shape. (b) The deformation model provides a constrained manipulation interface for exploring the collection. (c) The manipulation versus the resulting shapes.

Such techniques reduce the geometric shapes into a feature space that is used to compare two shapes. An example is given in Figure 2.1. This method is typically used for retrieval systems.

An alternative query-based approach is presented by Fisher and Hanrahan [37]. Their approach provides context-based queries. Usually, the user searches for a shape that is well suited for a particular 3D scene. Instead of querying with keywords, sketches or examples, their systems suggest shapes, which are well suitable for the created 3D scene. The user specifies a region in the scene, where the shape should be placed (using a simple bounding box). Then, the system returns a list of well-suited shapes. An example of this application can be seen in Figure 2.2. The results differ according to the place and the size of the bounding box.

Exploration of Shapes using Creation and Manipulation

While keywords and shape examples enable to choose between a broad categorization (e.g., distinguishing between airliners and cars) the exploration of finer scale variations within a category is still a challenge. Funkhouser et al. [39] enable to create new shapes using parts of other shapes. The user defines a region of interest, which should be replaced. Then, their system searches in a large database of 3D shapes to find alternative parts. The user selects the desired part, which is then cut out of the shapes with intelligent scissoring. The scissored parts are then used to create a new geometric shape [39]. A similar system is presented by Xie et al. [103].

Umetani et al. [96] present an interactive framework, which checks the validity of a 3D shape, while creating/editing. They define several constraints that the generated 3D shape has to fulfill (e.g., stability). If one or more constraints are violated after a user manipulation, the system generates multiple suggestions to restore the validity of the shapes. The advantage of

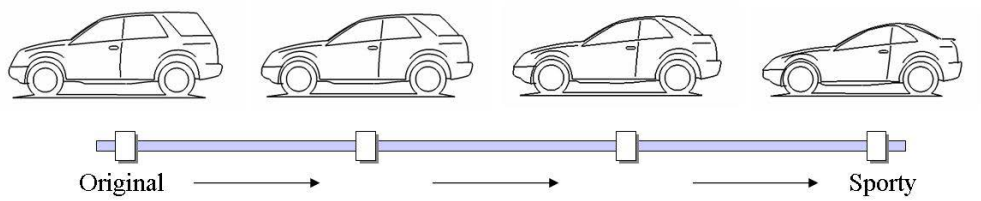

Figure 2.4: Adding increasing amounts of 'sporty' to an initial vehicle [92].

their system is that the user can focus on the esthetic aspects of the design while the system helps to achieve physical plausibility.

Ovsjanikov et al. [72] and Smith et al. [92] enable the exploration of a set of 3D shapes using template objects. The user can deform a template model, and the corresponding object is shown. An example of this approach is shown in Figure 2.3. Coffey et al. [30] extend this technique to explore a set of simulations or visual effects. By dragging parts of an object, the user can explore different simulations.

Parametric Spaces of 3D/4D shapes

Besides the exploration of 3D geometric shapes, in some applications the exploration of shapes with additional abstract properties is important. For example, Smith et al. [92] add parameters like sporty and weight to a set of cars. The user can vary the attributes to generate new shapes. An example of the effect of sporty is shown in Figure 2.4. A similar system is created by Blanz and Vetter [15] to model 3D faces. Just like Smith et al. they use a set of morphable shapes to vary a shape. Each shape can be adjusted by changing the corresponding parameters. Their system enables to reconstruct 3D faces from single images. Then, photo-realistic image manipulations can be performed in the 3D space by changing the corresponding parameters. Allen et al. [1] extend this approach to entire human models. They reconstruct a space of body shapes from 250 scanned human figures using a sophisticated registration algorithm. Examples of the parameter space of human bodies are shown in Figure 2.5.

Killian et al. [61] presents another approach for exploring a shape space. The input shape space is created by spanning a number of input shapes. This input space is projected on a 2D polygon by metric multidimensional scaling. So, each corner of the 2D polygon represents an input shape. The user can explore the shape space by drawing a curve on the 2D polygon. Each point of the curve results in an output shape. The output shapes are created by using interpolation between and exploration from the input shapes [61]. Note that Sloan et al. [90] use a similar idea for creating new 3D shapes. New 3D shapes are created by interpolation between several examples.

Instead of visualizing a parameter space of shapes, Coffey et al. [30] and Bruckner and Möller [21] deal with the parameter space of simulations. Coffey et al. [30] visualize the attributes of a simulation using a wing plot. The simulations can be selected with varying the attributes within the wing plot. Bruckner and Möller [21] present a result-driven exploration

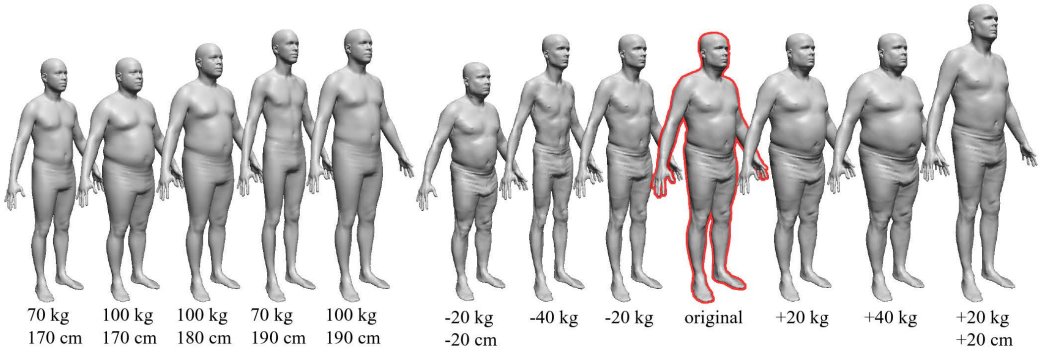

Figure 2.5: The parameter space of human bodies [1]. The bodies vary in size and weight.

approach for exploring physically based multi-run simulations. They split animations (created with different parameter settings) into similar segments. The segments are then grouped using a density-based clustering algorithm. There approach supports the user in finding of a parameter setting to achieve a specific simulation behaviour.

Marks et al. [68] introduce with Design Galleries a general concept for exploring multi-dimensional parameter spaces. Their system projects a multi-dimensional parameter space using multi-dimensional scaling. As basic element, they represent an object by little thumbnails. Additionally, some results are shown on the side of the visualization, which are linked with lines to the corresponding thumbnail. The focus of their application is to find good parameter settings for an algorithm.

Talton et al. [93] present a collaborative tool to get high quality and alternative models of a geometry generator depending on a given example. They use a linked representation of the modeling tool and a map that shows alternative 3D shapes. The 3D shapes are placed in a semantic map depending on the similarity. The map can be adjusted by an administrator. An example of the semantic map is shown in Figure 2.6. To detect high-quality models in the parameter space, the activity of the user is tracked additionally.

The idea of using abstract data to explore a set of spaces is also used by Busking et al. [24]. They enable the exploration of 3D shapes using a 2D scatterplot and replace the data points with 3D shapes. Additionally, linked views for the evaluation of a shape and comparison of different shapes are provided.

Exploration of Abstract Data using Composite Views

Balabanian et al. [4] extends the idea of combining abstract and spatial data to hierarchical data. They nest the spatial data from a computer tomography into a graph representing the hierarchy of the body parts. An example of their approach is shown in Figure 2.7. Schmidt et al. [83] perform a hierarchical clustering in order to identify differences between sets of images. This approach also enables to drill-down into the hierarchy in order to evaluate the data set.

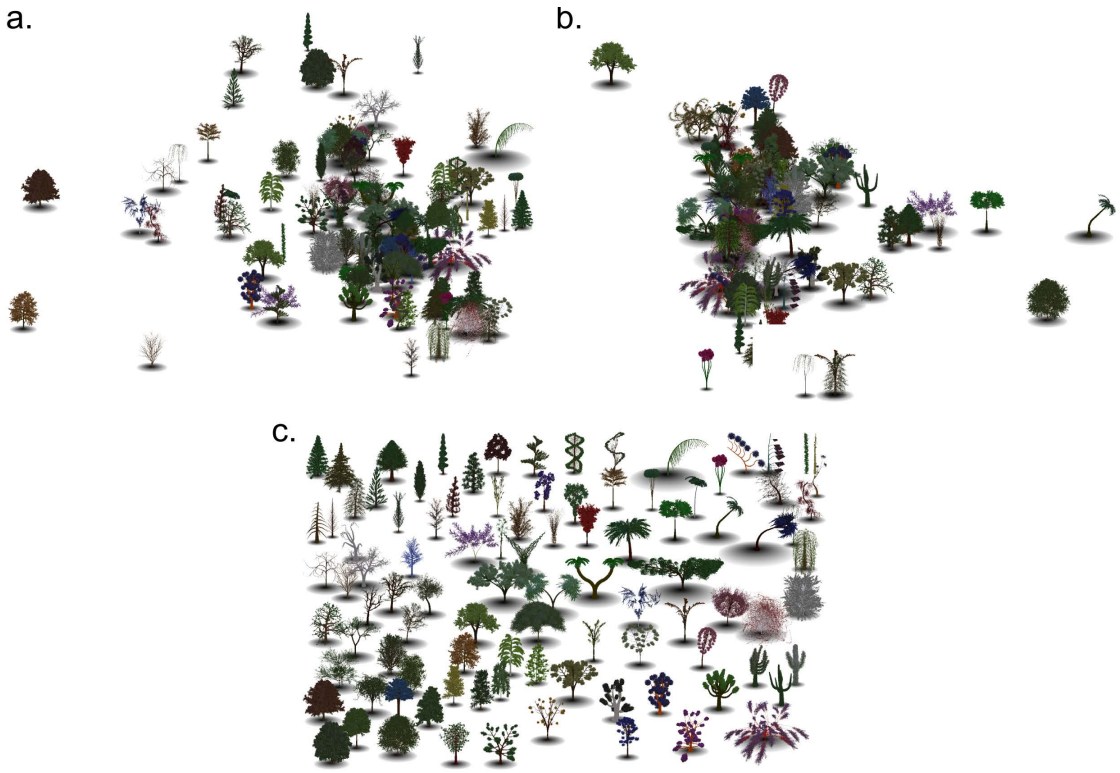

Figure 2.6: Comparision between (a) principal component analysis (PCA), (b) multi-dimensional scaling (MDS), and (c) the semantic map presented by Talton et al. [93].

Open Challenges

Almost all applications presented in the previous section use a similarity measurement. As mentioned by Ovsjanikov et al. [72] measuring similarity is still an open problem. Finding a good similarity measurement is very difficult. While some methods perform well for defining the similarity for a broad classification of objects, others perform well for defining the similarity within a category. A lot of research in defining the similarity between objects has been done in the last years. A solution, which works for all application areas is still not found. We discuss this problem in more detail in Section 3.8.

Another open challenge is to perform a detailed study of the parameter space. The works focus primarily on the exploration of a set of geometric shapes and using abstract parameters mainly to select shapes. Analyzing the parameter space of an underlying geometry generator is not studied sufficiently. We discuss the usefulness of the presented work in more detail in Section 2.3.

Our work differs from the approaches described above in that we focus on a more complex parameter space. We provide a new integration of spatial data into an abstract space using parallel coordinates. We not only explore a set of shapes, but also provide techniques for evaluating the complex relationship between the parameter space and the generated shapes. The thesis uses

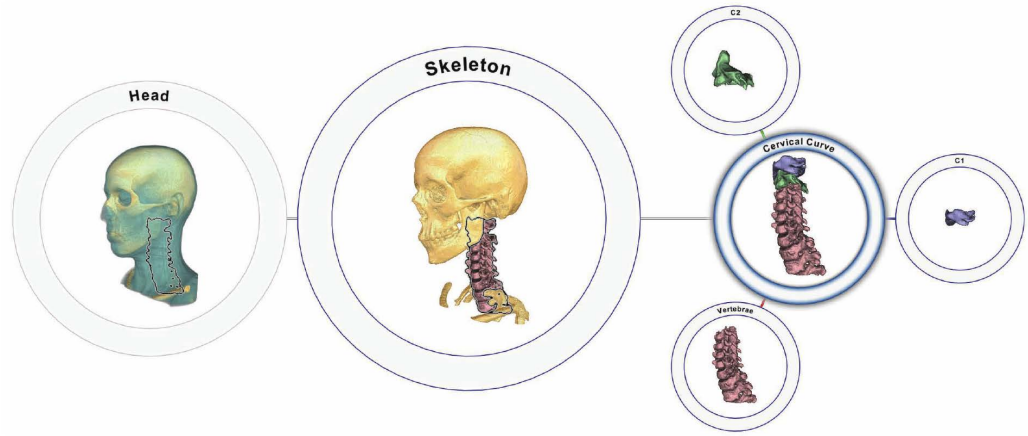

Figure 2.7: Hierarchical visualization of segmented head and neck. The cervical curve is focused by showing its relative position in the neck and highlighting its substructures [4].

methods from parameter studies of other application areas to provide methods of analyzing the parameter space of a set of shapes.

2.2 Parameter Studies

Parameter studies are successfully used in several application areas. Usually, the parameter space is studied in a systematic way by varying parameters.

One of the first approaches is HyperSlice [97]. HyperSlice is created to study scalar functions with a multi-dimensional input space. Similar to a scatterplot matrix, the multi-dimensional parameter space is visualized by orthogonal 2D slices, showing each combination of two parameters. In contrast to a scatterplot matrix, only a part around a focal point is shown. The user navigates through the parameter space by manipulation of the focal point. The idea of HyperSlice is extended by DosSantos and Brodlie [33], Piringer et al. [76] and Berger et al. [11]. Dos Santos and Brodlie [33] replace the matrix layout with a graph interface. Each node represents a 1D, 2D or 3D plot. Moreover, they provide multiple focal points. Piringer et al. [76] also builds upon the concept of HyperSlice. They introduce HyperMoVal to validate regression models. HyperMoVal combines the multi-dimensional functions and known validation data [76]. Berger et al. [11] extend HyperMoVal for analyzing the continuous parameter space of a simulation. They map the local neighborhood around the focal point in the input parameters to the output domain. A guided navigation enables to predict target ranges within a small input region as well as a sensitivity analysis.

Booshehrian et al. [17] presents Vismon an application to support data analysis of fisheries management. They provide a sensitivity analysis, comprehensive and global trade-offs analysis, and an approach to visualize uncertainty of the underlying simulation model [17]. Torsney-Weir et al. [94] present with Tuner an application to find well-suited parameter settings for image segmentation algorithms. Pretorius et al. [78] use a similar idea to determine the parameter settings

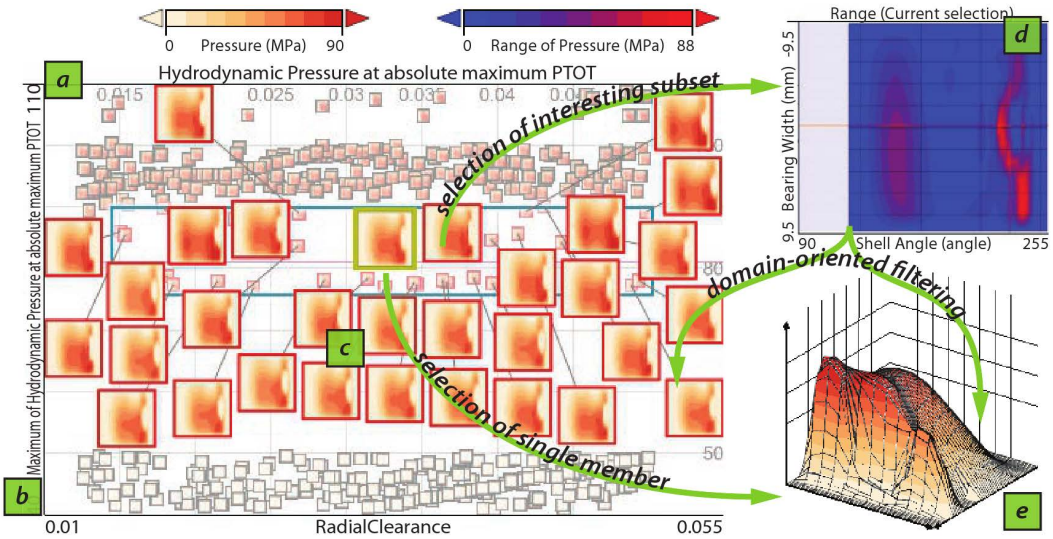

Figure 2.8: Visualization of 2D funcion ensembles [77]: (a) the member-oriented overview employing feature-based placement (b) for 524 2D functions; (c) the mid-level focus of 31 selected members; (d) the domain-oriented overview showing the point-wise range of the selected subset; (e) the 3D surface plot of a single member. The arrows indicate key interactions for linking the parts [77].

of image analysis algorithms. The earlier presented techniques from Bruckner and Möller [21] and Marks et al. [68] also use the ideas of parameter studies to find good parameter settings.

Finally, Piringer et al. [77] present an approach for analyzing 2D function ensembles. The 2D functions are represented by icons that are nested into 2D scatterplots, which depict the simulation parameters. The icons can be analyzed at different levels of detail using brushing. An example of their system is shown in Figure 2.8. While their approach uses 2D scatterplots to represent the parameter space of the simulation, we deal with a higher dimensional parameter space using parallel coordinates. Additionally, we focus on the analysis of similar shapes using hierarchical clustering.

In the literature, there are a lot of other parameter studies. Also, other approaches use multiple linked views to study the complex relationship between input and output of a simulation model [57, 58, 69].

2.3 Conclusion

In Section 1.1, we present three tasks for exploring the results of a geometry generator. The first task is to categorize the shapes and to find the corresponding parameter values. Then, invalid and unwanted shapes should be detected. The third task is to perform a sensitivity study of the parameter space. In the following, the presented works are analyzed concerning their usefulness for the task.

The first task is to find similar geometric shapes and to find the corresponding parameter settings. Finding similar 3D shapes can be done by using the querying methods like keywords or proxies of shapes. However, these techniques do not provide an overall visualization. The techniques using template objects also miss an overview representation. However, an overview visualization of the shapes is very important because we want to compare many shapes.

Other techniques like Design Galleries [68] and Talton et al. [93] provide such an overview using a map representation. While Design Galleries use the well-known multi-dimensional scaling (MDS), Talton et al. [93] use a semantic approach to layout the map. The MDS layout algorithm projects the parameter space to a 2D map, with preserving the global distances between the shapes. The distance between two shapes is defined by their similarity. So, a cluster of similar shapes is located in a small region of the map. The semantic map, presented by Talton et al. [93] , sorts the shapes by similarity. Zooming is supported. While at the beginning only an overview visualization is given, the user can explore details by zooming into a specific region. The map visualization is very useful to categorize the shapes. With a principal components analysis (PCA), and the MDS map, clusters of similar shapes are located in a small area. In the semantic map visualization, a representative of a cluster is shown in the overview visualization. Examples of the PCA, MDS, and semantic map is shown in Figure 2.6.

If the parameter space is categorized, the parameter regions have to be identified. Some approaches use additional linked representations to visualize the corresponding parameter settings. For instance, Coffey et al. allow selecting simulations with a wing plot. Talton et al. [93] provide a semantic map, which is linked with the corresponding geometry generator. If the user selects an object in the semantic map, the parameter settings of the object are passed to the geometry generator. The user can explore the geometry generator by small modifications of the parameters.

The second task is to find invalid objects. While Umetani et al. [96] present an approach, which checks the validity of shapes at runtime, the other approaches do not provide any support for this. In these approaches, the users have to do a manual checking of the validity of geometric shapes. The collaborative approach from Talton et al. [93] may support this task because they track the popularity of shapes. Invalid objects would not be used after a manual checking by the artists. So, the application would recommend other geometric shapes.

The third task is to perform a sensitivity study and to determine the influence of parameters. The methods presented in Section 2.1 are not able to perform such tasks. Most applications provide no or only a few methods to explore the abstract parameter values. Only Busking et al. [24] provide a scatterplot to analyze the parameter space. However, a scatterplot only visualizes two parameters. Comparing different parameters with only one scatterplot is very hard.

The presented works for the exploration of a set of 3D shapes are partially usable to analyze our tasks. Especially, the applications do not provide methods for sensitivity studies. In contrast to the exploration of 3D shapes, the sensitivity analysis is a very important topic in parameter studies research. For example, a great application to study sensitivity is presented by Berger et al. [11]. They propose a linking and brushing method to analyze 1D CFD simulations (e.g., for timing of fuel injection) [11]. Using linking and brushing enables to follow input samples to the corresponding output and vice versa. The linking and brushing method enables an easy sensitivity study. If the region in the input space result in a small region in the output space, the

input space is not sensitive. If the region in the input space covers a large region in the output space, the input space is sensitive. However, parameter studies are usually done to analyze simulations or an algorithm. Analyzing a set of geometric shapes needs a novel visualization system.

In summary, the most sophisticated works of exploring a set of geometric shapes miss methods to analyze the corresponding parameter space. Primarily, the used techniques focus to explore shapes using querying and modifying of template objects. The corresponding parameter values are visualized using linked views, but do not provide methods to do research in more detail, like a sensitivity analysis. In contrast, the examples of parameter studies provide great techniques to analyze a parameter space. The main idea of our application is to combine the techniques from the parameter studies and the exploration of geometric shapes. While, most parameter studies analyze simulations, we analyze the parameter space of a geometry generator.

CHAPTER 3

Methods

In this chapter the basic techniques are presented that are used to study the complex parameter space of the geometry generator. The created application is called *Cupid*, and it is presented in the next chapters. An overview of *Cupid* is given in Figure 3.1. The application visualizes the abstract multi-dimensional parameter space, as shown in Figure 3.1a. Methods to represent multi-dimensional data are the focus of Section 3.1. Very important techniques, such as small multiples, scatterplots, and parallel coordinates are presented in Sections 3.2-3.4. In order to ensure a good readability, the amount of data to display has to be reduced. *Cupid* groups similar 3D shapes with a hierarchical clustering approach. Figure 3.1b shows a representation of the

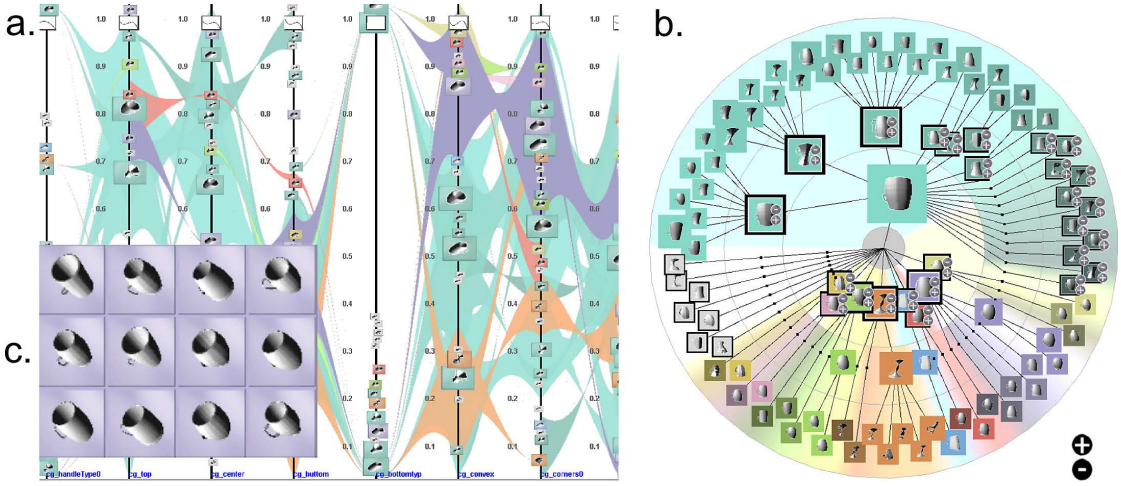

Figure 3.1: Overview of *Cupid*. (a) Composite parallel coordinates are shown that directly relate the parameter space of a geometry generator and the clusters of similar generated 3D cups. (b) Radial tree depicting the hierarchy of the clusters. (c) If a user clicks on an icon, a detail window, showing all members, is displayed.

hierarchical clustering. Methods to display hierarchical data are presented in Section 3.5. Both visualizations nest the 3D shapes using icons. This is an example for a composite visualization. This design pattern is used to combine different visualizations in one geometric space [54]. We give an overview of composite design patterns in Section 3.6. Both layouts are linked. Interaction in one visualization is also shown in the other representation. In Section 3.7 we present several interaction methods. In Figure 3.1c the members of a cluster are shown in a detail window. *Cupid* is using a similarity-based clustering approach to group similar shapes. More information about measuring the similarity of geometric shapes and clustering techniques is given in Section 3.8.

3.1 Visualization of Multi-Dimensional Data

An important part of *Cupid* is the visualization of the multi-dimensional abstract parameter space. The effect of each parameter also depends on the other parameters. It is not sufficient to visualize only one dimension of the higher dimensional data. False interpretations of the higher dimensional data can occur (compare it with mix effects presented by Armstrong and Wattenberg [3]). We need visualization techniques to display the complete multi-dimensional parameter space. However, the visualization of higher dimensional data is a challenging problem. According to Keim [59], the most sophisticated methods for visualizing multi-dimensional data can be categorized as follows:

Standard 1D/2D/3D visualizations / Small multiples A common method to display multi-dimensional data is the use of standard 1D/2D/3D visualization techniques. The basic idea is to apply popular techniques, which are used to visualize 1D/2D/3D data, for representing a multi-dimensional space [59]. Common examples for visualizing 1D/2D/3D data are bars (see Figure 3.2a), boxplots (see Figure 3.2b), histograms (see Figure 3.2c), arrows (see Figure 3.2d), 1D/2D function plots (see Figure 3.2e,f), thumbnails (see Figure 3.2g), and star glyphs (see Figure 3.2h). To use 1D/2D/3D techniques for visualizing multi-dimensional data, a common approach is to show the same representation for different objects side-by-side. Common examples of this technique are permutation matrix, survey plot, and scatterplot matrices.

Geometrically transformed displays: Geometrically transformed display techniques comprise all techniques, which aim to find a meaningful transformation of a multi-dimensional data set [59]. Examples of such techniques are scatterplot, parallel coordinates, and starplots. Such data are often analyzed using coordinated multiple views (see Roberts [81] for an overview). Different data dimensions are explored in multiple linked views, including scatterplot matrices [6] and parallel coordinates [45, 51]. Another common technique is to use dimension reducing techniques like multi-dimensional scaling (MDS) or self organizing maps (SOA). Such techniques projects the multi-dimensional (parameter) space into the low dimensional visual space and preserve the (global) distances between the objects.

Stacked displays: Stacked display techniques visualize data by using a hierarchical relationship [59]. In case of multi-dimensional data, the dimensions are partitioned. Each dimension of the data is shown in another level of the hierarchy. A common example for such techniques are treemaps [3]. Several dimensions are displayed using a hierarchical representation. Additional attributes of the data points are mapped to some attributes of the nested rectangles (e.g., color, size, etc.).

Icon-based displays: Icon-based techniques map attribute values of multi-dimensional data to some visual features of icons [59]. Typical features are color and shape. Common examples are the Chernoff face, needle icons, star glyphs, and stick-figure icons. These representations are very effective, if the icons relate to the characteristics of the data [59].

Dense-pixel displays: The basic idea of dense-pixel displays is to map each dimension value to a colored pixel [59]. Additionally, the pixels are grouped into adjacent areas. The ordering of the pixels is very important. Dense-pixel techniques provide different arrangements, depending on the purpose. The pixels are arranged to find correlations, dependencies, and hot spots [59].

In the next sections, we discuss the used techniques in more details. In Section 3.2 we discuss small multiples in more detail because we also use this technique to visualize members of a cluster. In the Section 3.3, the scatterplot visualization is discussed in detail, and in Section 3.4 we present parallel coordinates.

3.2 Small Multiples

Small multiples are a visualization concept introduce by Edward Tuttle [95]. He describes them as: „Illustrations of postage-stamp size are indexed by category or a label, sequenced over time like the frames of a movie, or ordered by a quantitative variable not used in the single image itself“ [95].

The idea of small multiples is to use the same basic item (a 1D/2D/3D visualization) multiple times to display an object. Typically, a bar chart, histogram, or function plot is used as the basic item. An overview of commonly used standard 1D/2D/3D visualization techniques is given in Figure 3.2. A bar chart is a representation of 1D data. The size of the bar relates to the corresponding 1D value. For an nD data, n bar charts are created. A histogram is a representation of the distribution of the data. We separated the data into m-intervals. Each interval is called a bin. Then, each bin is incremented for each data point, which is within the interval of the corresponding bin. Each bin is represented with a bar, and the height of the bar corresponds to the number of data points which are within the bin. For an nD data n, histograms are used. This representation is reasonable, if the distribution of the objects across a dimension is in focus. A data point is shown by highlighting each bin across each dimension, which contains the selected date. In other visualization scenarios also graphic are used (for example, landmarks). The multiples show multi-dimensional data without trying to use a too complex objects.

Small multiples are primarily used to compare objects across all dimensions of the data. The user looks after patterns (like correlations, dependencies, hot spots) in the 1D/2D/3D visualiza-

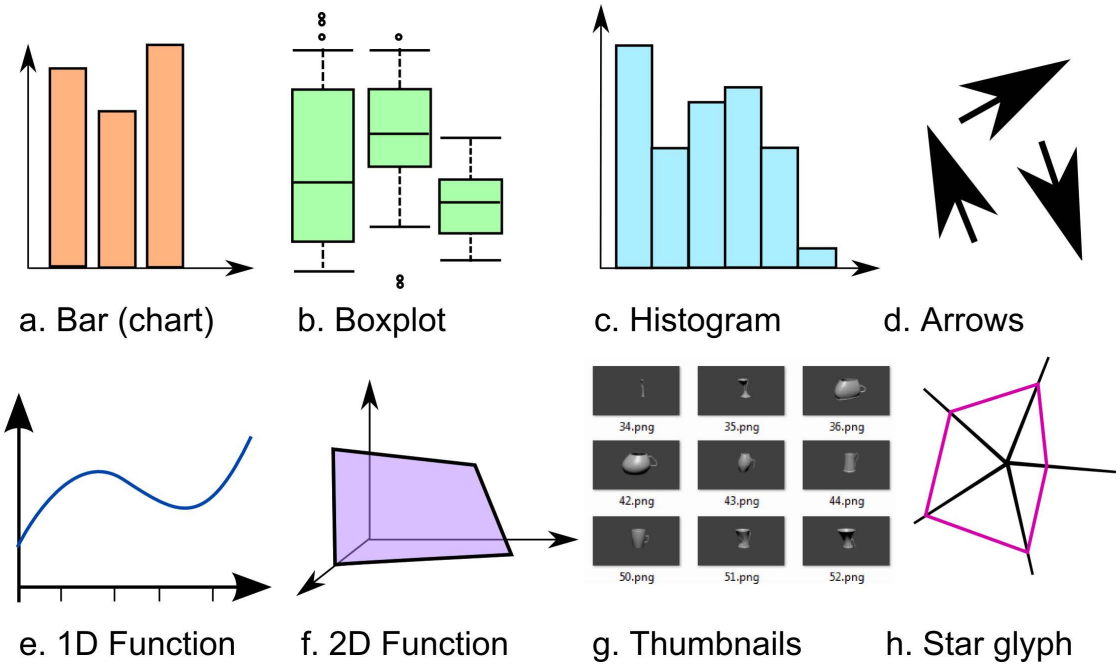

Figure 3.2: Examples of basic visualizations, which are commonly used for small multiples.

tions. The visual mapping has to share the same measures, scales, size, and shape to support the user by comparison of different elements. A change of one attribute at an axis have to affect all plots. Figure 3.3 gives an example of small multiples. Figure 3.3a displays a time-series of a dynamic graph with small multiples. The user can analyze the changes of the graph by comparing it among different timesteps [2]. Figure 3.3b displays the velocity of a tornado. Using many tiny arrows enables to identify the center of the tornado.

The advantage of small multiples is the easy readability. The user only needs to understand a single visualization. Then, the user can apply this knowledge to the other visualizations. To ensure a good readability, the ordering of the basic items should follow a logical ordering. Furthermore, the used items should be simple because the user needs to be able to process many items and to preserve visual clutter.

3.3 Scatterplots

A scatterplot is a diagram using Cartesian coordinates to display values of 2D data. A scatterplot is used to plot data points on a horizontal and vertical axis in the attempt to show how much one variable is affected by another one. Each datum is represented by a marker depending on its values on the X- and Y-axis. The marker is typically a point, but also other representations, like thumbnails, can be used. Figure 3.4 shows an example of a scatterplot matrix.

The scatterplot can suggest various kinds of correlations between these two variables. Typical types of correlations are clusters and linear structures. If a linear structure exists, the data

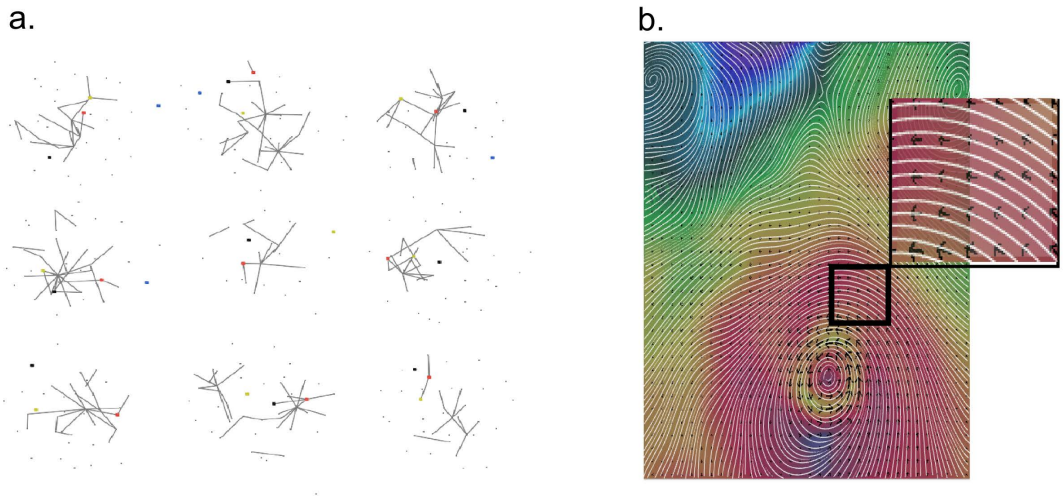

Figure 3.3: (a) Small multiples to display a time-series of a dynamic graph [2]. (b) Small multiples of arrows are used to display the velocity of a tornado [9].

points are close to form a straight line in the scatterplot. In these cases, the two variables have a high linear correlation. Scatterplot can also show a nonlinear correlation. For instance, if the data points are close to form a function $f(x) = x^2$ a quadratic correlation can be identified. The correlation can be positive or negative. A positive correlation is a correlation, where the data points are located from the lower left for the upper right. If the data points are located from the upper left to the lower right, a negative correlation exists. A scatterplot can also be used to identify clusters. Clusters are regions, where the data points are concentrated. The main advantage of a scatterplot is the easy use, and correlations in the data can be identified easily. Furthermore, in a lot of other sciences and applications scatterplot matrices are a basic tool for analyzing 2D data. The main disadvantage of scatterplots is the limitation of showing only two variables.

However, in this thesis multi-dimensional data have to be analyzed. A scatterplot matrix addresses the problem of only displaying 2D data. A scatterplot matrix is a series of scatterplots displaying each combination of variables. At each of the axes one of the variables $x_1, .., x_n$ are displayed. So, all combinations of the variables are shown in a matrix assembly of scatterplots. The data points at the diagonal elements of the scatterplot matrix are always diagonally positioned, because in this case the same variable is assigned to the horizontal as well as the vertical axis. This representation follows the idea of small multiples [95].

Small multiples are great for analyzing many variables. To ensure a good readability, the arrangement of the charts follow a logical ordering. In the scatterplot matrix, the ordering follows a matrix layout. This layout ensures that the user can analyze the data set by comparing one attribute on one axis to all other attributes on the other axes. The ordering of the attributes should also be modifiable. This feature is important to enable the user to add some meta information. Another application area of small multiples is the visualization of a set of objects.

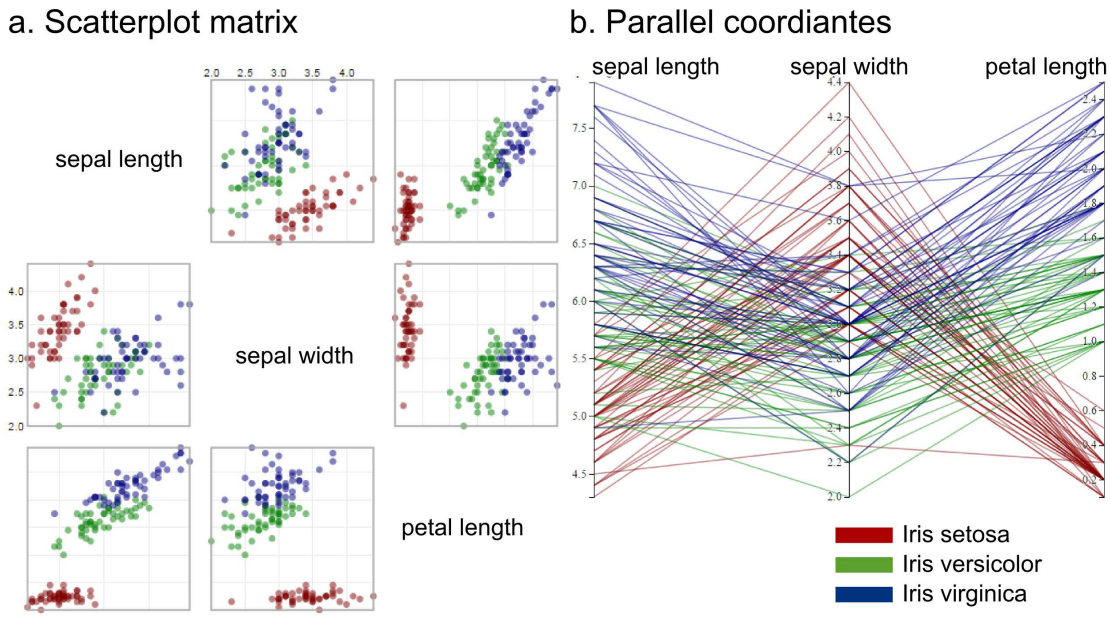

Figure 3.4: (a) Visualization of the multi-dimensional Iris data set with a scatterplot matrix [18] and (b) parallel coordinates [19].

3.4 Parallel Coordinates and Starplots

Parallel coordinates consist of a set of parallel lines, which each represents a dimension of the data. These lines are typically drawn vertically and equally spaced. Each line is also called an axis. A datum in the n-dimensional space is represented as a polyline with a vertex at each parallel axis. The position on each axis corresponds to the value in the associated dimension. Figure 3.4b shows an example of parallel coordinates.

Typically, the parallel coordinates plot is used to find patterns, which identify the relationship between the dimensions of the data. If two dimensions are positively correlated, the lines between two axes are parallel. If the correlation is negative, all lines are crossing at a point between both axes. Clusters can be identified, if a set of lines starts with an ϵ-region and ends in another ϵ-region of the other axis. Outliers can be identified if the line path of the outlier differs from the other line paths of the data.

The ordering of the axes is important. Typically, the user analyzes several orderings of the axes and applies some meta information to find a good ordering. In some research heuristics are presented to find a good ordering of the coordinates [13].

The axes are not parallelly oriented, but oriented within a circle. The lowest value of each axis is positioned at the center of the circle (or shifted a little bit outwards) and the highest value is positioned at the radius of the circle. The orientation within a circle is done by using a transformation reaching from Cartesian coordinates to Polar coordinates. To use starplots, the data-set should be at least three-dimensional. Similar to parallel coordinates, starplots enable to display many coordinate axis. Preventing overloading of the visualization, the dimension of the

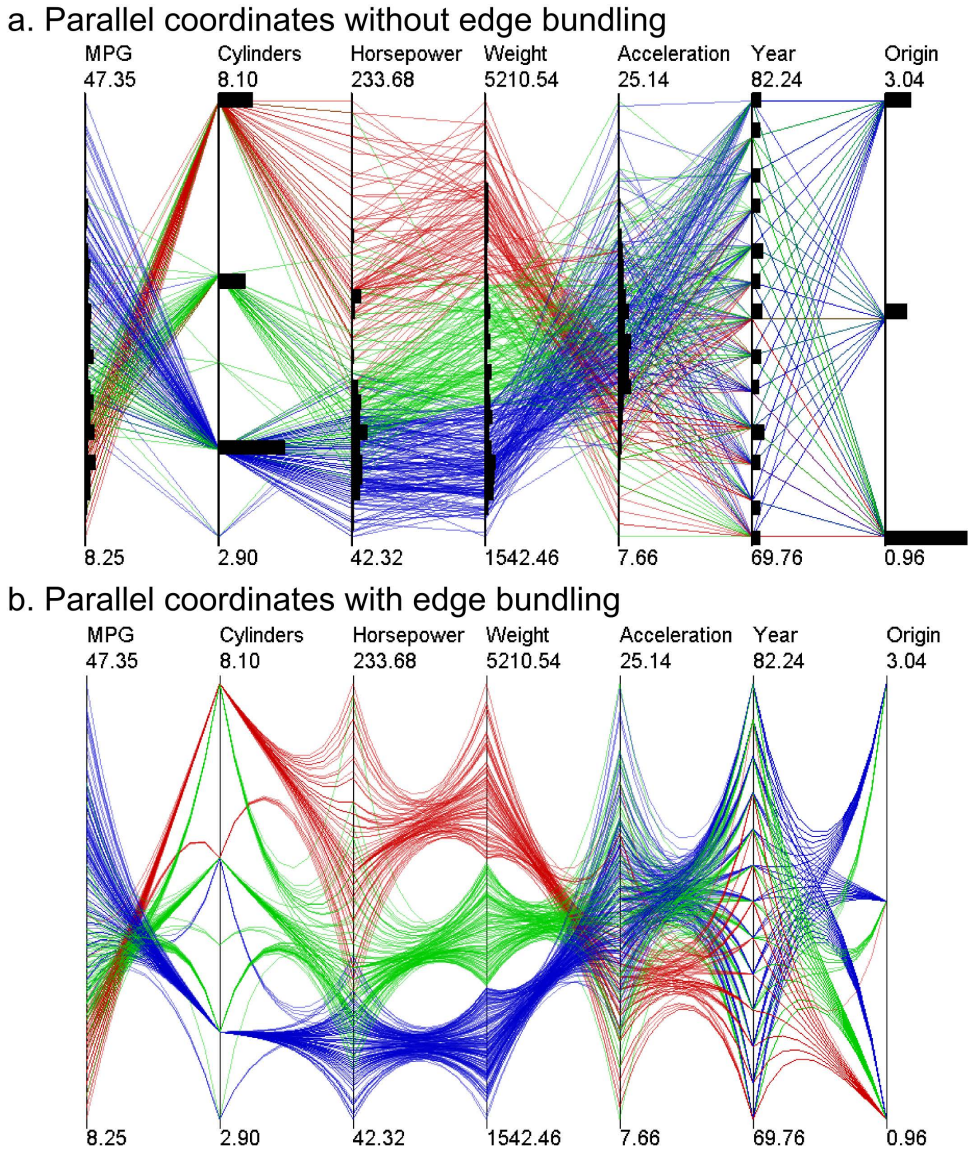

Figure 3.5: Parallel Coordinates without edge bundling (a), and with edge bundling (b) [70]. Edge bundling bundles all line paths which relate to a cluster.

data should be limited to approximatly 10 axes. The optimal number of dimensions is between 5 and 7 axis.

Extending parallel coordinates to support clusters

Visualization of many objects can result in visual clutter. A common way to enhance the readability is to group similar objects. With clustering techniques, objects are grouped into clusters

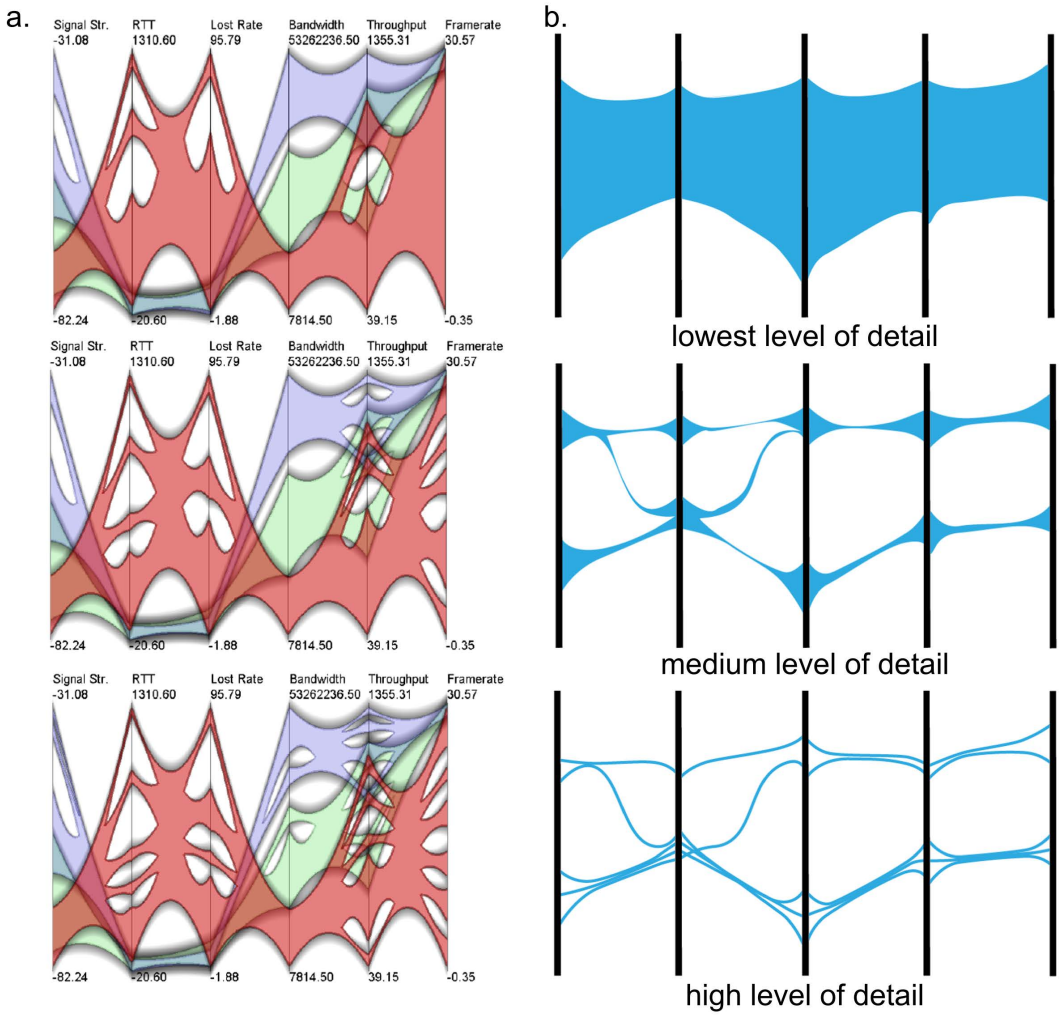

Figure 3.6: Branching technique from McDonnel and Mueller [70]. (a) Different branching widths provide insights into the distribution of the data. (b) Illustration of the highest, medium, and the lowest branching widths. A high branching width results in a polygon between highest and lowest B-spline. A medium branching width results in a polygon between highest and lowest B-spline with holes. A low branching width results in line paths.

using one or several properties (like similarity). Then, all (similar) objects within a cluster are analyzed. In Section 3.8 we present methods to cluster data.

A common technique for visualizing the clusters is to assign each cluster a unique color. Each line path is then colored with the corresponding color. This technique works well for small data sets, but visualizing large data sets can result in visual clutter and over-plotting of clusters. To improve the readability, several edge bundling techniques are presented in the literature. The idea of edge bundling is to bundle all line paths which relate to a cluster. This technique re-

duces visual clutter and decreases the amount of required screen space (compare to Holten [48]). An example is given in Figure 3.5. An overview and a comparison of different edge bundling techniques are done by Heinrich and Weiskopf [45].

McDonnell and Mueller [70] provide an illustrative approach for visualizing clusters. Instead of visualizing a set of line paths for each cluster, they use the minimal polygon which covers all line paths of the cluster. Their technique enables to define the level of detail of a cluster interactively. At the lowest level of detail, a polygon which is determined by the lowest and highest line path is shown. At higher levels of details, the polygon gets holes at regions where the distance between two line paths exceeds a (user-defined) threshold (see Figure 3.6a). The highest level of detail only shows line paths. Figure 3.6b gives an example of a low, medium, and high level of detail.

A lot of different clustering algorithms exist. One common technique is the use of hierarchical clustering. The user can analyze clusters at different levels. Fua et al. [38] extend parallel coordinates for supporting hierarchical clustering. The user interactively can drill-down into a tree structure. Lex et al. [67] proposes a focus+context visualization for comparing separately clustered groups of variables of biomolecular data. Clustered records are connected across multiple groups of variables using bundled curves and ribbons.

3.5 Visualization of Hierarchical Data

Data with an inherent relation are an important class of data. If there is an inherent relation among the data elements, then the data can be represented as a graph. A graph is a pair $G =$

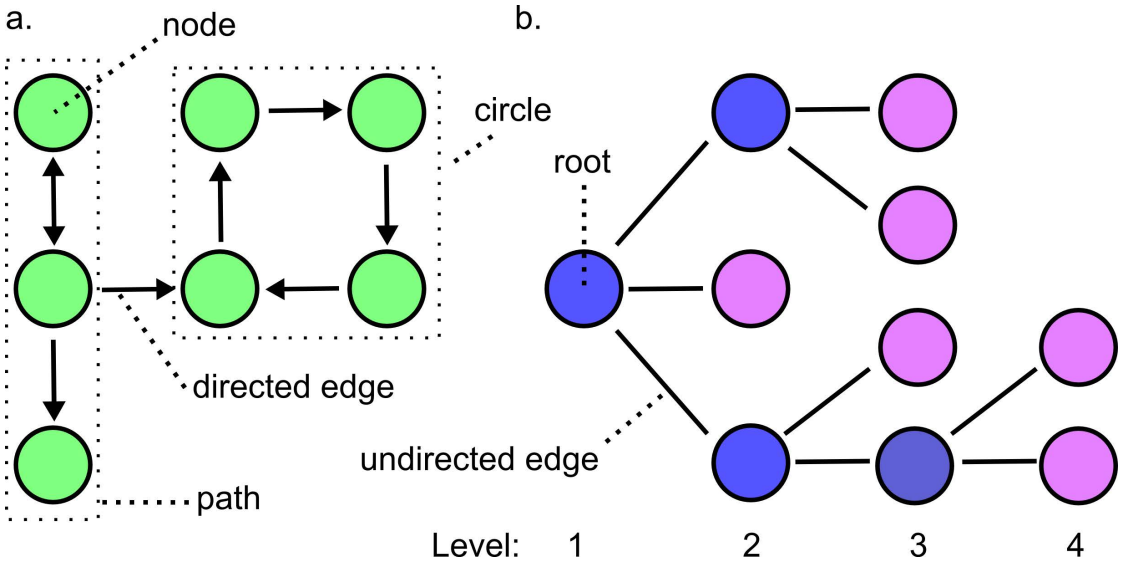

Figure 3.7: a. Directed graphs: green nodes shows a path, orange nodes depicts a path b: Undirected tree: yellow node labels a root node, blue nodes represents parents nodes, and nodes, colored with magenta, shows leaf nodes

(V, E), where

- V: Set of nodes
- E: Set of edges connecting nodes

A *node* represents a datum, and an *edge* is a connection between two nodes (see Figure 3.7) [100]. In particular, the literature differentiates between directed and undirected graphs. If the direction of the edge has to be taken into account, the graph is called *directed graph* (see Figure 3.7a), otherwise it is called *undirected graphs* (see Figure 3.7b) [100]. A permutation of edges is called a *path* (see Figure 3.7a) [100]. *Cycles* are closed path, i.e., a permutation of edges, where the start and end node of the permutation are equal (see Figure 3.7a) [100]. A graph that has no cycles is called *tree* (see Figure 3.7b) [100]. If a node of the tree is singled out to a special node, then the tree is called *rooted* [100]. The special node in the tree is called the *root* (see Figure 3.7b) [100]. A rooted tree has an orientation away from the root [100]. The *parent* of a node n is the node that connects the node n to the root on a path (see blue nodes in Figure 3.7b) [100]. Every node is called *child*, if it has a parent node. A *leaf* is a node, that is not parent of another node (see magenta nodes in Figure 3.7b). [100]. *Internal nodes* are nodes that have at least one child. Trees are often treated as hierarchies, where the length of a path to the root denotes the *level* of a node in the hierarchy [100].

The visual representation of trees is a difficult task, especially, for large trees. The main task of graph-drawing is to ensure high readability and an efficient use of space. Tree drawing algorithms include several tree layout algorithms [47, 85]. Various examples of tree drawing algorithms are given in the Figures 3.8 and 3.9. In the following, we present different graph-drawing algorithms to visualize hierarchical data.

Node-Link Diagrams

A popular approach for displaying trees are node-link diagrams. In Figure 3.8a,b,c common examples for node-link diagrams are depicted. The nodes of the tree are drawn using circles, and edges are drawn using simple lines. The standard layout (see Figure 3.8a) is drawn using a simple recursive algorithm from top-to-bottom (starts at the root node and traverses the tree down to all leaves). This algorithm is not space efficient. Reingold and Tilford [80] present an improved version. Their layout algorithm runs bottom-up and merges the sub-trees. Then, in an additional top-down pass the final node positions are determined. This version is more space efficient. Walker generalizes the algorithm for all general trees [52], and Buchheim et al. [22] shows that the algorithm can be implemented in linear time $O(n)$.

A radial tree is a variation of the node-link diagram. It displays the tree similar to the standard representation, but with an additional transformation using polar coordinates. The depth of the tree is encoded in the radius, and the ordering is encoded using the angle [47]. An example of a radial tree is shown in Figure 3.8c. Another approach to provide a more space efficient representation are balloon trees (see Figure 3.8b). This layout places the children circularly around the parent [47]. Rendering a balloon tree is not easy, especially, if the number of child nodes varies much (i.e., one child node has more than 30 children, and another child nodes has only five or less children). In such cases overdrawing or an inefficient use of space can

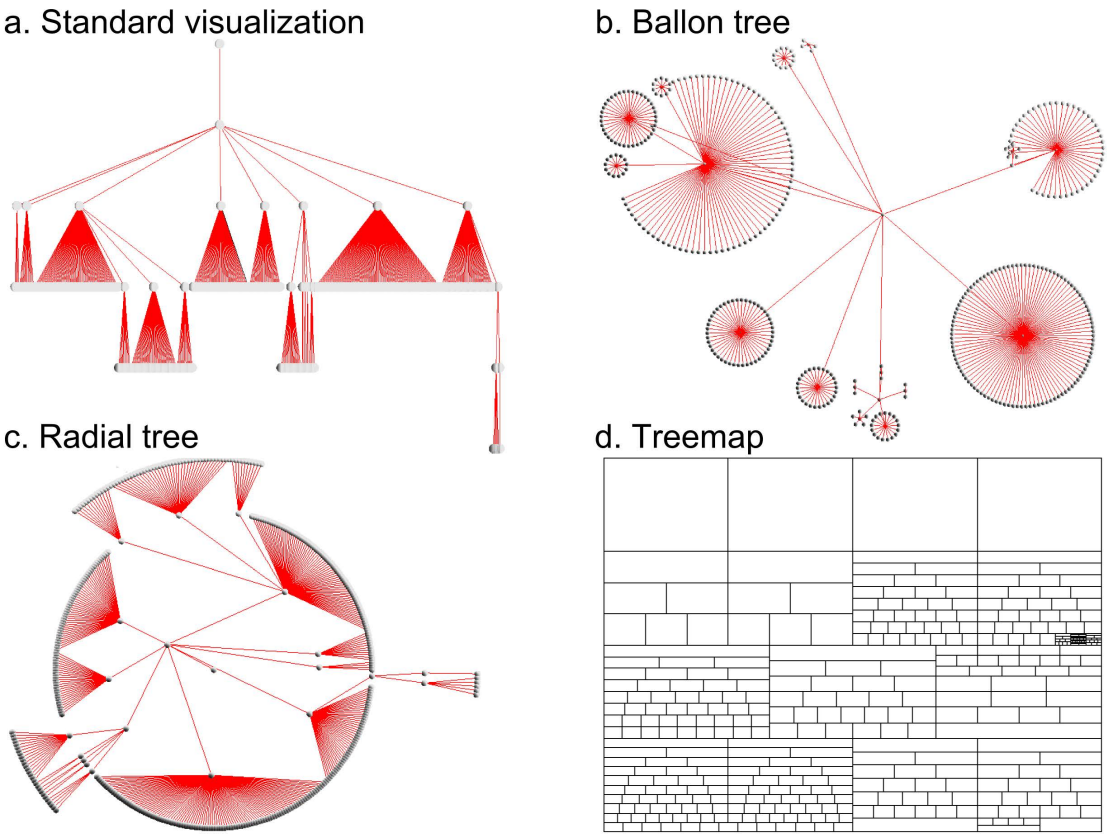

Figure 3.8: Different tree representations [10]: (a) standard visualization, (b) ballon tree, (c) radial tree, (d) treemap (the rectangle size depends on the hierarchy level)

occur [25]. To prevent these problems, different solutions are available. For example, Carriere and Kazman [25] propose a solution which works bottom-up and approximates the circles with a polygon.

Finding a space-efficient representation of large trees in a 2D space is a difficult task. Several approaches extend the output space, and render the tree in 3D space. A common example is the cone tree [47]. This layout is similar to the balloon tree, but the hierarchy is encoded in the height (or depth). Figure 3.10 depicts an example of a cone tree. As shown in this representation, the 3D layout is hard to read. Compare it with the shadow, which results in a balloon tree. Furthermore, navigating in 3D space is more difficult, than navigating in 2D space. So, extending the output space is not recommended.

Layered Diagrams

Layered diagrams are introduced to enhance the use of the available space. These diagrams use layering, adjacency or alignment to represent the tree structure. An example of a layered diagram is an Icicle plot [47]. This method gives a more space-efficient representation and a

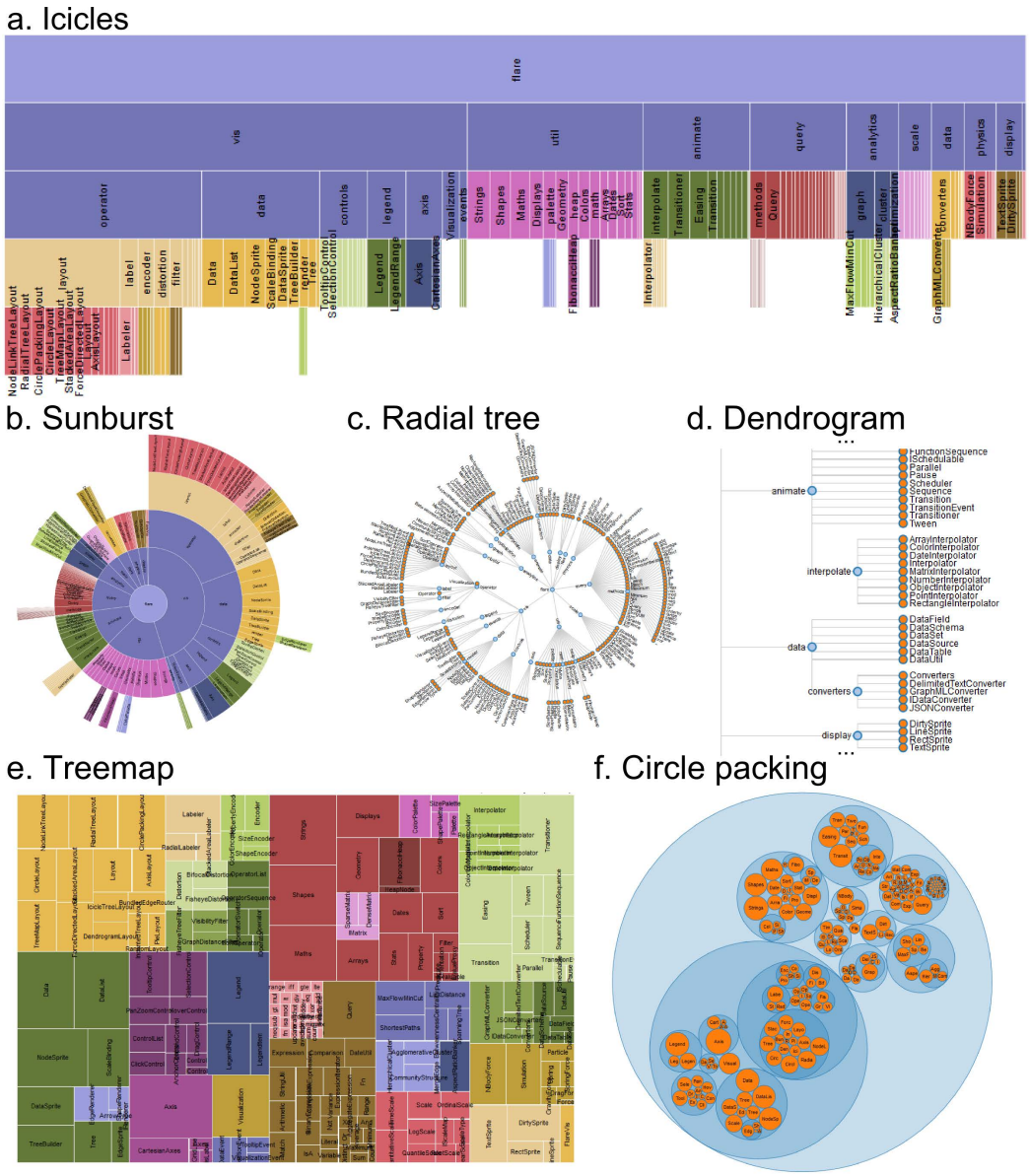

Figure 3.9: Comparison of several tree layouts using Protovis [18]: (a) icicles, (b) sunburst diagram, (c) radial tree, (d) dendrogram, (e) treemap, (f) circle packing

clear impression of the tree structure. Similar to the standard layout, much space is used for the visual mapping of the inner nodes. An additional polar coordinate transformation of the Icicle trees results in a Sunburst diagram [47].

The advantage of layered diagrams is that they are more space efficient than the node-link diagrams. In Figure 3.9a, an example of an icicle plot is depicted, and Figure 3.9b shows a sunburst diagram.

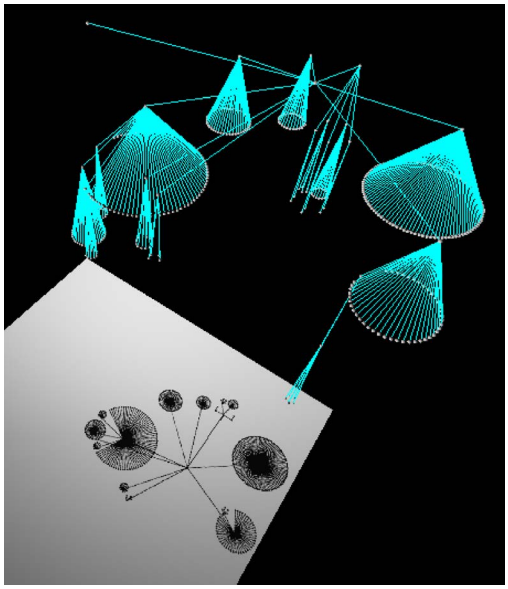

Figure 3.10: Comparision between cone tree (object) and ballon tree (shadow): Using a 3D layout does not enhance the readability [10].

Treemaps

Ben Shneiderman [89] introduces an alternative technique to represent hierarchical data called treemaps. Treemaps use rectangles instead of nodes and encode the hierarchy using a tiling algorithm. The root node is represented by a large rectangle. Then, the rectangle is subdivided into smaller rectangles for every child node. This process terminates if all nodes are subdivided. The size usually correlates to an attribute (e.g., costs, importance, ...). The advantage of treemaps is that they result in a very space-efficient representation of the tree because they use the entire display space.

Various algorithms exist to till the nested rectangles. In Figure 3.11, an overview is given. The easiest algorithm is called Slice and Dice [89] (see Figure 3.11a). The algorithm tills the rectangles in one direction (horizontal or vertical). The size of the rectangles usually correlates to an attribute of the data. The tilling direction is switched between horizontal and vertical at each level of the tree.

Other algorithms for creating a treemap are strips (see Figure 3.11b), squarified (see Figure 3.11c), binary trees, pivot-by-middle (see Figure 3.11d), pivot-by-size (see Figure 3.11e), and pivot by split (see Figure 3.11f) [7, 35]. The performance of the algorithm differs in the aspect ratio of the resulting rectangles, the ordering of the rectangles, and the stability. An aspect ratio of the nested rectangles (the width of the rectangle/the height of the rectangle)) near one ensures better readability. The ordering measures how good the neighborhood of the tree is achieved. The stability measures the changes in the treemap, if a new node is inserted. Ben Bederson and Martin Wattenberg [7] compare the BinaryTree, Ordered, SliceAndDice, Squarified, and Strip algorithm after this measurement. The results are shown in Table 3.1. All algorithms

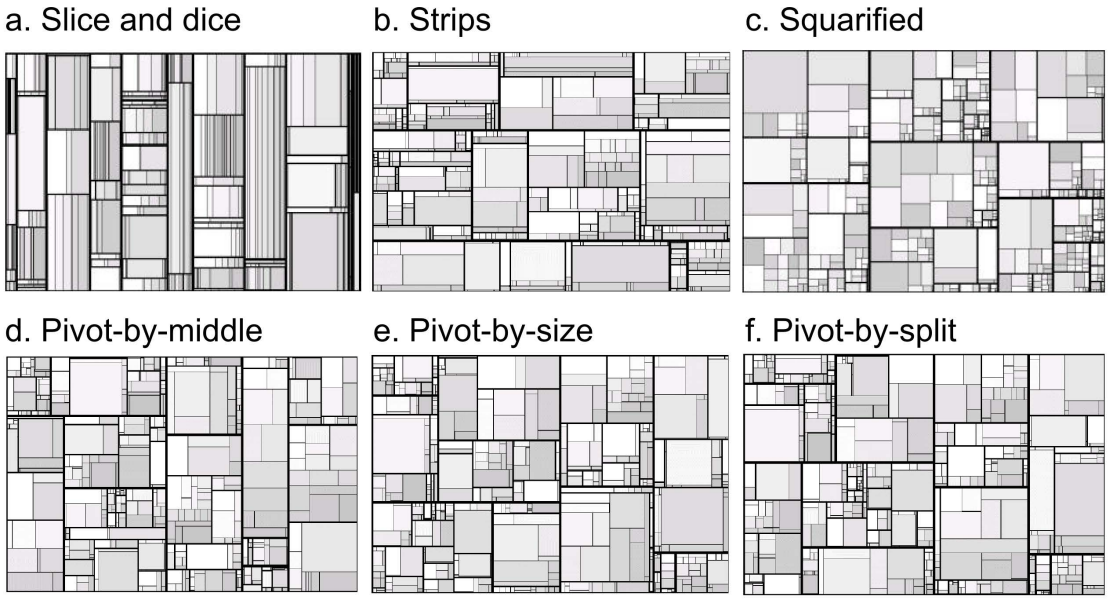

Figure 3.11: Different treemap tilling algorithms displaying a stock portfolio data set [7]: (a) slice and dice, (b) strips, (c) squarified, (d) pivot-by-middle, (e) pivot-by-size, (f) pivot-by-split.

have strengths and weaknesses. For instance, if the ordering is critical, the Strip algorithm is a good choice. However, the commonly used squarified algorithm results in the best aspect ratios.

A common issue of treemaps is that the user cannot identify the borders because the colors of the rectangle give too less contrast. Cushion treemaps solve this problem by shading the rectangles [102]. Another solution for this problem is the use of nested treemaps. This method scales every rectangle down for highlighting the edges. Figure 3.12 shows an example of a Cushion treemap.

The main advantage of treemaps is that they use the display space very efficiently. A disadvantage of treemaps is that they are sometimes difficult to understand. A variation of treemaps is Circular Treemaps, which are introduced by Wenzel [101]. This method uses circles instead of rectangles. As shown in Figure 3.12f, this representation is easier to read but less space-efficient. Another variation is Voronoi Treemaps [5], which use polygons for creating a map. This variation improves the aspect ratio of the map.

Algorithm	**Ordering**	**Aspect ratio**	**Stability**
SliceAndDice	Ordered	very bad aspect ratios	stable
Strip	Ordered	medium aspect ratios	medium stability
BinaryTree	Partially ordered	not very good aspect ratios	stable
Ordered	Partially ordered	medium aspect ratios	medium stability
Squarified	Unordered	best aspect ratios	medium stability

Table 3.1: Comparison of several tilling algorithms [7]

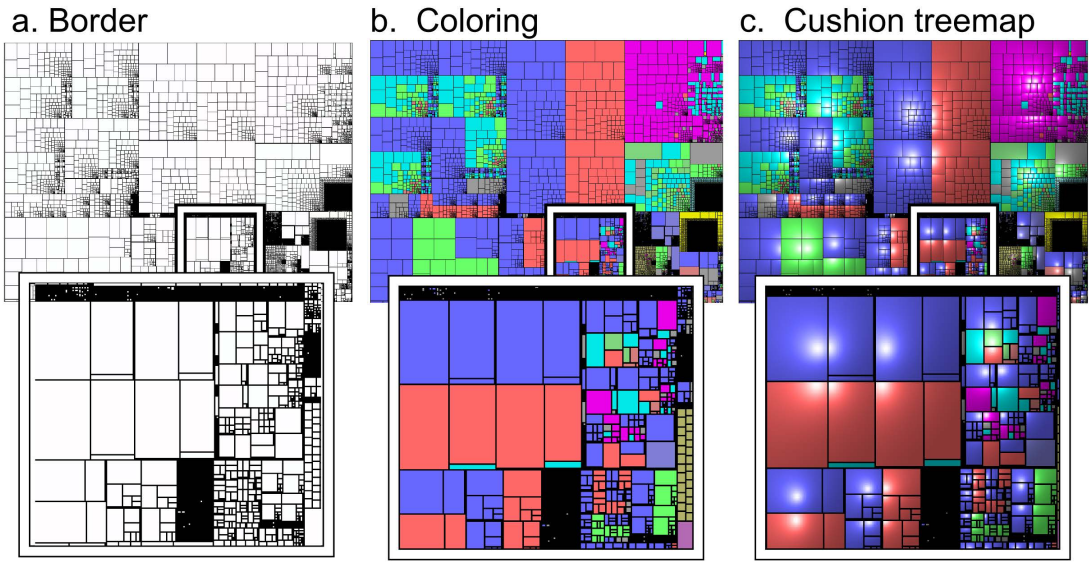

Figure 3.12: Different coloring styles of a treemap created with KDirStat [56]: (a) Border, (b) Coloring, (c) Cushion treemaps

Treemaps are a common visualization technique. Besides the representation of hierarchical data, treemaps can be used to display multi-dimensional data (compare with *Stacked displays* presented in Section 3.1) [3]. Vliegen et al. [98] adapt treemaps to mimic familiar business diagrams, like pie chart diagrams or pyramid. An overview of different applications that use treemaps is given by Shneiderman [88].

Other Techniques

Because of the importance of rendering hierarchical data, many other approaches are presented in the literature. For instance, the Hyperbolic layout [65] generates a tree layout using hyperbolic geometry. Then, the geometry is transformed into the Euclidean plane. This algorithm results in a focus+context representation (compare to Section 3.7 and with Kelly and Ma [53]). Other examples are H-Trees [47], indented layout (often used in file-browsers) [47] and dendrograms (see Figure 3.9d) [47].

3.6 Composite Visualizations

The previous sections discuss different techniques to visualize the abstract multi-dimensional parameter space as well as to represent hierarchical data. However, to analyze a geometry generator, we also need to consider the generated 3D shapes. Our visualizations have to represent the abstract parameter space as well as the generated 3D shapes.

Composite visualization is a common strategy to combine different visualizations in the same geometric space [54]. The idea is to combine different visualization methods to overcome

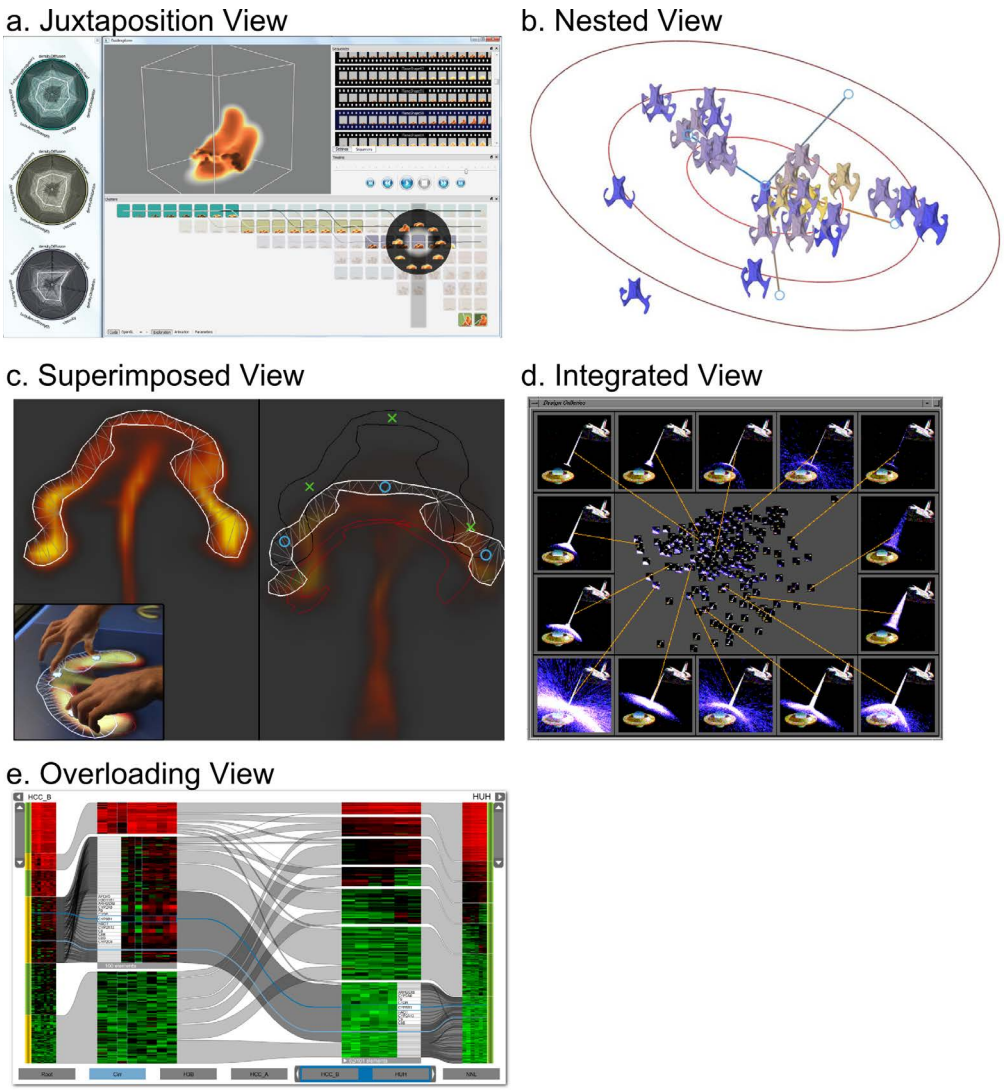

Figure 3.13: Examples of composite visualization: (a) Bruckner and Möller [21] display several visualizations side-by-side. (b) Busking et al. [24] nest volume renderings inside the abstract parameter representation. (c) Coffey et al. [30] show the iso-surface of a volume on the top of volume rendering. (d) Design Galleries [68] use Visual Links. (e) Lex et al. [67] overlay a heat map in parallel coordinates.

the shortcomings of a single technique [59].

Javed and Elmquist [54] give an overview of composite visualizations. They differ in five design patterns:

- **Juxtaposition views:** Showing multiple visualizations side-by-side (compare with small multiples) [54].

- **Integrated views:** Showing multiple visualizations in one view and connecting similar objects with visual links [54].
- **Superimposed views:** Creating a visualization by overlaying several visualizations on top of each other [54].
- **Overloading views:** Rendering a visualization inside another visualization [54]. In contrast to superimposed views, several visualizations are inside the view.
- **Nested views:** A visualization is nested into another visualization. Usually, the marker of the overview representation is replaced by another visualization [54].

In Chapter 2, we present several applications that use composite visualization techniques to combine different visualizations in the same geometric space. The use of *juxtaposition* views is very popular. Busking et al. [24] display a 3D scatterplot, a 3D volume, and a shape comparison view side-by-side. Bruckner and Möller [21] show a 3D simulation, circular parallel coordinate plots, sequences, and a clustering side-by-side (see Figure 3.13a). *Integrated* visualization is used in Design Galleries by Marks et al. [68]. Design Galleries represent the parameter space using multi-dimensional scaling (MDS). Additionally, they show several objects of the parameter space on the side. The objects are connected with the MDS visualization using visual links (see Figure 3.13d). An example of an *overloading* display is given by Lex et al. [67]. They overload the coordinates of the parallel coordinates with a heat map (see Figure 3.13e). Coffey et al. [30] show the iso-surface of a volume on the top of volume rendering. This application is an example of a *superimposed* view (see Figure 3.13c). In Chapter 2, we also show some examples of *nested* views. Busking et al. [24] use 3D shapes as a marker for the 3D scatterplot (see Figure 3.13b). Piringer et al. [77] nest 2D function ensembles in the scatterplot matrix. Talton et al. [93] nest 3D shapes in a semantic map.

3.7 Interaction Techniques

In the previous sections, we present several techniques for visualizing data. However, the goal of visualization is not only to find a good representation of the data (like illustration) but also to provide interaction techniques to work at, and move between, focused and contextual views of a data set [59]. In this section, we present the following interaction techniques in more detail (based on the categorization of Keim [59] and Cockburn et al. [29]):

- **Overview+Detail** techniques use a *spatial separation* between the focused and contextual views [29].
- **Zooming** techniques use a *temporal separation* between the focused and contextual views [29].
- **Focus+context** techniques *integrate* the focus within contextual views [59].
- **Cue-based** techniques *highlight or suppress* objects within the view selectively [29].
- **Interactive Filtering** of data enables to explore large data sets [59].
- **Linking and Brushing** technique enables to interact between different visualizations [59].

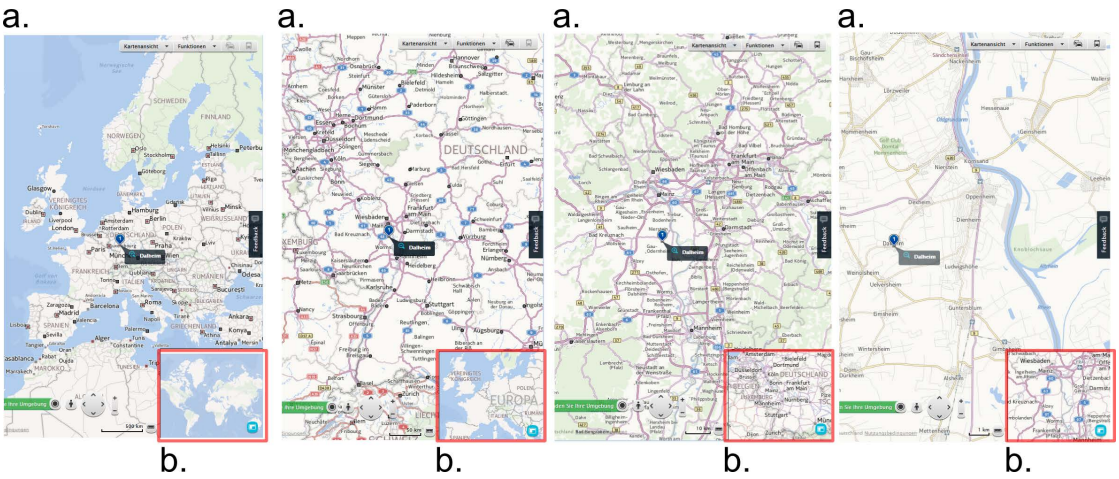

Figure 3.14: (a) An example of *zooming* [46]: First, only an overview is provided. The zoom-in enables the user to display the data in more detail. (b) Additionally, an *overview+detail* representation is provided on the bottom right.

Overview+Detail

Overview+detail is a commonly used technique that visualizes an overview and a detail representation side-by-side. The idea of this technique is that the user can explore all details in a detail view. An additional view shows fewer details (overview depiction). The overview prevents the user from getting lost in all details.

This technique is used in a lot of different applications. For instance, such representations are used in racing video games. The user sees the driver's perspective on the entire display. Additionally, the track is shown at the top of the screen. The user can see his position on the race course. Without this technique, the user has to memorize the whole race track.

Furthermore, standard desktop environments also use the overview+detail technique. Image viewers show the images of a directory using a list of small thumbnails while a select image is shown in all details. A similar approach is used in presentation software systems to display the slides of a presentation (Microsoft PowerPoint, Libre Office Presentation, Google Documents,...).

Zooming

Zooming is a well-known technique to explore details without overloading the visualization. In contrast to overview+detail, it uses a temporal separation between the overview and detail representation. Keim [59] describes zooming as changing the resolution of the displayed data by the user. First, an overview representation is shown. Then, the user can interactively manipulate the view to explore the details of the data [59]. Zooming not only means to display the data objects larger, but also, to represent the data in more detail [59]. To ensure a good user experience, the mental map of the user has to be preserved. Too many changes in a short time can confuse the user. To preserve the mental map of the user, the changes are performed step-by-step.

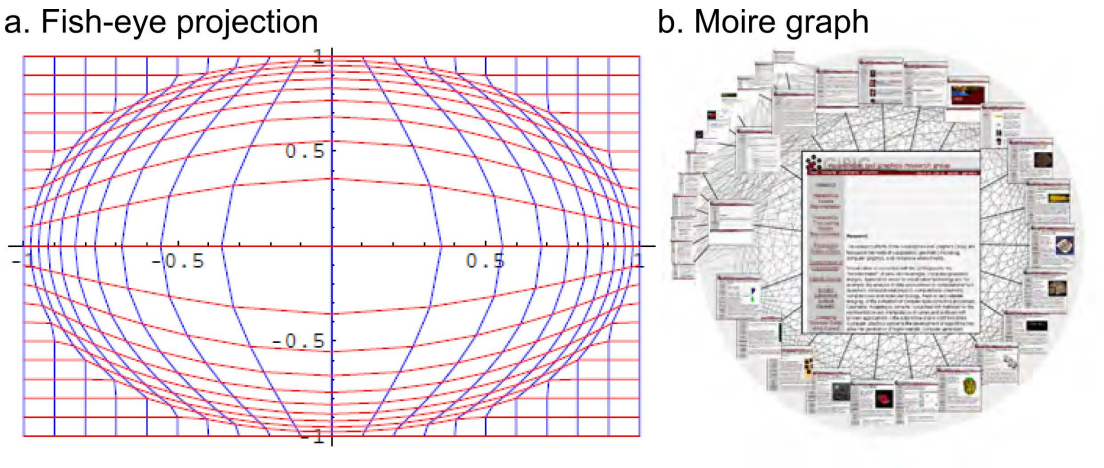

Figure 3.15: Focus+context: (a) a fish-eye projection [47] (b) A moire graph provides a focus+context visualization of a radial tree [53]

A common example of zooming is used by web mapping services, like Google Maps, Bing Maps, Nokia Here, and OpenStreetMap. In the beginning, only a basic representation showing seas and countries is shown. If the user zooms-in, more information is shown, and cities and motorways are displayed. In the most detailed representation, houses, trails, and sights are visualized. An example is shown in Figure 3.14a.

Other examples of zooming are presented in the state-of-the-art (see Chapter 2). For example, Talton et al. [93] use zooming for their semantic map representation. Piringer et al. [77] use zooming to show different levels of 2D function ensembles. Low level of detail results in a visualization of the maxima and minima of the 2D function. The intermediate values of the 2D function ensembles are only shown in a high level of detail.

Focus+Context

A disadvantage of overview+detail techniques is that the context can be lost, if only some parts of the visualization are shown. In contrast to overview+detail, focus+context techniques visualize an overview and a detailed representation in a single view (instead of side-by-side). The basic idea is to visualize the focused parts with a high level of detail while contextual parts are shown with a lower level of detail. The visual seam between the higher and lower level parts is minimized [59]. Combining the focused and contextual view in a single visualization prevent the user from losing the contextual information.

Common examples for focus+context are distortion techniques. For example, hyperbolic and spherical distortions are often used. In Section 3.5 we present a Hyperbolic tree, which provides a focus+context visualization of a tree [65]. Figure 3.15 shows an example of the spherical distortions. More information on distortion techniques is presented by Leung and Apperley [66] and Keim [59]

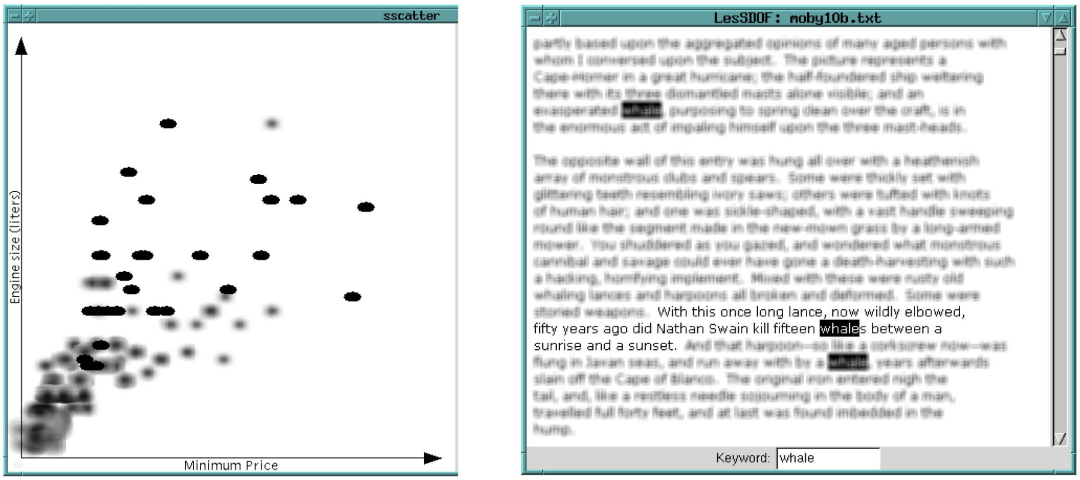

Figure 3.16: Two examples of cue-based techniques [64].

Cue-based Techniques

Cue-based techniques are used to highlight or suppress items of the visualization [29]. The basic idea is similar to the well-known photography technique. If the viewer should focus on an object in the image, the focus of the camera moves to the object, and the camera's depth-of-field is reduced. The resulting image shows the object in sharp focus, while the other objects are blurred [29].

According to Cockburn et al. [29], cue-based techniques are used to highlight objects in the focus or to simplify contextual objects. Kosara et al. [64] use the idea of depth-of-field of a camera to simplify contextual objects (see Figure 3.16). Another approach is to render objects of the background with less opacity or saturation. For instance, this method is used by Berger et al. [11].

Filtering of Data

Exploring a large set of data is a difficult problem because human recognition is limited. To handle large data sets, the user usually splits the data into segments [59]. Then, the segments of interest are focused. A popular technique for exploring data is direct selection or querying [59]. However, direct selection can be difficult if the data set is too large. Querying can result in undesired results [59]. To solve this issue, filtering of data techniques are introduced. The user fades out uninteresting segments with filtering.

Magic Lenses [14] are an excellent example for a filtering of data technique. Magic Lenses provide a magnifying glass. The data under the glass is filtered directly in the visualization. The filtered data is displayed differently than the remaining data set [59]. The data outside the magic lens remains unaffected. An example of the Magic Lenses is shown in Figure 3.17.

Another method for filtering data is the use of transfer functions. This technique is commonly used in volume rendering to define a mapping between the voxels of a volume data and a

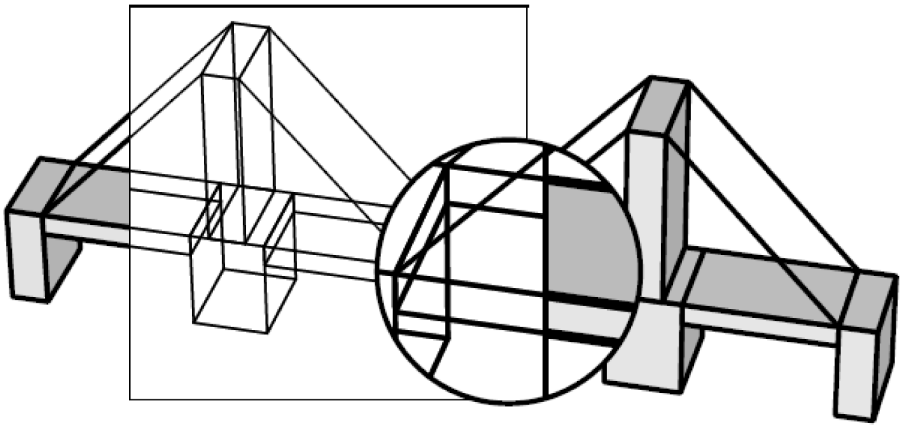

Figure 3.17: MagicLens is shown as an example of filtering techniques [14]

resulting color and opacity in an image [62]. To fade out parts of the volume (e.g., skin, bones), the opacity of the transfer function is set to 0. These techniques enable the fade-out of irrelevant parts of the data set.

Linking and Brushing

As presented in Section 3.6, composite visualization enables the user to combine different visualizations to overcome the drawbacks of a single visualization. In particular, *juxtaposition* views are commonly used to show different visualizations side-by-side. However, the information, which is gathered in one visualization, has to map to the other visualizations to ensure a good user experience. To navigate between different visualizations, linking and brushing is an excellent technique. The user can select items of interest with a brushing tool. The brushed items are then also highlighted in the other visualizations. This method enables the user to navigate and explore the data across different visualizations [59].

Berger et al. [11] present an interesting example of linking and brushing. They study the abstract parameter space of a simulation. The user can select simulation runs with brushing in the input or target space. Then, the simulation runs are highlighted in both visualizations. This technique enables the user to study the sensitivity of a region in the input as well as in the target space. Figure 3.18 shows an example.

3.8 Clustering and Similarity Measurement

The visualization of large data is a difficult problem. The human recognition of objects is limited. Displaying many shapes, images, or other data within one visualization will result in a too complex representation and will confuse the user.

A common approach, to handle large data, is to draw a very simple representation of the objects. In Section 3.1, we present dense-pixel displays, which show each object by only one colored pixel. The dense-pixel display is sorted according to a specified criterion. Only the

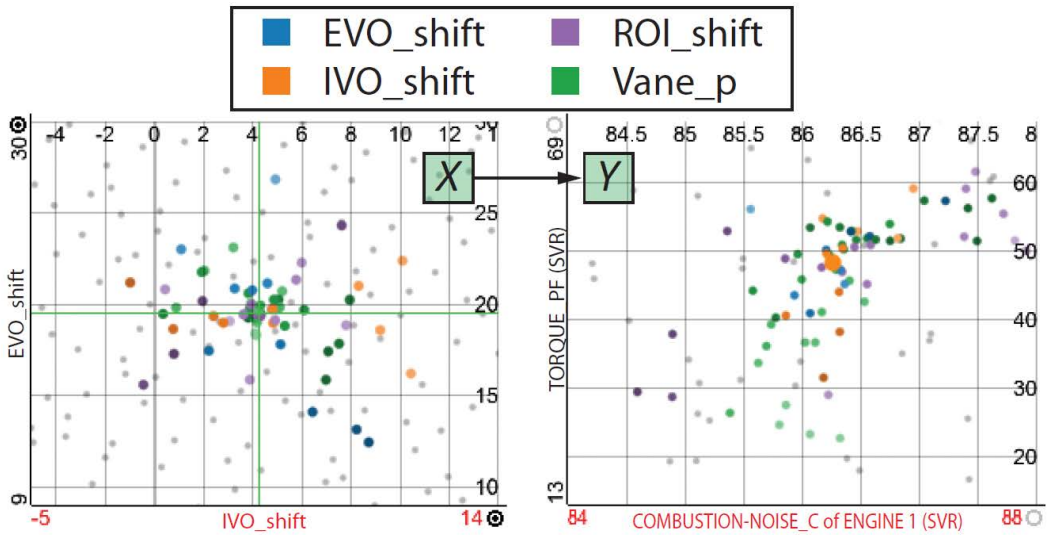

Figure 3.18: An example of linking and brushing [11].

combination of the simple representation and sorting ensures a good readability. The user looks than for patterns (correlation, hotspots, ...). Small multiples also use a similar idea.

Another popular approach is to draw only some representatives. Marks et al. [68] only display some examples (selected randomly) of the parameter space in detail. Another approach is to group similar objects according to a feature (or multiple features) into a cluster. Depending on the application area, the feature can be a similarity measure, or a parameter (combination of parameters, respectively). Then, the objects are grouped using a clustering algorithm (e.g., k-Means, DBSCAN, ...). This approach is used by Bruckner and Möller [21] and by Talton et al. [93].

In the next sections, we present the clustering approach in detail. First, we present several algorithms to cluster a set of objects. A popular approach is to group the 3D shapes according to the similarity. We present different techniques to measure the similarity of shapes in the last section.

Clustering

Clustering techniques group similar objects. Such methods are successfully used in various areas like pattern recognition, image processing, and data mining. In the literature, several algorithms to cluster objects are presented. For example, hierarchical clustering algorithms [55], k-means [12], expectation-maximization (EM) [32], and the DBSCAN algorithm [36] are common techniques to cluster data.

The k-means clustering approach partitions n objects into k clusters. Each object belongs to the cluster with the nearest mean. This problem is NP-hard, but a lot of efficient heuristic algorithms are available [12].

A common algorithm for this approach is Hartigan's method [12]. In an initial step, the user has to specify the number of clusters. If the number of clusters is unknown, several automatic algorithms are available to determine the numbers of clusters [41]. Moreover, the algorithm assigns each object to a cluster randomly. After the initial step, the algorithm calculates the mean of each cluster. Then, the algorithm calculates for each object the distance to each cluster. Each object is then assigned to the cluster with the smallest distance. The algorithm repeats this process until the result converges.

Another heuristic to partition n objects into k clusters are Gaussian mixture models trained with the expectation-maximization algorithm (EM). The algorithm finds the parameters of a statistical model by maximizing the likelihood [12]. First, the EM algorithm expects the log-likelihood using the current estimation of the parameters. Then, the parameters are modified to maximize the log-likelihood. The computed parameters are used in the next expectation step. This algorithm proceeds until the log-likelihood converges. This algorithm is well suited if the data are normally distributed.

The DBSCAN algorithm is another common clustering algorithm [36]. All objects, which are more similar than a user-defined threshold, are grouped into a cluster. As a result, each object within the cluster is similar to at least one member of the cluster. In contrast to k-means clustering, the number of clusters must not be determined before clustering [12]. The shape of the cluster can be arbitrary (as opposed to EM algorithms), and different similarity distance metrics can be used [12].

Another popular approach are hierarchical clustering algorithms. The basic idea is to build a hierarchy of clusters [55]. The literature differentiates agglomerative and divisive methods for creating the hierarchical structure [12]. Agglomerative methods start with each object in an own cluster. Pairs of clusters are merged and moved up in the hierarchy [12]. In contrast to the agglomerative methods, divisive methods start with a cluster, which groups all objects [12]. Then, the clusters are split. The newly created clusters move down the hierarchy. Moreover, the hierarchical clustering algorithms can differ according to the used metric and the linkage criteria [12]. Hierarchical clustering algorithms can be used with different metrics, like Manhattan, Euclidean, and Mahalanobis distance [12]. The linkage criterion determines the distance between two sets of objects using a function of pair-wise distances. Some commonly used linkage criteria are maximum, minimum, and mean linkage clustering [12]. The maximum linkage clustering groups two sets of objects, so that the distance between an object of *group a* and an object of *group b* is maximized. The minimum distance is used for the minimum linkage clustering. The mean linkage clustering calculates the mean of both sets

Similarity Measurement

A common approach to group a set of 3D shapes is to measure their similarity. The similarity measure is used as input for the clustering algorithm to group similar objects. This section presents some methods to calculate the similarity of 3D shapes.

A very popular technique to determine similarity is the use of feature-based methods [34]. These approaches use a set of features to describe a category of objects. Commonly used features are centroid, size, Euler number and even more. Then, a feature vector is created using a well-suited set of features. Each dimension of the feature vector is a numerical entry of one feature.

Besides the feature vector, a metric has to define. The metric determines the similarity between two feature vectors. A common approach is to weight the features with a scalar value and measure the similarity using the dot product of two feature vectors or using the cosine [34].

Chen et al. [27] use a Lightfield descriptor to measure the similarity. They render the shape from different camera positions and use the commonly used 2D Zernike moment descriptor and Fourier descriptor to measure the similarity. While Chen et al. refer the 2D Zernike moment descriptor, Fisher and Hanrahan [37] use the 3D Zernike descriptors to measure the similarity between two 3D shapes. Additionally, they reduce the set of potentially similar shapes by considering the shape's size. Another approach for comparing 3D shapes using 3D descriptors is presented by Chaudhuri and Koltun [26]. They use a shape diameter function, which captures the local thickness of the shape at a sample point and creates a multi-dimensional histogram as a descriptor [26]. Additionally, they use a pyramid-match kernel to compare shapes at different levels of detail [26]. Using the feature descriptor also enables the comparison of different representations of objects. For instance, Xie et al. [103] and Chen et al. [27] calculate the similarity between 2D sketches and 3D shapes.

The presented works enable us to distinguish between different classes of objects to get a broad classification [72]. Finer-scale variations within a class of shapes is still a challenge [72]. The calculation of the similarity measurement with feature-based methods only depends on the features and the metrics used. In other application areas, the user wants to evaluate finer variations of a category of similar shapes. For instance, the user wants to compare different cars [50] or wants to detect the same object in various poses (e.g., compare human poses [49]). An approach is to find a set of features that works only for a particular class of objects. Finding a suitable combination of features and metrics depends on the application area. It is still a very difficult and time-intensive task because the user needs to analyze the complete data set. This technique is often used in computer vision and image understanding for object recognition.

A particular case of feature-based methods is the use the vertex-to-nearest-vertex distance. Such algorithms are mainly used to compare shapes of the same class. The algorithm searches for each vertex of a mesh the nearest vertex of the other mesh and measures the distance. This distance is weighted, and the sum of the distances determines the similarity. If both meshes are equal, the distance is 0. The advantage of using this approach is that the similarity value depends only on the structure of the mesh, and it can be calculated easily. Bruckner and Möller [21] present a similar technique for 3D volumes. Instead of comparing the distances of the vertices of a 2D shape, they measure the similarity of objects by comparing the voxels of the two objects. A similar technique is used by Funkhouser et al. [39]. They measure the similarity by comparing the distances between two shapes using the dot function. Using voxelization enables the use of a shape descriptor. Instead of measuring the similarity of the complete mesh, they compare the parts of the shape. The parts are segmented automatically. A survey of different automatic shape segmentation techniques is given by Samir [87]. Ovsjanikov et al. [72] measure the Euclidean distances between N pairs of points sampled uniformly over the mesh, and the distribution of distances is convolved with a Gaussian kernel.

In the literature, several works explore a set of morphable objects instead of a set of arbitrary shapes. Morphable objects are shapes where a correspondence exists between the vertices of the shapes. This relation is used to morph one shape into another one using (linear) interpolation.

The relation between the vertices can also be used to determine the similarity between objects using statistical tools to determine the variation. Blanz and Vetter [15] explore this idea in the context of 3D face models. Subsequently, the framework has been extended to analyze shapes of human bodies in consistent poses [1]. Smith et al. [92] use a similar technique to explore the parameter space of registered car models, and Coffey et al. [30] explores simulations. The presented methods are very useful if an accurate correspondence between the vertices is provided. In such cases, it is recommended to use such a technique. In order to avoid restrictions, we prefer other methods.

In conclusion, we do not find a perfect solution for calculating the similarity between two arbitrary objects. All methods have their strengths and weaknesses. Some methods are useful for calculating the similarities between different classes of shapes while other methods are excellent to calculate the similarities of the shapes within a category. Some techniques work with arbitrary 3D shapes while other algorithms only work for shapes with an accurate correspondence between the vertices. So, depending on the application area, different solutions are useful.

However, these issues motivate the use of visualization techniques. If the similarity could be calculated accurately, some tasks can be done automatically. For example, a perfect similarity measurement would enable an automatic categorization of shapes (compare with Task 1 in Section 1.1) or to detect regions with high sensitivity (regions of a parameter where a lot of categories are located; compare with Task 3 in Section 1.1). Instead, our solution provides useful visualizations of the parameter space to get new visual insights.

CHAPTER 4

Overview of *Cupid*

Our main goal is to study the relations between the abstract parameter space of a geometry generator and the resulting 3D shapes. Unlike other approaches that show these two domains side-by-side using linked views (compare to the *juxtaposition view* in Section 3.6), our idea is to combine both into a single *composite visualization* (compare to the *nest view* see Section 3.6). We use parallel coordinates for analyzing the multivariate relationships of the data. However, the naïve approach of representing each parameter combination together with the resulting 3D shape in the parallel coordinates would lead to visual clutter. Therefore, we first apply hierarchical clustering to group similar 3D shapes (compare also to Task 1).

In the visualization, we first show clusters at the highest hierarchy level, where the user can selectively drill-down into sub-clusters. The clusters are shown using *composite parallel coordinates* [70]. We augment this abstract visualization of the parameter space by *icons* that represent the 3D shapes or other properties of the clusters. This allows the user to directly relate regions in the parameter space to the corresponding clusters of similar 3D shapes, and vice versa (compare to Tasks 1 and 3). For understanding the hierarchical structure of the clusters, we adapt a linked *radial tree* to display the same icons representing the individual clusters. Our presented technique for composite parallel coordinates can also be used with other multi-dimensional visualization methods. We adapt a *composite scatterplot matrix* to show that the presented technique is not limited to parallel coordinates only.

As an application example, we explore the results of a cup generator that is used for testing the object recognition capabilities of a domestic robot. The parameter space consists of eleven parameters. Certain parameters define parts of the cup (e.g., handle type, overall silhouette) and other parameters represent global modifiers of the shape (e.g., smoothing or distortion). More details of the cup generator are presented in Section 7.3.

The presented application is called *Cupid*. A conceptional overview of *Cupid* is depicted in Figure 4.1. The user starts with determining the parameters of the geometry generator and a sampling strategy for the parameter space. The parameter space is then sampled in order to generate the 3D shapes. We also derive certain measures that describe quantitative properties of the 3D shapes (e.g., area or convexity), which can be used in the visual exploration. The 3D shapes

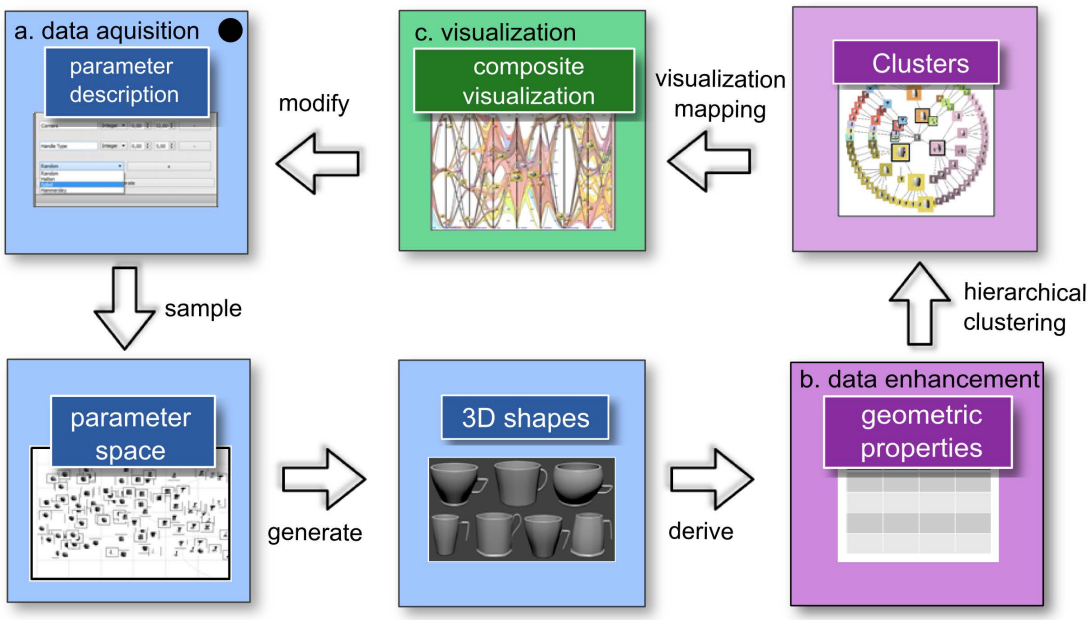

Figure 4.1: Overview of our system for exploring the results of a cup generator: First, the user determines the parameters of the geometry generator and the sampling strategy. The parameter space is then sampled and the corresponding 3D shapes are generated. Measures describing geometric properties of the 3D shapes are derived and the 3D shapes are clustered based on their similarity. Finally, the resulting clusters are visualized in a radial tree and a composite visualization that shows both the abstract parameter space of the geometry generator and the generated 3D shapes.

are hierarchically clustered based on their similarity. The resulting clusters are displayed in the radial tree as well as the parallel coordinates and scatterplot matrix.

4.1 Data Generation and Pre-Processing

Initially, the user defines the parameters of the geometry generator and chooses a sampling algorithm (see Figure 4.1a). Each parameter is described by a name, a type (categorical or continuous), and the range to be sampled. As shown in Table 7.1, the cup generator has attributes such as handle type, convexity, and parameters describing the overall shape (e.g., width of the cup at the top, middle and bottom). Additionally, the user defines the number of samples and selects the sampling method. We provide random sampling and low-discrepancy sampling. The advantage of using low-discrepancy sequences is to achieve a uniform sampling of the entire parameter space as opposed to random sampling [60] (see Figure 4.2).

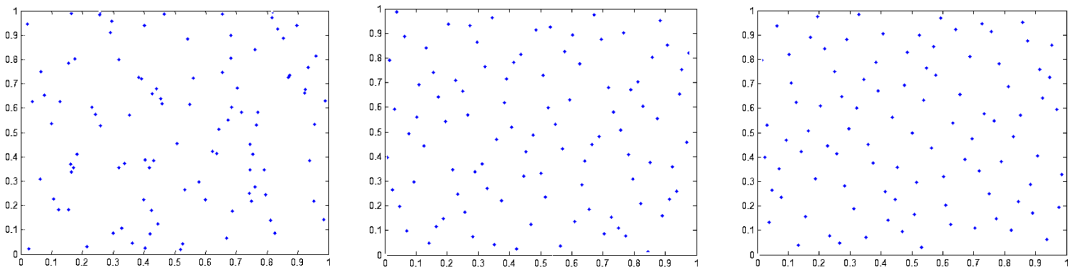

Figure 4.2: Different sampling techniques: Comparison between (a) random sampling and (b,c) low discrepancy sampling. Low discrepancy sampling uniformly distributes the data over the entire sampling space.

4.2 Coregistration and Hierarchical Clustering

We use clustering to group the created 3D shapes according to their shape similarity. This helps to reduce visual clutter in the visualization and supports the user in finding regions in the parameter space that result in similar geometric shapes (compare to Tasks 1 and 3). Since the 3D shapes are generated with different alignment, we have to coregister them first. We then use agglomerative hierarchical clustering to create a hierarchy of clusters, where clusters of similar 3D shapes are merged as one moves up the hierarchy. This has the advantage that initially only a few clusters need to be displayed, and the user can drill-down into selected sub-clusters.

To align the generated 3D shapes, we use the *iterative closest point* (ICP) algorithm which calculates a transformation T (translation and rotation) that minimizes the difference between the vertices of two geometric meshes [105]. There are many approaches for computing mesh similarities. Due to its simplicity, we chose to use the sum of Euclidean distances between each vertex x_i in the mesh M_1 and its nearest neighbor in the mesh M_2, which can be computed rapidly using kD-Trees:

$$d(M_1, M_2) = \frac{1}{|M_1|} \sum_{x_i \in M_1} \min_{x_j \in M_2} |x_i - x_j|$$

If smaller parts of the 3D shapes are not aligned (e.g., the handles of two similar cups have different positions), however, the measure can result in a large difference. Since our cup generator creates an identifier for each part of the cup, we align the individual parts first and compute the similarity based on the aligned parts of the 3D shapes:

$$d^* = \frac{1}{\sum_i w_i} \left(\sum_i w_i \cdot \max(d(\text{part}_{i,1}, \text{part}_{i,2}), d(\text{part}_{i,2}, \text{part}_{i,1})) \right),$$

where $\text{part}_{i,k}$ is the i^{th} part of mesh k and w_i is the corresponding weight. Accordingly, more important parts of the geometry can have higher influence on the similarity than less important ones. If a part of the shape is missing, we add a penalty value to the similarity value. Note

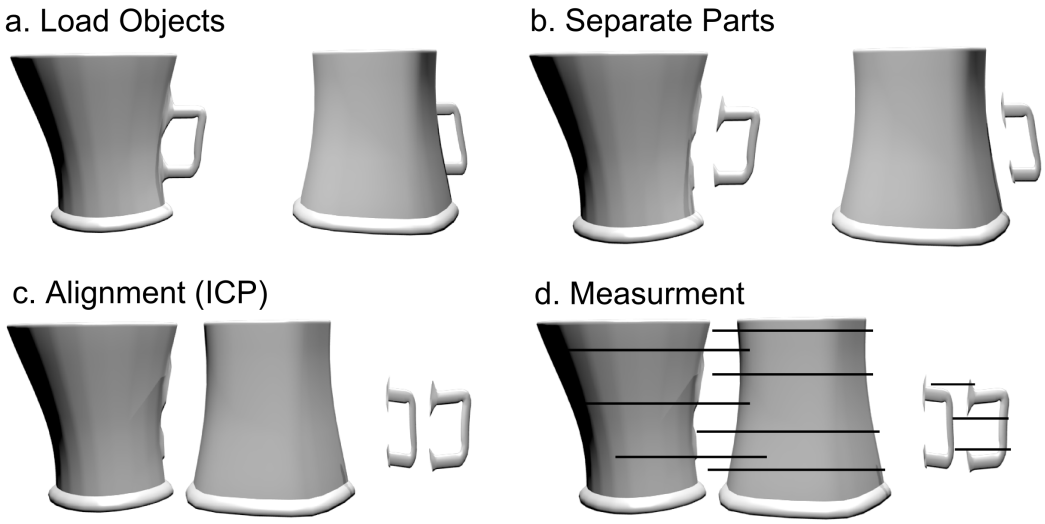

Figure 4.3: Similarity Measurment: (a) The data are loaded. (b) All parts are separated. (c) All parts are rotated to the same orientation. (d) The similarity can be measured.

that the idea of using parts of a 3D shape is not limited to our cup generator. For example, fractal vegetation generators often have symbols and predicates, which identify the parts of the geometry [79].

Our hierarchical clustering is based on DBSCAN [36] which is a popular density-based clustering algorithm (see Section 3.8). Areas with higher density (similarity) than the rest of the data form arbitrarily shaped clusters. Objects in sparse areas that separate the clusters are considered to be noise. Advantages of this algorithm are that it does not require a predefined number of clusters and it can identify arbitrarily shaped clusters of similar 3D geometries. Additionally, it can be easily adapted to create a hierarchy of clusters. The algorithm has two parameters, namely a similarity threshold st and a minimum number of cluster members $minS$ in order to be considered a dense area. Starting from an arbitrary 3D shape created by the geometry generator, the 3D shapes with a similarity d^* below st are queried. If the number of similar 3D shapes exceeds $minS$, the region is considered sufficiently dense and the similar 3D shapes are used as seeds for growing the cluster. 3D shapes located in non-dense regions are classified as noise.

Since the number of clusters resulting from DBSCAN can be very large (depending on the data set and the parameter settings of the algorithm), we extend our approach to hierarchical clustering (compare to Fua et al. [38]). The idea is to group similar clusters to reduce the number of clusters to be displayed. During the visual exploration, the user can then interactively drill-down into selected sub-clusters for a detailed inspection. The initial result of DBSCAN already provides us with the leaf nodes in our hierarchical structure of clusters. In order to merge clusters, we iteratively apply DBSCAN where the similarity threshold st is increased by Δst,the resulting hierarchy is shown in the radial tree in Figure 3.1b. The construction is

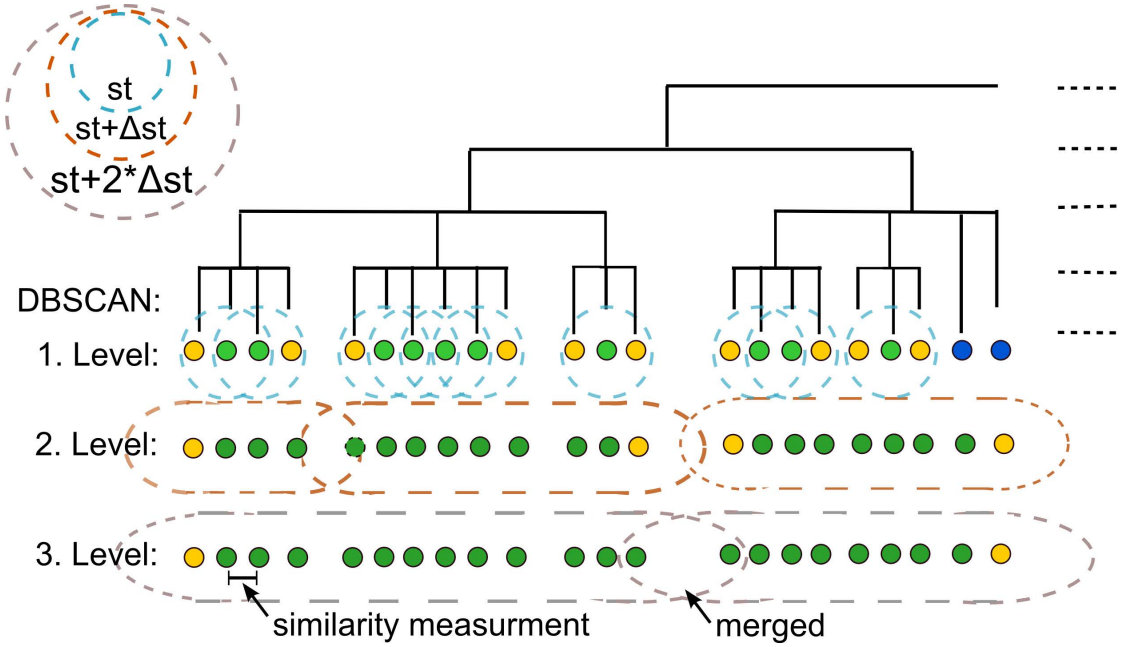

Figure 4.4: Hierarchical Clustering: At first, similar objects are grouped into a cluster using DBSCAN (green: merged, yellow: density-reachable, blue: noise). Incrementation of threshold st results in a hierarchical clustering.

illustrated in Figure 4.4. In order to keep our approach simple and usable, we only display four hierarchy levels of the clustering (the fourth level only displays the root). Since we pre-compute the similarities between each pair of 3D shapes, the user can interactively modify the parameters st, Δst and $minS$, and inspect the resulting hierarchy in the radial tree.

CHAPTER 5

Composite Visualization of Abstract and Spatial Data

After computing the clusters, and geometric properties, the multi-dimensional parameter space, and hierarchical clustering data are visualized. *Cupid* provides a cluster-based composite parallel coordinates approach and a composite scatterplot matrix for visualizing the multi-dimensional parameter space. The hierarchical clustering is visualized using different hierarchical layout algorithms, like radial trees and circular dendrograms.

In the following, we describe the composite visualization of both abstract and spatial data using parallel coordinates and a scatterplot matrix. We then describe our linked hierarchical layouts (radial tree, circular dendrogram, and treemap), and finally the icons, which are the basic items for nesting the spatial information and steering the views. In the last section, some other features are presented, like a simple shape browser.

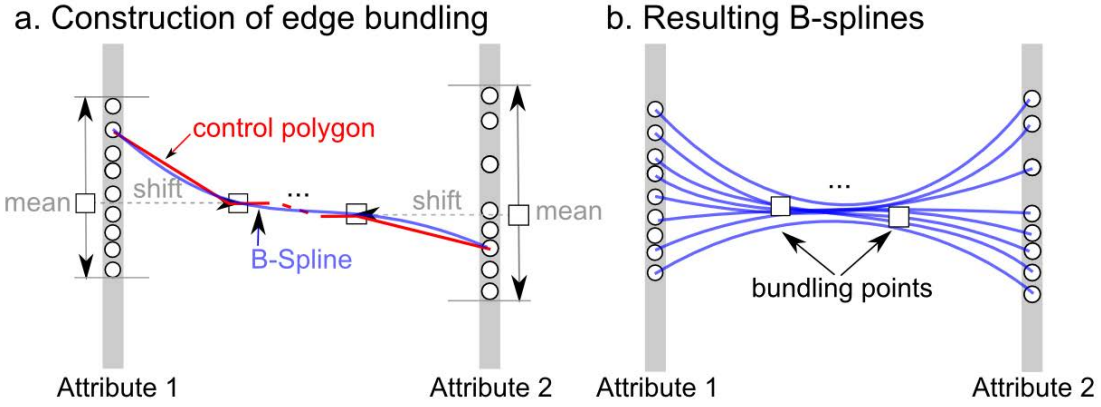

Figure 5.1: (a) Construction of edge bundling (b) Resulting B-splines.

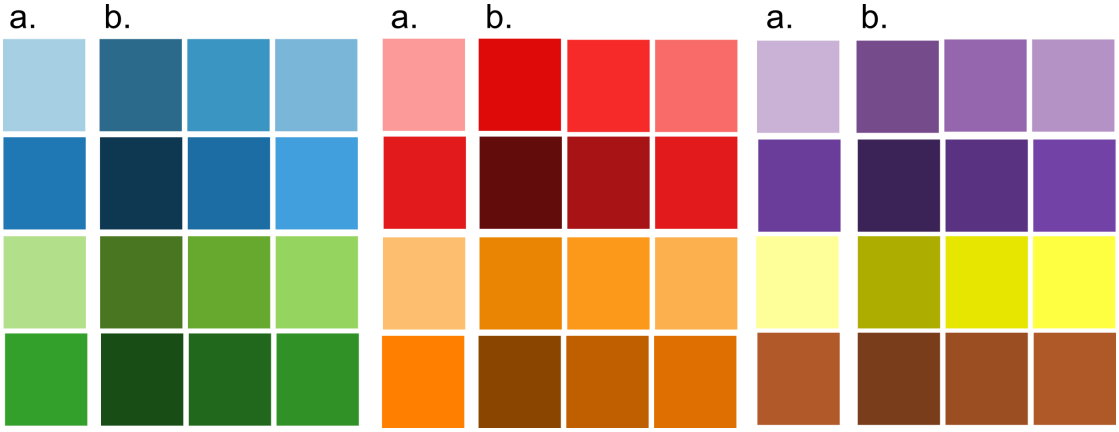

Figure 5.2: (a) Colors used for clusters at the top-level (b) Variation of the luminance used for the second level of the hierarchical clustering

5.1 Composite Parallel Coordinates

For exploring the parameter space of the geometry generator, we use a novel cluster-based composite parallel coordinates approach with nested spatial information (compare to the *nest view* see Section 3.6). The basic items of the visualization are the clusters resulting from the hierarchical clustering, which are represented as nested icons. The icons show the 3D shapes or other properties of the cluster representatives. Additionally, we use illustrative techniques to enhance the visual representation of the clusters.

Illustrative techniques can enhance the readability of parallel coordinates [70]. Each cluster is represented by a colored polygon that shows the extend of the cluster (see Figure 3.1). Rendering just the convex hull of the polygons would not provide much insight into the underlying data. Therefore, we use the branching technique from McDonnell and Mueller [70], which enables the user to control how densely or sparsely the data are distributed in the parallel coordinates (see Figure 3.1a). As presented in Section 3.4 the rendering colored polygons instead of all line paths reduces visual clutter. The user can control the level-of-detail with the branching parameter (compare with focus+context techniques presented in Section 3.7). We apply edge bundling to enhance the visual representation [48, 70]. The start and end positions of the B-spline curves are equal to those of the polylines. Additionally, we add two control points which are used for bundling the curves. The control points are placed at the mean of the corresponding clusters, which are shifted to the region between a pair of coordinates. Figure 5.1 shows the creation of the B-spline curves. The user can control the bundling factor, which reduces overlapping between clusters and decreases the amount of screen space occupied by a cluster. Moreover, individual 3D shapes, which have been classified as noise are represented as B-splines with gray icons.

To differentiate between clusters, the user can select between various styles for a cluster. We offer color-only, contour, shadow and shaded style, according to McDonnell and Mueller [70]. The styles help to discriminate between several clusters. Additionally, the opacity and saturation

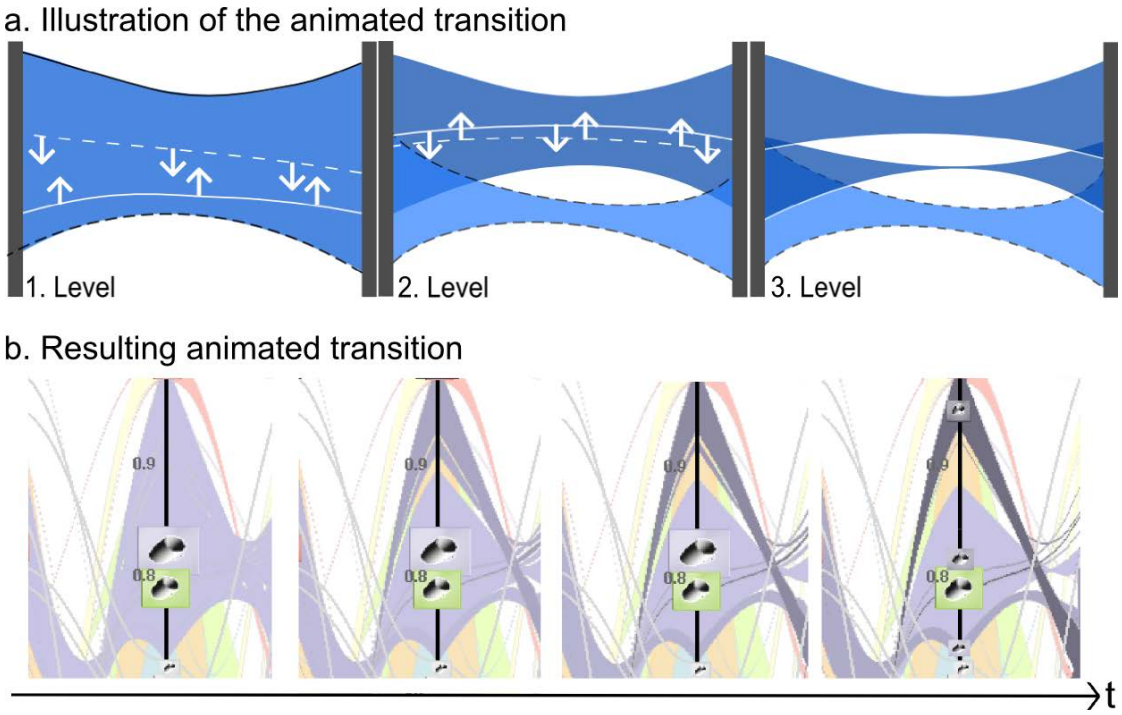

Figure 5.3: Animated transition between two levels of the cluster hierarchy: The geometry and color are morphed from the parent to the child clusters. Coloring with different luminances enables the user to discriminate between child clusters and ensures consistency during interaction.

can be selected. To highlight the selected cluster, a different style, opacity, and saturation can be assigned to the colored polygon (compare with cue-based techniques presented in Section 3.7).

Cupid uses the cluster hierarchy for color assignment. Each cluster at the top level of the hierarchy gets a different color using a qualitative color map from ColorBrewer [44]. In Figure 5.2a, the colors are presented. If the user drills-down into a cluster of interest, the sub-clusters get the same color but with a different luminance. This color assignment ensures consistency during the visual exploration, and the difference in luminance supports the discrimination between different sub-clusters. In Figure 5.2, an overview of the effect using different luminances is given.

In order to preserve the mental map of the user, we perform an animated transition that morphs a selected parent cluster into its sub-clusters (see Figure 5.3). For this task, we create a new polygon, which saves for each vertex the start and the end position of the animated transition. The animated transition is then performed using a simple linear interpolation between the start and the end position.

The first step is to calculate the child polygons. We use the child polygons to morph between the start and end position. Moreover, the polygons determine the end of the animated transition. The next step is to calculate the start of the animated transition. For this task, we search for each parent polygon the corresponding polygons of the child clusters. We need to differentiate

between two cases. If the parent polygon has only one child polygon, both polygons are equal. In the other case, the parent polygon has two or more child polygons. The lower vertices of the parent polygon relate to the lowest vertices of all child polygons. The upper vertices of the parent polygon relate to the highest vertices of all child polygons. This relationship is used to set the start and the end positions of the animated transition. The other vertices of the child polygons result in overlapping areas or holes. We close the holes by calculating the mean. The mean is used as start position (see Figure 5.4a). For overlapping regions, we also use the mean as start position. Additionally, the color is interpolated between the parent and child clusters. We apply this algorithm for each polygon of the parent cluster.

Several examples are shown in Figure 5.4. The left side displays the parent polygon, and the right side shows the resulting child polygons. In Figure 5.4a, the polygon A is split into two child polygons. As shown in this figure, the outer vertices of the child polygon are set to the parent polygon and the inner vertices to the mean. The arrows depict the correlation between the parent and child polygons. In Figure 5.4b, *polygon B* is also divided into two polygons, but *polygon C* does not change. As shown in this figure, the same algorithm as for *polygon A* is done. *Polygon B* is unchanged. In Figure 5.4c, *polygon D* is split into three child polygons, and in Figure 5.4d the *polygons E*, and *F* are split into two child polygons. We use linear interpolation to change the position of the vertices during the animation as illustrated in Figure 5.3a. Some time-steps of the resulting animated transition, are shown in Figure 5.3b.

In addition to the parameter values, each cluster is represented by nested icons that depict the 3D shape or other properties of the cluster (see Section 5.4). The size (area) of an icon represents the corresponding number of cluster members. The user can set a minimum and maximum size for the icons to avoid overlap or too small icons. For each parameter of the generator, the icons are vertically placed at the center of the corresponding clusters. If two icons overlap, they are moved away from each other. We place larger icons first, before placing smaller ones. The icons are used for steering the visualization. Mouse-over an icon highlights the corresponding cluster, which is then drawn in front of all other clusters and shown with a higher opacity (see the purple cluster in Figure 3.1a).

Cupid also provides a *detail window* for comparing 3D shapes within and across clusters. For example, the user can click on the icon of a top-level cluster, which then opens a detail window and shows the 3D shapes of the corresponding sub-clusters (see Figure 3.1). Alternatively, a line brush [63] can be used to directly select currently invisible 3D shapes, where the corresponding B-splines (also not shown) intersect with a simple line segment drawn in the view (e.g., see Figure 5.5). Within the detail window, the selected 3D shapes are first sorted according to their cluster membership at the top level. Then, the selected 3D shapes are sorted according their similarities within the sub-clusters. The color of the clusters at the bottom level is there by used as background color.

Moreover, some basic operations are implemented. The ordering of the parameters can be changed. This feature is very useful to analyze different parameter combinations. Each parameter can also be removed and inserted. If some parameters are not important, the user can remove the parameters. If the user wants to undo the operation, the user can insert the desired parameter. Furthermore, a mirroring tool is implemented. Usually, the lowest parameter value is located at the bottom of the corresponding coordinate axis, and the highest parameter value is

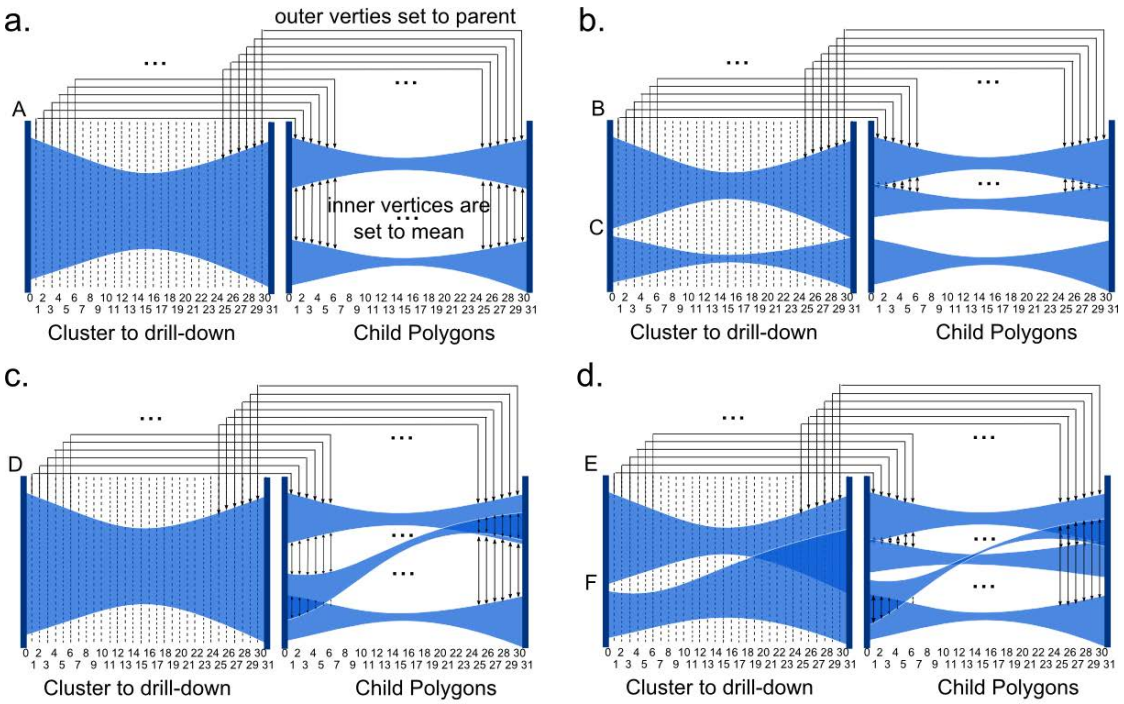

Figure 5.4: Construction of the animated transition: (a) Polygon A is split into two child polygons. (b) Polygon B is also split into two child polygons, but polygon C is unchanged. (c) Polygon D is split into three child polygons. (d) Polygon E and F are split into two polygons each.

located at the top. With mirroring, the parameter is inverted. Then, the highest parameter value is located at the bottom of the corresponding coordinate axis (and the lowest parameter value at the top). The mirroring tool is useful for finding correlations (compare with Harrison et al. [43]) and selecting B-splines with the line brush tool (see Section 8.2).

The composite parallel coordinates visualization is primarily used to analyze the geometry generator's parameters space. This representation can also be used to modify the result of the hierarchical clustering, where the user can manually split and merge clusters. The user can click on two icons and merge them into a new cluster. Clusters can be split as well by marking 3D shapes in the detail window. The selected shapes are grouped into a new cluster. Using the merging and splitting tool enables to group shapes with respect to the parameter values. In Section 8, some workflows using the splitting and merging tool are presented.

5.2 Composite Scatterplot Matrix

Besides the parallel coordinates, we also implement a composite scatterplot matrix to analyze the multi-dimensional parameter space. As presented in Section 3.3, scatterplot matrices are a powerful approach for identifying correlations between variables in multidimensional data.

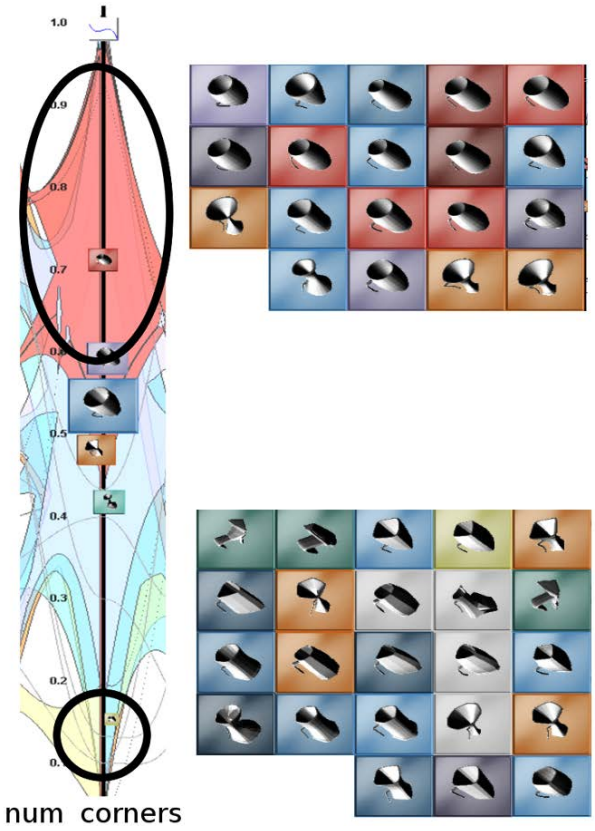

Figure 5.5: Comparison between the clusters within a selected region.

This visualization technique is used in several parameter studies, e.g., in Busking et al. [24] and Piringer et al. [76].

A method, to nest the shapes within the scatterplot matrix, is to draw icons instead of points as markers (compare with *nest view* presented in Section 3.6). Piringer et al. [77] use this approach for nesting 2D function ensembles in the scatterplot matrix. This technique enables to show all available information in one plot. First, we use a similar approach for visualizing the parameter space of a geometry generator. Instead of visualizing 2D functions, we visualize geometric shapes with icons. Similar to Piringer et al. [77], a geometric shape of interest can be highlighted in a detail window. In Figure 5.6a, the resulting scatterplot matrix is shown. As shown in the figure, the shapes are very small. Though zooming is supported by *Cupid*, it is very difficult to analyze the scatterplot matrix and identify similar shapes. So, this approach is not practical for our purpose.

Instead of drawing every shape within the scatterplot matrix, we use a novel composite clusters-based scatterplot matrix. We use an approach similar to the composite parallel coordinates. *Cupid* visualizes each cluster with an icon. As discussed in the previous section, we use hierarchical clustering to group similar 3D shapes. Drawing only one icon for each cluster in the scatterplot would not give much insight on the distribution of the clusters. For example,

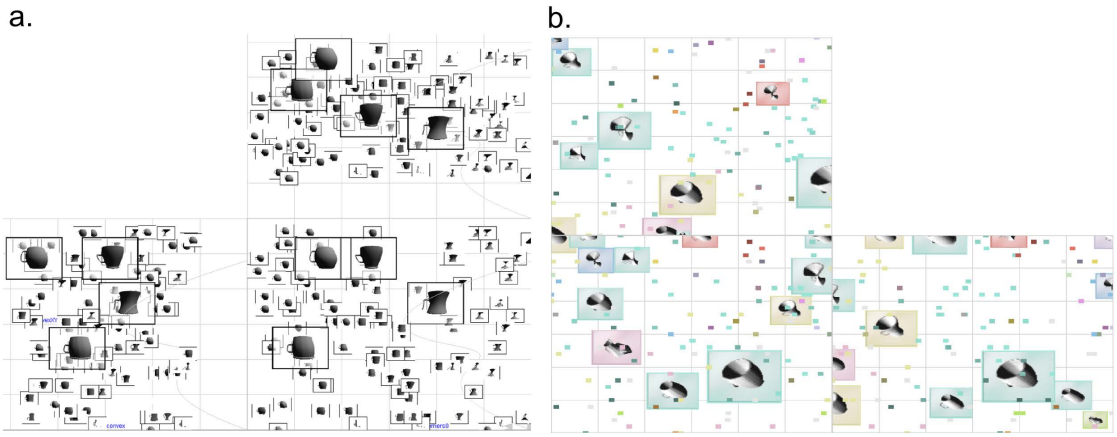

Figure 5.6: Composite scatterplot approach: (a) Method based on Piringer et al. [76], (b) Our composite scatterplot approach

parameter	description	value
k	determines the amount of attraction between the icon and the corresponding cluster members	$k = 0.172$
ϵ_0	determines the amount of repulsion between two icons	0.0005
d	determines the damping of the force at each timestep	$d = 0.1$

Table 5.1: Parameters of the force-based layout

some members at the bottom-left area and top-right area of the scatterplot would result in an icon, which is placed at the center. So, we create an icon for each region, where at least two members of the clusters are placed (equivalent to the branches in parallel coordinates). The algorithm measures the distance between the members of the cluster in each scatterplot using the Euclidean metric. All members of the cluster, which are below a user-defined threshold t, are grouped into one icon. The threshold t defines how sparsely the icons are distributed over the plot. We use this approach for each scatterplot in the scatterplot matrix. Additionally, all members of the cluster are drawn using a marker. The markers are colored with the corresponding cluster's color. Figure 5.6b gives an example of our scatterplot matrix approach.

After calculating the position of all icons, we use a layout algorithm to prevent overlapping icons. Identifying all overlapping icons, and setting them to a new position is not sufficient. The icons should be placed near to the center of the corresponding cluster members. To handle this problem, we adapt a force-based layout algorithm based on Hooke's law. Force-based layouts are popular for drawing graphs. Nodes, which are connected, are attracted, and nodes which are not connected are separated using a repulsive force. We adapt this approach for our scatterplot matrix. Each icon is attracted to the center of the cluster, while simultaneously repulsive forces are used to separate other icons. This algorithm prevents overlapping icons and it ensures that the icons are placed close to the desired position (i.e. close to the center of the cluster). The

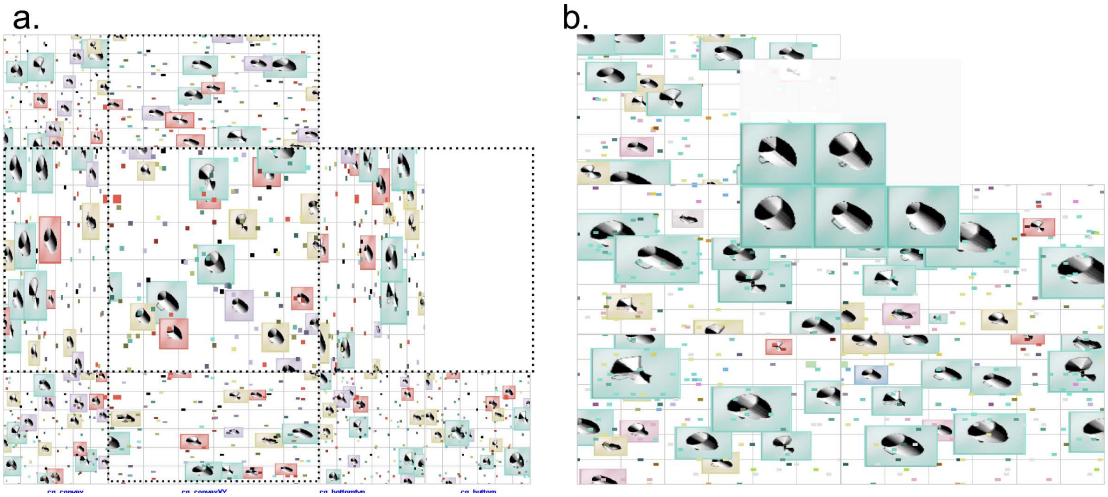

Figure 5.7: (a) Enlarging of rows and columns enables focus+context visualization. (b) Members of a cluster shown within a detail window

attraction force f_H is calculated for every icon using:

$$f_H = \sum_i (s_i - r) * k,$$

where r is the location of the icon, s_i is the location of the corresponding cluster member s_i, and k is the amount of attraction between the icon and the cluster members. The repulsive force is like those of electrically charged particles based on Coulomb's law. The position of the icons is limited by the border of the visualization. We compute the repulsive force f_R for every icon as follows:

$$f_R = (r_1 - r_2) \frac{q_1 * q_2}{4\pi\epsilon_0 \|r_1 - r_2\|^2},$$

where $r_i, i \in \{1, 2\}$ are the location of two icons, $q_i, i \in \{1, 2\}$ are the sizes of the icons, and ϵ_0 is the amount of repulsion. The force is accumulated, and the velocity v is updated using

$$v_{new} = (v_{old} + (f_H + f_R) * t) * d,$$

where t is the timestep, and d the damping factor of the force. The position of the icon p is updated using the Euler method:

$$p_{new} = p_{old} + t * v_{new}.$$

The parameters of the force-based layout are shown in Table 5.1.

To provide a consistent user experience, we use the same GUI elements (icon, detail window, buttons, etc.) and provide similar tools in composite parallel coordinates and scatterplot matrix. The members of each cluster can be highlighted a simple mouse-over of the corresponding icon. Additionally, all other icons, which represent the same cluster, are highlighted across

the whole scatterplot matrix. *Cupid* draws highlighted icons and markers 1.5 times larger. Additionally, *Cupid* provides a lasso tool, which is used instead of the line brush tool in the parallel coordinates. This tool enables the user to define a region of interest, where all markers which are within the region are selected. Analog to display all members of a cluster, a detail window shows all selected shapes.

In a scatterplot matrix, typically all combinations of two parameters are shown. Drawing all parameter combinations leads to visually overloaded representations. Each combination of parameters is represented twice (i.e. $(paramter_1, parameter_2)$ and $(parameter_2, parameter_1)$). The diagonal elements only show a diagonal line (i.e. $(parameter_i, parameter_i)$). While other approaches use this matrix representation, we decide to draw only elements below the diagonal of the scatterplot matrix. So, each combination is only drawn once. The diagonal matrix elements are removed. This representation results in a triangular matrix. However, *Cupid* also enables to remove columns and rows, and to change the ordering. These two operations are important, because in many application scenarios not every parameter is interesting. For example, the parameters defining the handle are not interesting if the body of the cups is interesting. Furthermore, in some application scenarios not every combination of parameters is in the focus of the user. If the user wants to compare two parameters with the others, it is not necessary to draw each combination. Removing unneeded parameter combinations results in a better use of space. For instance, if the user compares one parameter with the others, only one line of the scatterplot matrix is drawn.

Another feature of *Cupid* is to enlarge columns and rows of the scatterplot matrix. This operation follows the idea of focus+context (compare with Section 3.7). Figure 5.7a shows an example of this visualization.

5.3 Visualization of Hierarchical Clustering

We apply hierarchical clustering to reduce visual clutter and to identify similar 3D shapes at multiple hierarchy levels. For exploring and modifying the results of the clustering, we adapt a radial tree which is a popular technique for representing hierarchical data (see Schultz et al. [86] for an overview). The radial tree is a common node-link tree layout with a transformation to polar coordinates (see Figure 5.8a). The advantage of radial transformations is a better usage of space for trees with few hierarchy levels but many leaf nodes. We replace the nodes of the radial tree with icons that represent the clusters (see Section 5.4). While drawing the radial tree, we check for overlapping of icons. If two icons overlap, one icon is shifted. Additionally, we draw circles in the background to discriminate between the different hierarchy levels. The user can compare the 3D shapes of different cluster members and manually split or merge clusters in the visualization.

We also implement other hierarchical layouts like circular dendrograms and treemaps. The user can select the hierarchical representation at run time. Circualar dendrograms are a typical technique for visualizing hierarchical clustering. A circular dendrogram is very similar to a radial tree but intermediate nodes are not drawn to avoid visual clutter (see Figure 5.8b). However, we prefer the radial tree layout because the user can relate the intermediate nodes to the cluster representatives shown in the parallel coordinates, for example, using linking. Treemaps

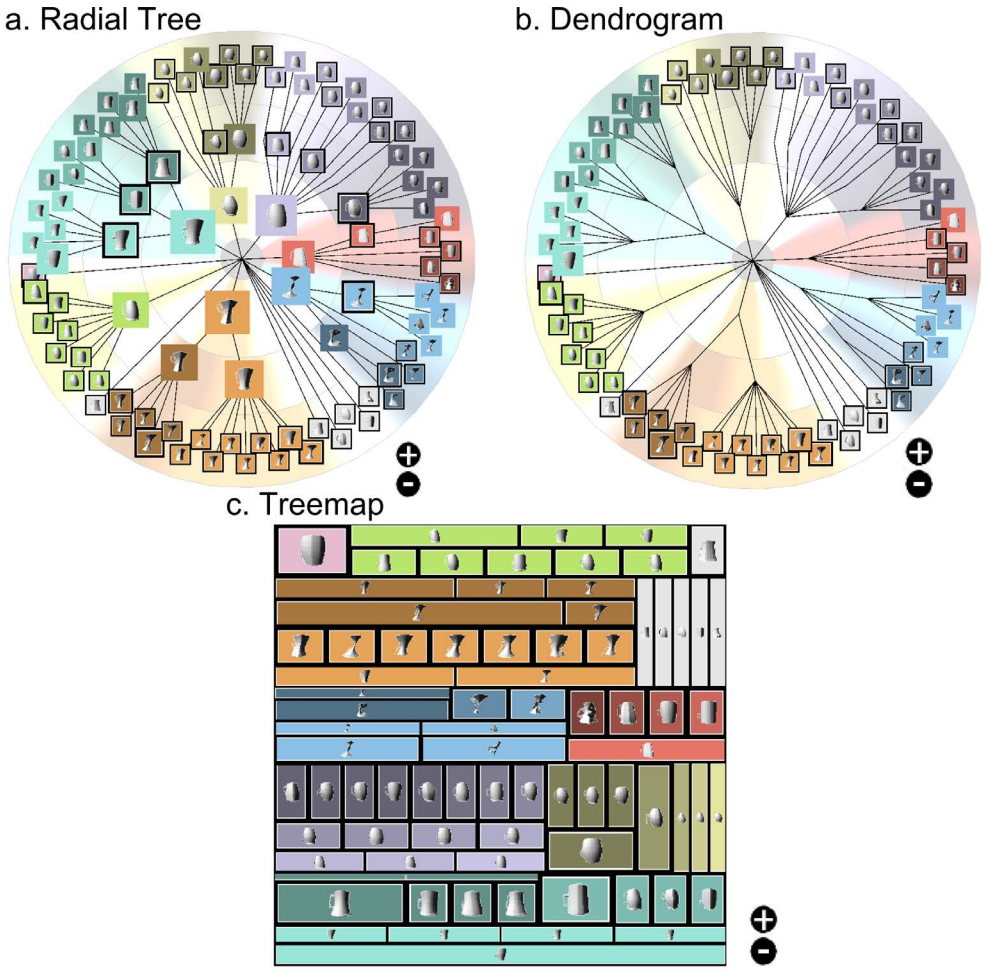

Figure 5.8: Different representations of the cluster hierarchy: (a) Radial tree, (b) Circular dendrogram, (c) Treemap.

are another popular technique for representing hierarchical data. We implement a treemap using the strip tilling algorithm [7] which ensures good readability and preserves the ordering of the nodes. We replace the nested rectangles with icons (see Figure 5.8c). The advantage of a treemap is its compact layout, but the structure of the hierarchy is more difficult to read. In the evaluation, our domain experts rated the radial tree as the preferred choice for *Cupid*.

The hierarchical layout can also be used to modify the result of the hierarchical clustering, where the user can manually split and merge clusters. In Figure 5.9a, for example, the user clicks on the icons of two similar clusters (highlighted in red) and merges them by pressing the minus button. The result is shown in Figure 5.9b. Clusters can be split as well by marking different 3D shapes in the radial tree, for example, using a detail window (see Figure 5.9c) and by pressing the plus button. Since the ordering within a cluster does not provide information,

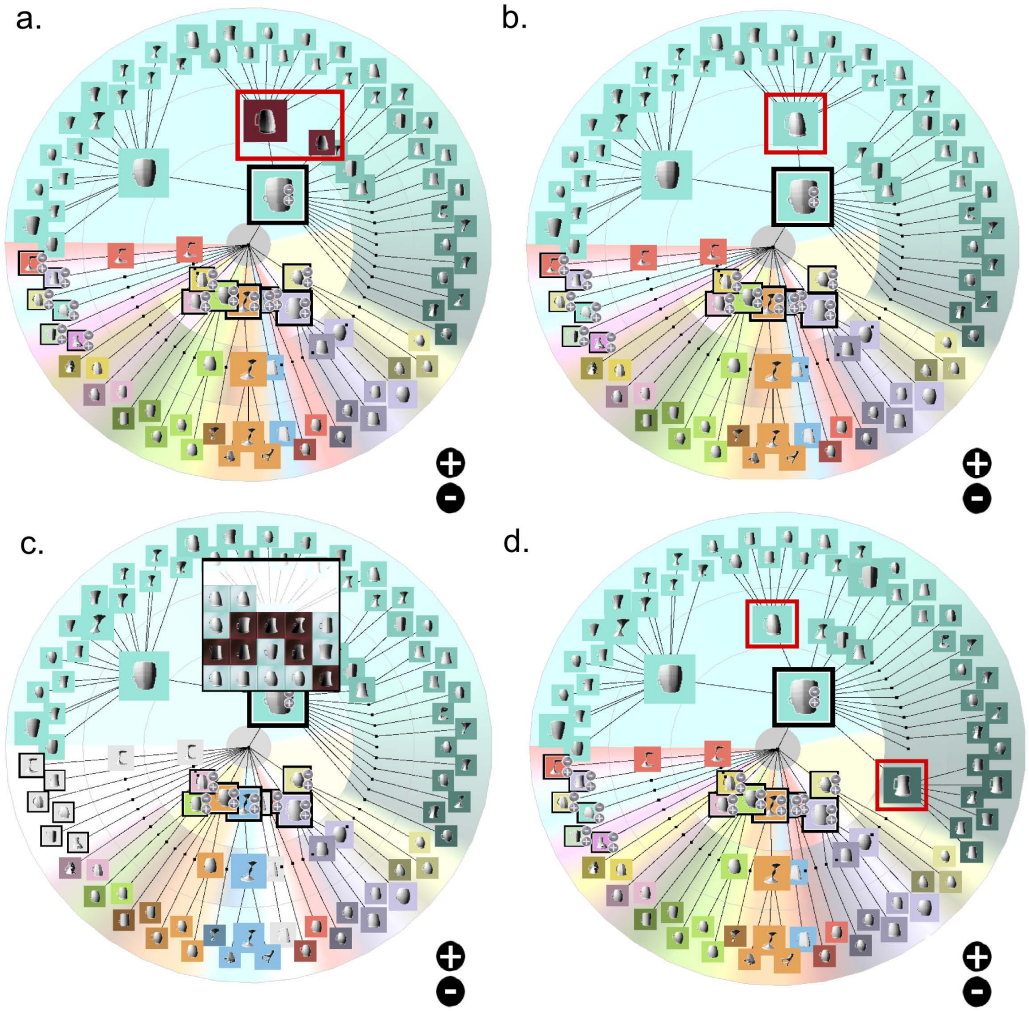

Figure 5.9: Two similar clusters in (a) are merged in (b). (c) Using a detail window, (d) selected members of a cluster are split into new clusters.

the new cluster is simply added at the end of the sub-nodes in Figure 5.9d. The user can also change the ordering of the sub-nodes by dragging.

5.4 Nested Icons

We nest geometric representations in *Cupid* using icons. The icons can either represent a cluster of similar 3D shapes or depict the individual cluster members at the bottom level of the hierarchy. The icon's background represents the color of the cluster, and the size/area indicates the number of cluster members. If an icon is selected (for merging or splitting a cluster), then the color of the

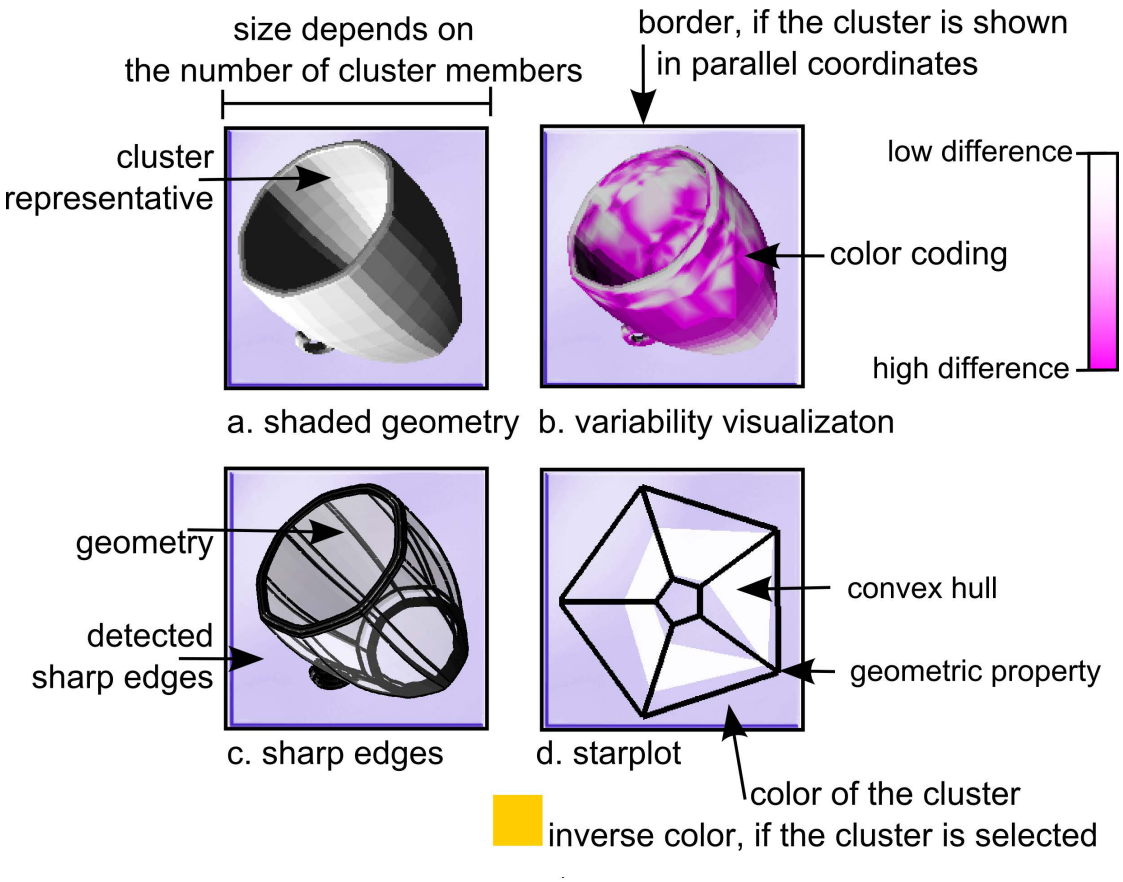

Figure 5.10: Types of icons: (a) Shaded geometry, (b) variability visualization, (c) sharp edges, and (d) starplots.

icon is inverted. The creation of an icon is shown in Figure 5.10. This consistent representation in both views ensures a good user experience. By default, the shaded geometry is shown from the same point of view set by the user (see Figure 5.10a). For icons representing a cluster, we display the 3D shape of the cluster member with the shortest Euclidean distance between its parameter values and the average values of the cluster. In the following, we describe three alternatives to the shaded geometry.

In some applications, the variability of the members within a cluster is in focus. For this purpose, we encode the variance between the nearest vertices of the cluster members to the cluster representative using a color map (see Figure 5.10b). This enables the user to see constant and varying regions of the 3D shapes within a cluster. As another alternative representation, we offer an illustration of the object's silhouette which is very important for comparing 3D shapes. Following the idea of iWires [40], the sharp edges of the geometry are drawn to display just enough information to identify the 3D shape. Additionally, the shaded geometry is shown with low opacity (see Figure 5.10c).

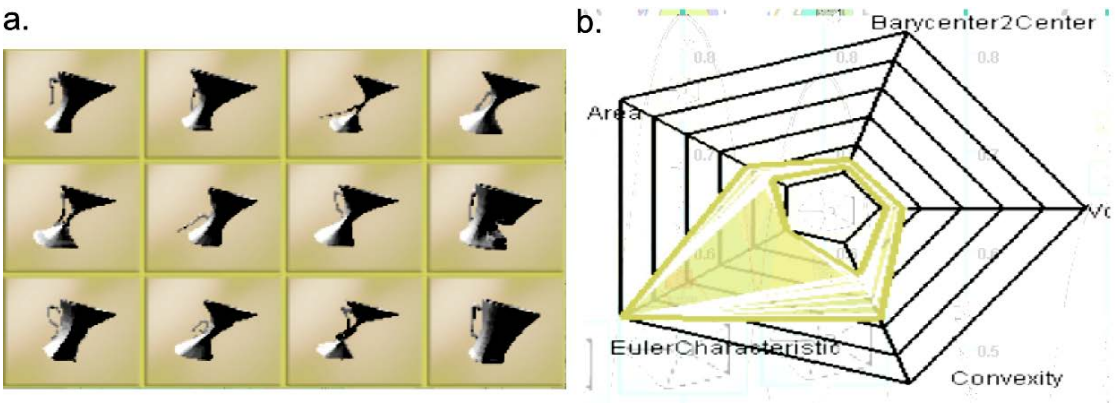

Figure 5.11: Detail window: (a) Visualization of the shaded shapes, (b) Visualization using a starplot

Finally, *Cupid* can depict derived geometric properties as starplots [21]. These are created analog to parallel coordinates but with polar coordinates. To ensure good readability, we only plot the convex hull of the line-paths for the cluster members. Because starplots are well suited for showing outliers, they enable the user to find implausible cups quickly, e.g., by looking at derived parameters such as shape stability (see Section 6.1). An example is shown in Figure 5.10d.

All visualizations are linked which enables the user to investigate clusters of similar 3D shapes across different views (compare to the *juxtaposition view* in Section 3.6). The icons in both views support the same operations which enables consistency during interaction. Clicking on an icon selects the related cluster. Mouse-over enlarges the related icons in both views and highlights the cluster in the composite parallel coordinates. Additionally, the user can drill-down into sub-clusters by clicking on the icon of an intermediate node. While the radial tree already displays the sub-clusters, an animated transition is performed in the composite parallel coordinates. In the composite scatterplot matrix the nodes are only replaced.

The 3D shapes of underlying sub-clusters are shown in a detail window. Usually, the shaded geometries are depicted side-by-side. An example is shown in Figure 5.11a. This representation follows the idea of small multiples presented in Section 3.2. Visualizing different shapes side-by-side enables the user to compare them easily. The same representation is shown, if the user selects a variability visualization or the sharp edges icon. Only the icons are replaced with the variability visualization or with the sharp edges. If the user selects the starplot representation, a starplot is shown in the detail window. Additionally, *Cupid* displays all line paths of the corresponding objects and the geometric properties' names. An example is shown in Figure 5.11b. The detail window enables to select shapes. If a shape is selected, the cluster can be split to modify the hierarchical clustering and the corresponding line path is shown within the composite parallel coordinates. Selecting of shapes within the small multiples representation is done with clicking on the shape. In the starplot representation, the shapes can be selected using the brushing tool. This approach follows the idea of linking and brushing, presented in Section 3.7.

5.5 Additional Features

All presented visualizations provide a basic zooming tool. The user can translate and scale the visualization. While in the scatterplot matrix and the hierarchical clustering only a region of interest is enlarged, the B-spines are automatically faded-in at high zooming levels. Furthermore, a basic fish-eye view is provided for all visualizations. The fish-eye transformation enlarges a region of interest and scales down the other parts (see Section 3.7). The fish-eye view is created with the following transformation:

$$T(d) = 1/\left(\sqrt{1.0 - d^2} * \tan\Theta_{fov} * 0.5\right),$$

where d is the Euclidean distance of the actual point to the center of the projection, and Θ_{fov} is the angle of the field of view (formula from PanQuake [74]).

Besides the composite parallel coordinates, scatterplot and hierarchical layout, *Cupid* provides a simple shape browser. The shape browser shows a list of all shapes using icons at the bottom. The user can select a shape by clicking on the icon. The selected shape is shown in the middle of the display. Additionally, the browser is linked with the other views. If the user selects an object the shape is also shown in the shape browser.

CHAPTER 6

Geometric Properties

Cupid provides multi-dimensional visualizations and representations of the cluster hierarchy for visual exploration of similar 3D shapes and the corresponding parameters (compare to Tasks 1 and Task 3). In addition, we provide derived geometric properties that help the user to identify interesting 3D shapes such as implausible or deformed cups (compare to Task 2). The properties can be depicted in the composite parallel coordinates, as background of the scatterplot matrix, or in the icons using starplots.

In the next section, we present our set of geometric properties in detail. In Section 6.2, we describe our approach for integrating the properties in the composite parallel coordinates. In Section 6.3 we discuss the approach to integrate the derived properties in the composite scatterplot matrix.

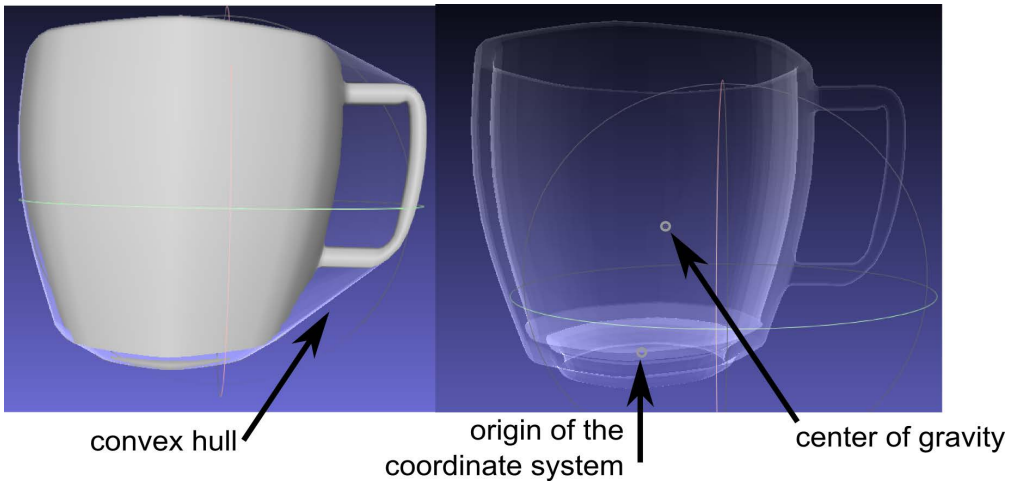

Figure 6.1: Visualization of a cup (shaded and x-ray) with the convex hull, the center of gravity and the origin of the coordinate system (created with MeshLab [28]).

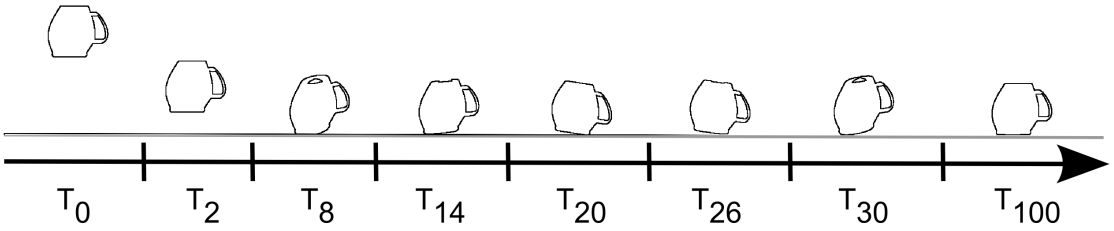

Figure 6.2: Tip over - Fall test: Some time steps of the tip-over test.

6.1 Derived Geometric Properties

In the pre-processing stage (see Figure 4.1b), we calculate the following geometric properties in addition to the similarity. We use these properties for exploring the parameter space:

- **Surface area:** $\sum_{i=1}^{|T|} A_i$, where A_i is the area of a triangle$_i$ and $|T|$ is the number of triangles or a cup.
- **Volume** enclosed by the mesh V of a cup.
- **Convexity:** V/C, where V is the volume enclosed by the mesh and C is the volume enclosed by the convex hull (see Figure 6.1).
- **Shape stability:** $1 - \frac{\|d-b\|}{r}$, where d is the origin of the coordinate system of the 3D shape (see Figure 6.1), b is the center of gravity of the shape (see Figure 6.1), and r is the radius of the bounding sphere.
- Number of **crossing faces**
- **Euler characteristic** $|V| - |E| + |T|$, where $|V|$ is the number of vertices, $|E|$ is the number of edges, and $|T|$ is the number of triangles. If the cup has an handle, then it should return 0, otherwise 2. If the Euler characteristic is neitjer 0 nor 2, then the meshtriangulation is invalid.

We have chosen these properties because they enable to detect invalid shapes (number of crossing faces larger than zero, Euler characteristic large) and to describe characteristics of cups such as shape stability or convexity. All properties are translation, rotation, and scale invariant. Finding good properties that describe 3D shapes is still an open problem. In order to deal with arbitrary geometries, we will include further parameters such as the fractal dimension of shapes (used to evaluate plants) in future work. Besides the geometric properties, we perform a drop test using a physics simulation engine (compare with Umetani et al. [96]):

- Tip over - **Fall test**: We drop a cup, and measure the orientation of the cup after 30 seconds. If the cup tips over, then the cup is not stable. Figure 6.2 shows an example of the simulation.

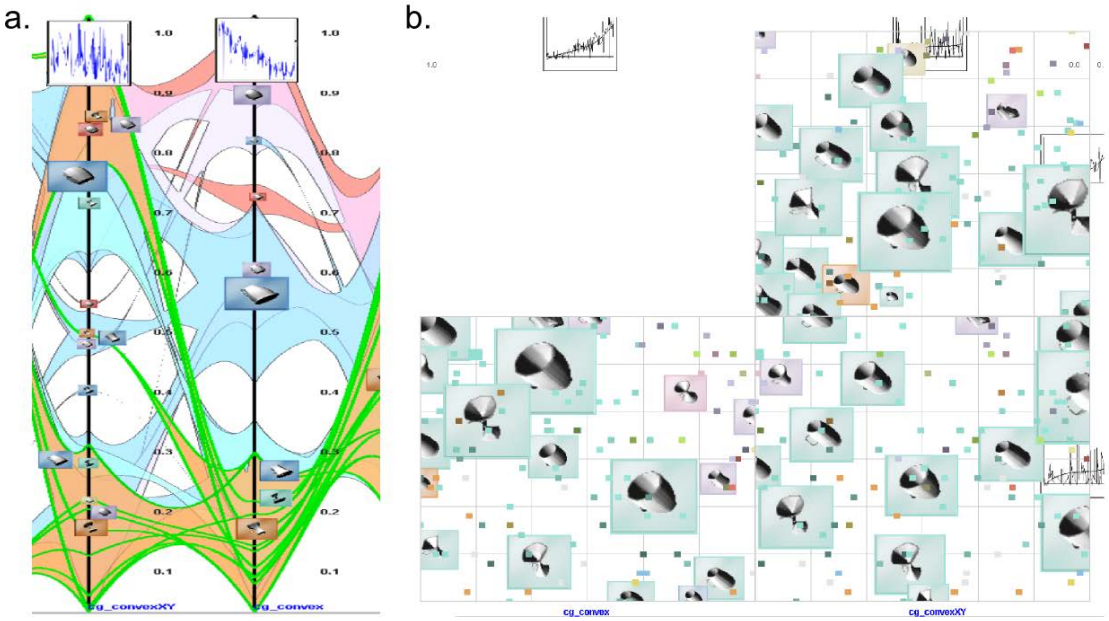

Figure 6.3: (a) A function plot in the parallel coordinates shows the derived geometric property *surface area* with respect to two parameters in order to identify possible correlations (like in (b)). (b) Function plots in the scatterplot matrix

6.2 Visualization of Geometric Properties in Parallel Coordinates

The derived geometric properties can be represented by starplots (see Section 5.4) or mapped to the axes of the parallel coordinates. The starplots can be enlarged in the detail windows, which are linked to the parallel coordinates. If the user selects a region of interest, the corresponding B-splines of the parallel coordinates are shown. This technique enables a result-driven exploration of the parameter space using linking and brushing.

Additionally, we add a function plot to each axis (see Figure 6.3a). Compare this to overloading views presented in Section 3.6. These plots show the function $f : x \rightarrow g$, where x is a parameter value of the geometry generator and g is a user-selected geometric property. In the function plot, we show the regression between the parameter value and the geometric property. This representation gives the user a hint about a possible correlation between the parameter and the geometric property and hides noise resulting from other parameters. Our function plots are similar to those in HyperSlice [97] at the diagonal positions of the matrix. Moreover, the function plots are linked with the composite parallel coordinates. A simple mouse-over in the function plot selects the corresponding B-spline in the composite parallel coordinates. This feature is useful to analyze outliers.

Additionally, HyperSlice [97] add 2D slices for the visualization of pairs of parameters. Our approach provides a similar visualization using the B-splines of the parallel coordinates. The geometric properties can be mapped to color and opacity, which are specified with a user-defined transfer function. For example, B-splines representing cups with a small surface area are shown

in green in Figure 6.3. The user can also fade-in (or hide) the B-splines in order to prevent visual clutter or to analyze a subset. For example, the user can hide all B-splines, which represent cups with no self-intersecting faces (property number of crossing faces is 0). The displayed B-splines represent invalid parameter combinations.

6.3 Visualization of Geometric Properties in the Scatterplot matrix

The composite scatterplot matrix also enables to analyze the geometric properties. We integrate the geometric properties in the background. Furthermore, the composite scatterplot matrix provides function plots.

The geometric properties can be mapped to the color and opacity of the markers with a user-defined transfer function. This visualization enables to detect parameter regions, which results in invalid 3D shapes (i.e., 3D shapes contains errors or tip over in reality). Additionally, *Cupid* provides the function plot, presented in Section 6.2, at each coordinate axis. This function plot is very good to analyze the effect of each parameter. Figure 6.3b shows an example.

CHAPTER 7

Implementation

In this chapter the software architecture and relevant implementation details of *Cupid* are presented. The next section gives an overview of the system architecture. In Section 7.2, we discuss all implementation details. In the last section, we present all relevant aspects of the cup generator as well as the corresponding parameter space.

7.1 Overview of the System

We implement *Cupid* in *VolumeShop* [20], which is a 3D visualization framework and rapid prototyping toolkit. *VolumeShop* is implemented in C++, and the user-interface is built with Qt.

The main strength of *VolumeShop* is its modular design. All components like renderings and object loading can be extended by a plug-in mechanism. The plug-ins enable to adapt *VolumeShop* to the needs of the visualization. Additionally, the modular design of *VolumeShop* ensures a good separation between data, logic and rendering, according to the model-view-control (MVC) pattern. *VolumeShop* provides the following plug-ins:

- **Renderer** plug-ins are commonly used to display data using a particular rendering algorithm [20].
- **Interactor** plug-ins are used to process user input such as mouse events. Additionally, the interactor plug-in can display graphical output [20].
- **Compositor** plug-ins are responsible for combining the output of multiple renderers and interactors in a viewer [20].
- **Importer** plug-ins enable to load data into a resource such as a volume or an image [20].
- **Exporter** plug-ins enable to write resource data [20].
- **Editor** plug-ins enable to provide advanced GUI elements for controlling certain parameters [20].

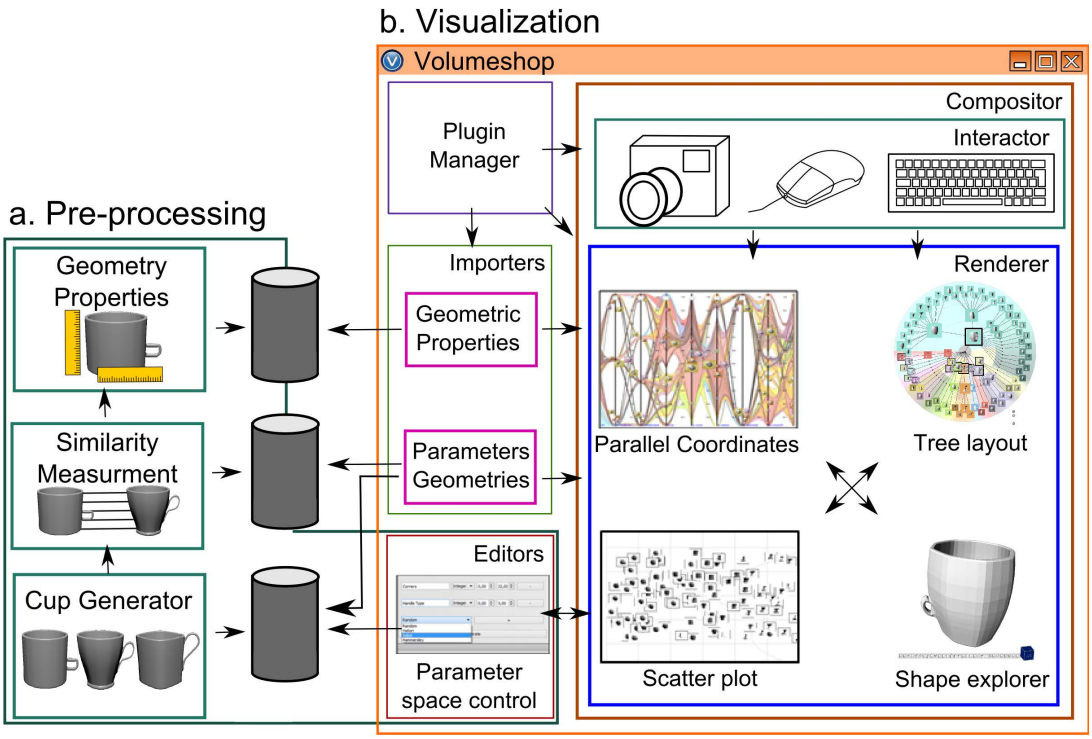

Figure 7.1: Overview of *Cupid*. (a) Pre-processing pipeline: First, the user defines the parameter space. Then, the cup generator creates the cups. Finally, the similarity is measured and the geometric properties are calculated. (b) Before the user can explore the cup generator, the parameter space, the cups, the result of the similarity measurement and the derived geometric properties are imported. Then, *Cupid* calculates the hierarchical clustering. At last, the user selects the desired visualization.

- **Host** plug-ins provide a text-based interface to *VolumeShop* objects. They can be used for making *VolumeShop* scriptable in a certain language [20].

Besides of *VolumeShop*, a lot of other open source (information) visualization applications and libraries are available like *The Visualization Toolkit (VTK)* [99] and *Pandas* [73]. In contrast to *VolumeShop*, they provide several implementations of visualizations like scatterplots and parallel coordinates. However, they only provide a very basic representation of such visualizations. Features, like the branching technique and edge bundling, are not provided. Thus, we have to create our visualizations from scratch.

VolumeShop provides several basic data structures to share data. If a data object is modified, an action handler notifies all plug-ins that use the data. Furthermore, *VolumeShop* provides the linking of the application's properties. The user can define a link between two properties across different plug-ins. If two properties are linked, a change of one property updates all the linked properties. This implementation enables an easy communication between several

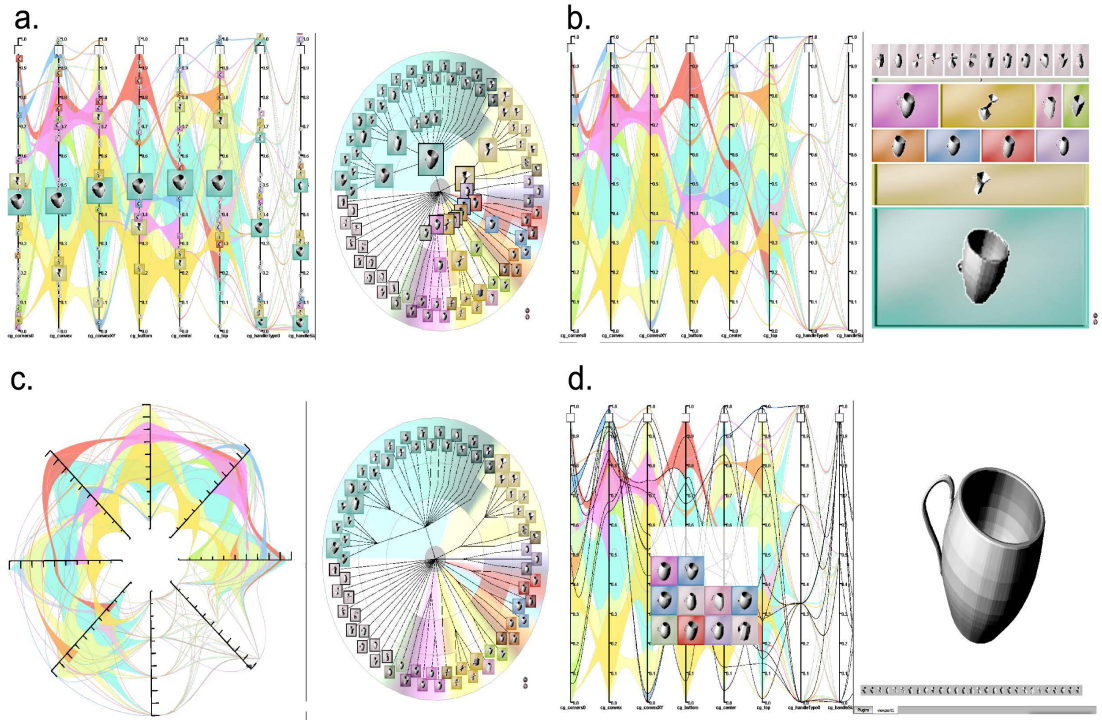

Figure 7.2: Examples of configurations of *Cupid*. (a) Parallel coordinates and radial tree, (b) Parallel coordinates and treemap, (c) Starplot + dendrogram, (d) Parallel coordinates and shape browser

plug-ins. The excellent support for implementing integrated and linked visualization techniques makes *VolumeShop* to an ideal visualization framework for the *Cupid* system.

An overview of *Cupid* is depicted in Figure 7.1. We use *VolumeShop* for visualizing the geometric parameter space and for controlling the pre-processing steps. Before the visualizations run the first time, the user has to define the parameter space and to execute the pre-processing pipeline. The user describes each parameter by its name, type, and the sample range. Moreover, the number of samples to generated as well as the sampling technique has to be determined. *Cupid* offers standard random sampling and low-discrepancy sampling using the Halton, Sobol or Hammersley sequence. Then, the application creates a batch file. The batch file calls the geometry generator to create the 3D shapes. Then, *Cupid* calculates the similarities between each combination of shapes and the corresponding geometric properties. After calculating the pre-processing steps the user imports the data, and *Cupid* calculates the hierarchical clustering. After creating all data, *Cupid* generating the desired visualizations. The user can select between several visualizations. In Figure 7.2 several example configurations of *Cupid* are given.

7.2 System Components

Cupid is composed of a collection of components, as seen in Figure 7.1. In the following sections, all system components are presented in detail.

Parameter Set Editor (Editor)

The parameter set editor is the central control point of *Cupid*. The editor is used to describe the parameter space, to load the data, and to perform the hierarchical clustering. *Cupid* loads the data by invoking the importer functions.

The editor provides a user interface to create the parameter space. As presented in Section 4.1, the user has to define the parameter names, the parameter type, and the sample range. Additionally, the user chooses the number of samples and the range to be sampled. Then, the geometric shapes are calculated using the geometry generator. Afterwards, the similarity between all pairs of shapes are calculated, along with the geometric properties.

As soon as the data are loaded, *Cupid* creates the hierarchical clustering. As presented in Section 4.2, the user can define the thresholds st, and Δst. If the user modifies the parameters, the hierarchical clustering is recalculated, and the visualizations are notified to update.

Pre-processing Pipeline: Similarity Measurement and Geometric Properties

Cupid derives geometric properties and calculates the similarity with external applications. Outsourcing to external applications enables it to replace pre-processing operations quickly and test different pre-processing pipelines.

For calculating the similarity measure, the shapes are aligned first. We use the ICP algorithm of the Point Cloud Library (PCL) [82] for this task. Then, we search for each vertex of the first mesh the nearest vertex in the other mesh and sum-up the distances. This process can be efficiently solved using a kD-tree.

In the last step, some geometric properties are derived from the 3D shapes. While the surface area, shape stability, and the number of self-crossings can be calculated easily, the volume of an arbitrary, watertight mesh is more complex. *Cupid* voxelizes the geometry using the method from Crane et al. [31]. The resulting voxels are counted to calculate the volume. The first step is to calculate the axis aligned bounding box (AABB) of the shape. Then, each triangle of the shape is rendered into a slice of a 3D texture using an orthogonal projection. The near plane of the orthogonal camera is set to match the current slice. The far plane is set to infinity. Additionally, a stencil buffer is initialized to zero. Then, all triangles of the shape are rendered. The stencil buffer is incremented if the triangle passes the back face test. Otherwise, the stencil buffer is decremented. Voxels are created for each non-zero entry in the stencil buffer. The algorithm repeats this process for each slice of the 3D texture. The stability tests are performed using the open-source Bullet physics engine [23].

Cups and Parameters Loader (Importer)

The parameter space and geometric properties are stored using XML files. The 3D shapes are given in the commonly used Wavefront object format. Both formats store the data as ASCII characters and are readable by humans.

Loading a large set of shapes is a time-intensive task. Our first object loader implemented in C needs more than two minutes to load the data set. This loader streams the files in the memory and reads the data using the *sscanf* function. To reduce the time for loading shapes, we use file mapping. Using file mapping loads the complete data set into the main memory without streaming the files. Furthermore, we find the conversion from ASCII characters to floating-point numbers to be a bottleneck. Loading the data by converting a *char*-array to float using the *atof*-function takes around one minute (using an Intel Core 2 Duo processor with 2.26 GHz, and Windows 8; see Section 9.7). Using an own conversation function, which does not process any error handling, reduces the time for loading significantly. Instead of loading the data in one minute, our loader imports the data in less than two seconds (the same configuration, only the *atof*-function is replaced with our conversion).

Parallel Coordinates, Scatterplot Matrix, Hierachical Layout, and Shape Browser (Renderer)

As presented in the earlier sections, *Cupid* provides composite parallel coordinates, scatterplot matrix, hierarchical layouts and a shape browser. All visualizations are linked. The visualizations can be adjusted by modifying their properties. To link the visualizations, we create internal objects, which save the current visualization state. These states are used to communicate between the visualizations. To adjust the visual representation, the user can adjust the properties of the visualization. The user can change the size of the icons, fade-in (fade-out) the colored polygons in the parallel coordinates, and much more. The properties are useful to test different visualizations, e.g., the user can fade-out all icons in the parallel coordinates, but visualize a label using the hierarchical clustering. In the appendix, an overview of all properties is given.

All visualizations are created using OpenGL and GLSL as shader language. This combination provides enough computational power to reduce the latency of our visualizations. The main advantage of using our architecture is that individual tasks are outsourced into shader programs, e.g., the display of the basic visualization elements is done using the shader. The CPU passes all data that are needed to calculate the visualizations (value of geometric property, color, etc.). The graphical elements are created using a geometry shader. Changing the representations can be easily done by selecting another shader. This architecture enables a better separation between logic and viewing code.

sRGB (Compositor)

The visualizations and the interactors have to be combined for rendering. We implement a simple compositor. The compositor renders all visualizations into a framebuffer object (FBO). Then, the visualizations are converted from the linear RGB color space to the sRGB color space. Optionally, the fish-eye view is rendered (see Section 5.5).

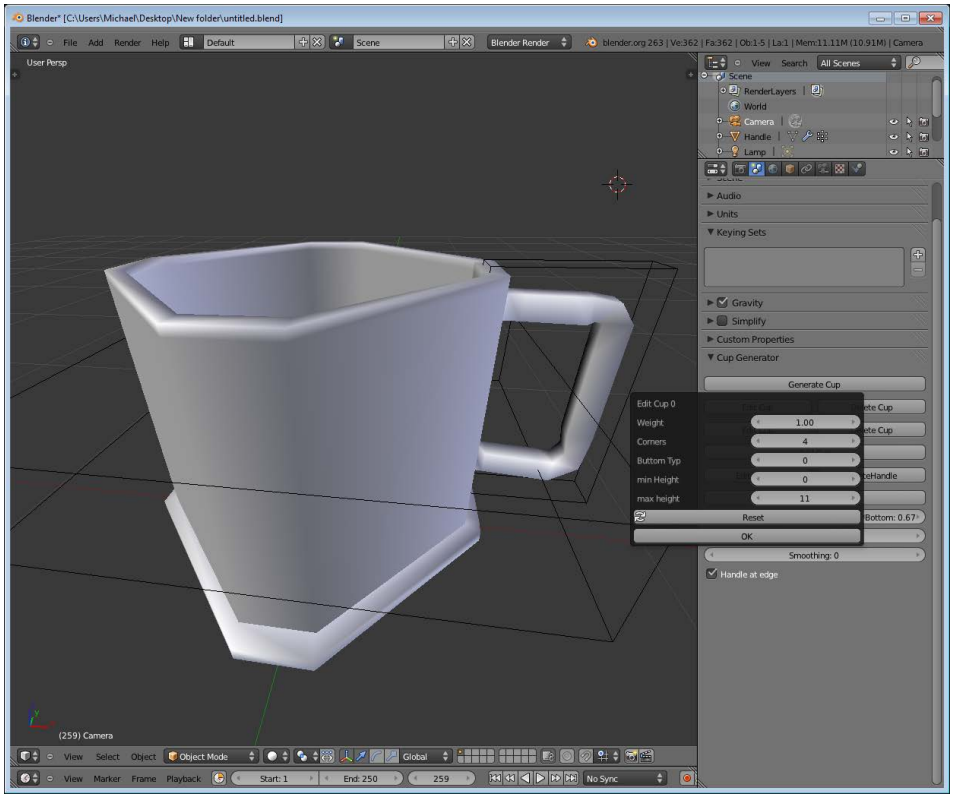

Figure 7.3: The cup geometry generator: On the right panel, the GUI elements are shown.

7.3 Cup Generator

As an application example, we explore the results of a cup generator that is used for testing the object recognition capabilities of a domestic robot. The parameter space consists of eleven parameters (see Table 7.1). Certain parameters define parts of the cup such as handle type or number of corners, and other parameters represent global modifiers of the shape such as convexity. Additionally, the cup generator creates texture coordinates and enables to morph between several cups.

The cup generator is implemented using Blender. [16] Blender is an open-source 3D computer graphics software used for creating animated films, video games, visual effects, and 3D models. Blender provides tools for 3D modeling, UV unwrapping, texturing, animating, and much more. Additionally, Blender supports Python scripting for the creation of new features, prototyping, importing from and exporting to other formats, and for the implementation of custom tools. We use Python to implement the cup generator.

The cups, created by the cup generator, consist of a cup body and a handle. First, the generator creates a basic cup body. This body is defined by the number of corners, the type and size of the bulge at the top, and the width at the top, center, and bottom of the body. If the user applies the morphing feature of the cup generator, a shape is created, which saves a correspondence be-

Parameter name	Description	Type
number of corners	roundness of cup	discrete
convexity side	convexity from side-view	continuous
convexity top	convexity from top-view	continuous
bottom width	width of cup at bottom	continuous
center width	width of cup at center	continuous
top width	width of cup at top	continuous
bottom type	type of bottom	categorical
bulge at top	if true, then a bulge is created	categorical
size of bulge	size of the bulge	continuous
handle type	type of handle (e.g., round, open, closed)	categorical
handle size	size of handle	discrete

Table 7.1: Parameters of the cup generator.

tween the cups. Afterwards, the cup generator creates the handle depending on the parameters handle type and handle size. If the cup should be morphable, also the correspondence between the handles is calculated. In the next step, the body and handle are joined. In the last step, we smooth the cup and modify the cups according to the convexity parameters.

Figure 7.3 shows an example of the cup generator. On the right side, the GUI of the cup generator is seen. The user can define the parameters of the cup. Additionally, the generator can also be used from the command line. The parameters of the cup are set by using environment variables. This feature enables to sample the parameter space of the geometry generator. Table 7.1 gives an overview of all parameters of the cup generator.

CHAPTER 8

Results

In the introduction, we present several tasks for analyzing a geometry generator. In this chapter, we provide solutions to the tasks presented in Section 1.1 using *Cupid*. In Section 8.1, we present workflows to analyze the effect of each parameter. Then, a workflow is discussed in Section 8.2 to select shapes according to some parameter values for using a query-based exploration of the cup generator. Subsequently, the tasks, presented in the introduction (see Section 1.1), are in focus. In Section 8.3 we evaluate the cup generator to find categories of similar shapes (see Task 1). Furthermore, we present our workflow to find invalid cups (see Task 2). According to the third task, we present in Sections 8.5 and 8.6 our workflow to determine the influence of parameters and to identify sensitive regions of a parameter (see Task 3).

We also evaluate our application together with domain experts of testing computer vision (CV) applications. All details of the evaluation are presented in Section 8.7.

8.1 Analysis of the Effect of the Parameters

A very important task is to understand the changes resulting from the parameters. Even if a geometry generator has a description of each parameter, it is wise to analyze the effect of each parameter. The description can be ambiguous, or the result of a parameter can be difficult to understand.

In the composite parallel coordinates, we use the line brush tool and the icons to evaluate the effect of a parameter. The line brush tool is a well-suited tool to select all shapes within a region without considering to the clustering. We usually start at the low end of the parameter range. Then, we brush a line upwards. If the brushing tool crosses a line path, the corresponding shape is shown in the detail window. To understand the effect of a parameter, the user looks for continuous changes of the 3D shapes. This workflow is well suited for parameters, especially, if the parameter strongly influences the 3D shapes. Another workflow for understanding the change of a parameter is to use the icons. The user can compare icons of the same cluster in different parameter regions. The detail window represents all members of the cluster. The user

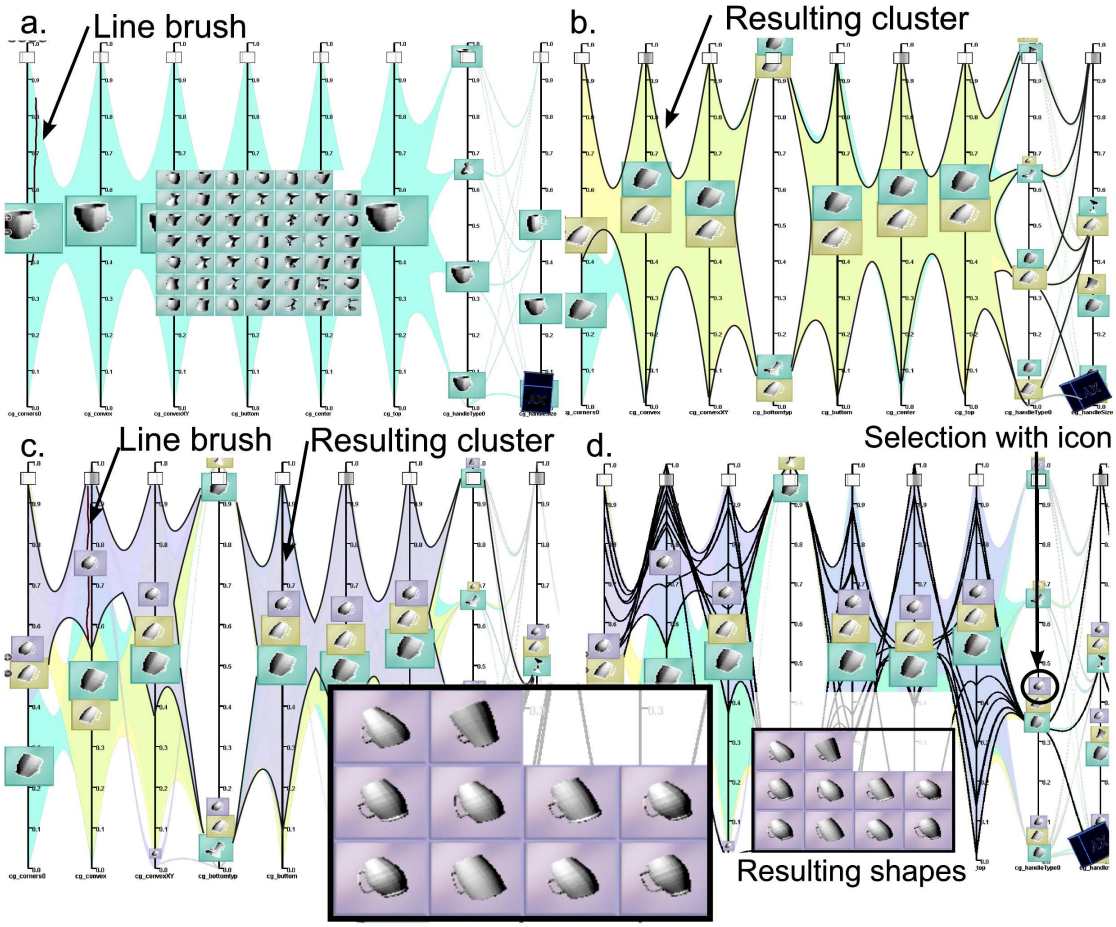

Figure 8.1: Query-based Exploration: (a) Selection of round cups using the line brush tool. (b)Result after spliting the round cups. (c) Result after splitting of cups with high convexity (d) Detail window shows all round, convex cups with an angular handle. Note that the convexity also depends on other parameters, such as the *top width*, *center width* and *bottom width*.

can estimate the change of the parameter when comparing the parameters of different parameter regions.

Analyzing the effect of the parameter is not limited to composite parallel coordinates. The composite scatterplot matrix can also be used though. The user analyzes the marker's position. Instead of employing the line brush tool, the user selects the markers with the lasso tool.

8.2 Query-based Exploration of the Geometry Generator's Parameter Space

The icons and the line brush tool are great tools for the selection of a region-of-interest, and to investigate the impact of a parameter. However, in some application scenarios, the user may want to select shapes according to some specific parameter values (e.g., all cups with a high *convexity side* parameter and an angular handle). For example, the user might need to see all cups with a high number of corners, high convexity, and a square handle. In Section 2.1, several applications are presented to query shapes according to attributes.

Cupid also provides a query-based workflow using the merging and splitting tool at the composite parallel coordinates. First, all shapes are grouped into one cluster. Then, the user can select all shapes within a parameter region of interest and split the cluster. The resulting cluster groups all shapes with the parameter region. The user can repeat this process for the other parameters. An example of this workflow is given in Figure 8.1. First, all shapes with a high number of corners are grouped into a cluster. Then, the shapes with a high convexity value are selected, and lastly, the shapes with an angular handle. The resulting shapes can be explored with the detail window. This task is not limited to the parameters of geometry generator, but geometric properties can also be used. For example, the user can select all round shapes with a high stability value.

In some application scenarios, the user is interested in selecting a combination of two parameters. For example, the user wants to select all cups with a similar parameter value for *convexity top* and *convexity side*. First, the ordering of the parallel coordinates has to be adjusted. Both parameters need to be placed side-by-side. Then, the user has to mirror one parameter horizontally. Next, the user selects all B-splines which overlap in the center between both parameters using the lasso tool. The desired cups are shown in the detail window. To select a combination of three or more parameters, the user uses the merging and splitting tool. First, the user groups all B-splines into a cluster, which overlap. The user repeats this process for the other parameter combinations. The B-splines are split from the cluster which do not overlap. The resulting cluster contains the desired cups. Note: This approach only works with a bundling factor of 0.

An easier workflow is the use of the composite scatterplot matrix. Instead of the line brush tool, the lasso tool has to be used to group shapes. First, the user hides all scatterplots that show other parameter combinations. Then, the user selects only the markers that are located in the diagonal position of the scatterplot. A combination of three or more parameters can be chosen with the splitting and merging tool.

However, we recommend the use of the composite parallel coordinates if the user wants to limit the range of each parameter, for example, if the user wants to select all angular shapes with a high convexity and a round handle. If the user wants to query a mix of parameters, then the composite scatterplot matrix approach is the right choice (compare with Harrison et al. [43]).

Figure 8.2: Some examples of detected categories: The cups vary in profile, roundness, handle and width.

8.3 Finding Similar 3D Shapes and the Corresponding Parameter Values

The first task is to characterize the generated 3D shapes. We want to detect all variations of the shapes and to identify the corresponding parameter values. Some example shapes are shown in Figure 8.2.

Initially, the results from the hierarchical clustering are inspected in the radial tree. We can adjust the parameters of the clustering interactively to improve the results (e.g., having between five and nine clusters at the top hierarchy level). After this step, clusters can be split or merged manually. In Figure 8.3a, for example, the 3D shapes within the top cluster look rather different from their parent and are thus split into separate clusters. In contrast, the sub-clusters in Figure 8.3b look very similar to their parent. The leaf elements group only very similar 3D shapes together by DBSCAN, which uses a low similarity threshold st. Furthermore, the user can quickly detect variations of shapes within a cluster using the starplot.

After clustering the 3D shapes, the corresponding parameter values can be studied easily using the composite parallel coordinates or scatterplot matrix. We only need to highlight (mouse-over) the selected icon because all views are linked. The corresponding cluster is highlighted in the multi-dimensional visualization. In the following, we use the hierarchical clustering to analyze the parameter space in detail.

8.4 Finding Invalid and Implausible 3D Shapes

The next task is to find unwanted or (physically) implausible cups. Such shapes would affect computer games or evaluation systems for computer vision and thus need to be identified. *Cupid* derives geometric properties to detect unwanted 3D shapes (see Section 6.1). Such cups can, for instance, have a high value for shape stability or a very small volume of the enclosing shape. Moreover, erroneous meshes can be detected using the number of crossing faces. By mapping the derived properties to the axes in the composite parallel coordinates, the scatterplot matrix or by using the starplot icon, the user can quickly identify such shapes.

While invalid cups can often be detected automatically, we are also interested in cases that are difficult to describe by the derived properties. For example, if a cup looks like a beer mug, it is very difficult to decide between cup and no cup. A beer mug is very stable and can be used just as a cup. The difference between a beer mug and a cup depends on semantic knowledge. Another example are cups with a very small handle. Some users classify them as unwanted and others

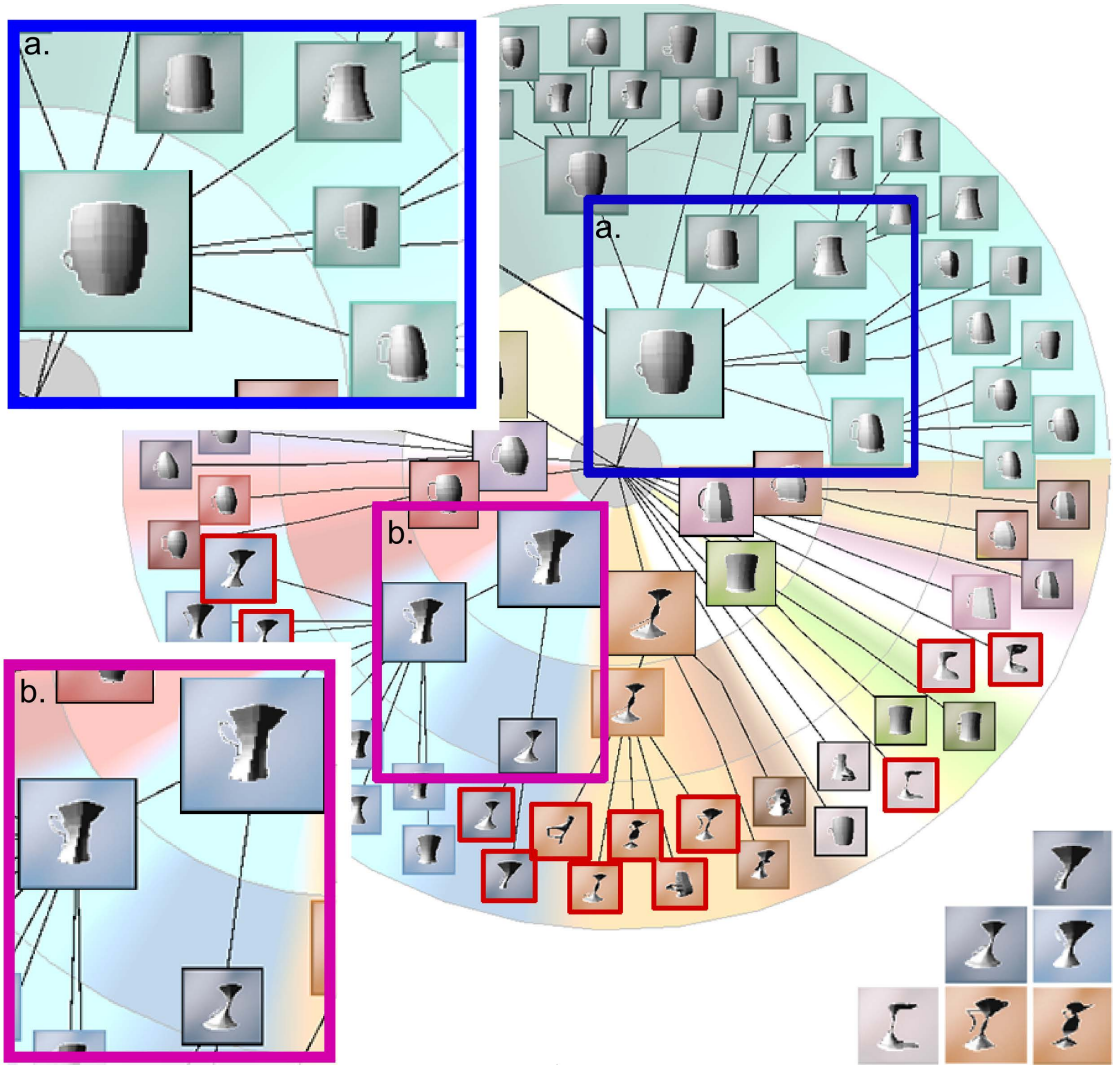

Figure 8.3: Parent and child icons in the blue rectangle look different and are thus split into separate clusters. In contrast, the icons in the pink rectangle belong to the same category. Icons highlighted with the red rectangle are labeled as unwanted cups by the user.

as cups with novel design. To handle such ambiguous cases, the user has to manually determine the validity of the cups. In the radial tree, we first inspect the icons located at the bottom level of the hierarchy. Since the icons are grouped by their similarity, it is easy for the user to spot outliers. In Figure 8.3, the cups inside the red rectangle have been labeled as unwanted by the user. The corresponding B-spline curves are drawn in the linked composite parallel coordinates, where we can investigate the respective parameter values. In the composite scatterplot matrix, the corresponding markers are highlighted. We can also change the view-point for the cups or use a detail window for further inspection.

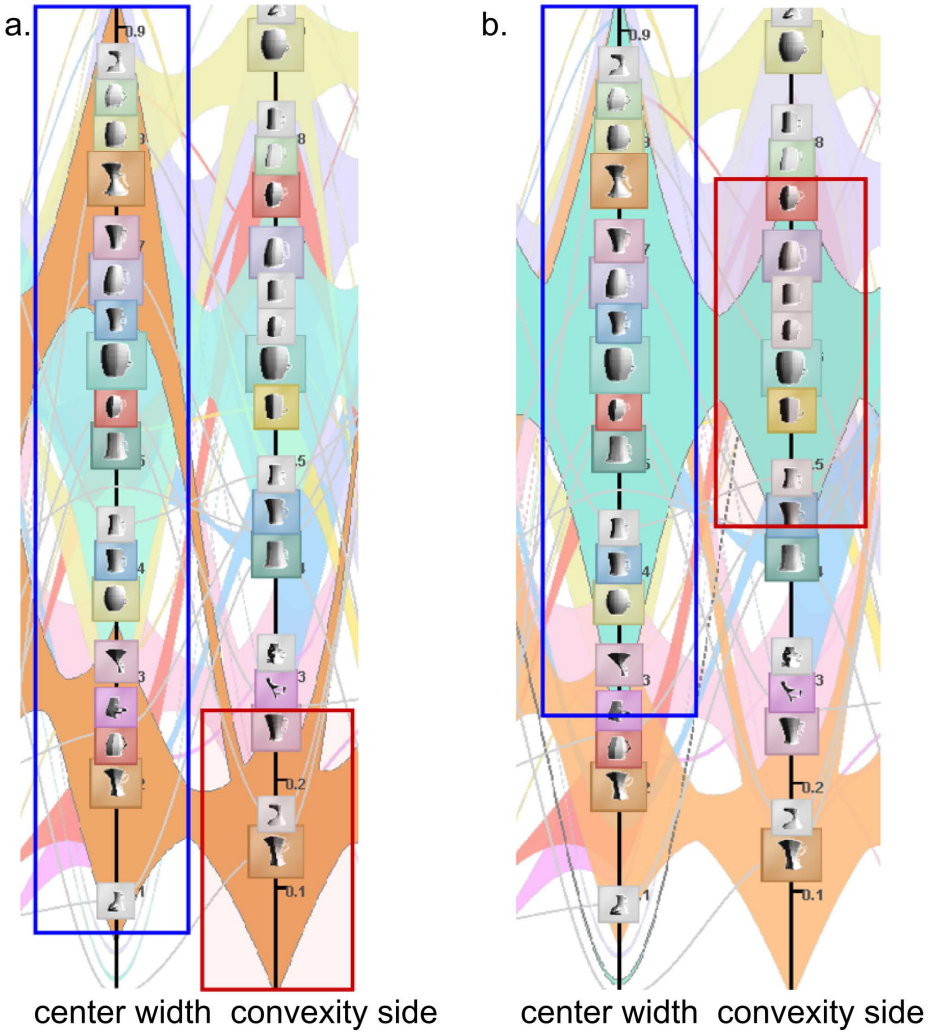

Figure 8.4: Exploring the influence of parameters on the created cups: The orange (a) and turquoise (b) clusters cover large regions of the parameter *center width* but only a small region of parameter *convexity side*.

8.5 Evaluation of the Influence of Parameters

According to Task 3, we study the influence of parameters in this section and identify sensitive regions in the parameter space in the next section. The clustering provides information about the influence and sensitivity of the generator with respect to its parameters.

To determine the influence of a parameter, we use the composite parallel coordinates and analyze the distribution of clusters of similar 3D shapes. If the polygon of a cluster covers only small portions of a parameter range, the parameter has high influence because small changes have a large effect on the resulting geometry. As an example, see the orange and turquoise clus-

ters for the parameter *convexity side* in Figure 8.4. In contrast, if large changes in the parameter value result in similar 3D shapes, the parameter has low influence. The individual polygons (or branches) of a cluster then cover large portions of the parameter range, e.g., see the orange and turquoise clusters for *center width* in Figure 8.4.

Next, we investigate 3D shapes that have been classified as noise by the clustering. Many such shapes have been generated by small values for the parameter *number of corners*. We select those shapes using a line brush and inspect the cups in a detail window. As shown in Figure 8.5c, the corresponding cups look very angular. Moreover, the clusters in this parameter range are very narrow, while the clusters for larger values of *number of corners* are larger. This is not surprising because changing from three to four corners has a higher influence on the generated 3D shape than changing from 30 to 31, for example. Using additional detail windows, we can investigate the influence of *number of corners* on the roundness of the generated shapes.

We analyze other parameters in a similar way. For example, the parameter *convexity side* also has a high influence on the resulting cups because the individual clusters cover only small parameter ranges and modify the profile of the shape (see Figure 8.5b). In contrast, the parameter *center width* has low influence because it influences the shape in combination with *top width* and *bottom width*. The respective clusters in Figure 8.5a cover larger portions of the parameter range.

The evaluation of the influence of the parameters can also be solved using the composite scatterplot matrix. Usually, it is enough to plot one scatterplot with the analyzed parameter as row (or column). We only analyze the x-axis (or y-axis) of all icons, because the y-axis (x-axis) depends on the other parameter. If the icons cover only a small portion of the parameter range, then the parameter has a high influence because small changes have a large effect on the resulting geometry. In contrast, if large changes in the parameter value result in similar 3D shapes, the parameter has low influence. The icons of a cluster cover large portions of the parameter range. It is important to note that the icons depend on the parameter to analyze and another one. So, several icons of the same cluster occur (because the other parameter affects the position of the icons). We recommend the use of the composite parallel coordinates because the distribution of icons is easier to follow.

However, the composite parallel coordinates and scatterplot matrix can also be used to analyze pairs of parameters. We can use the same workflow as presented earlier, but both (or more) parameters have to be consider. In the composite parallel coordinates, the user have to analyze the location of the icons as well as the colored polygons. The pair of two parameters has to be visualized close together, and the user has to apply a low branching parameter (to see a high level of detail). If each icon is connected to only one other icon, then the pair of parameters has a high influence. If the icons are connected with several other icons of the other parameter, then the combination of parameters has low influence. In the composite scatterplot matrix, the user can visualize a pair of coordinates in one scatterplot. If only few, big clusters are shown over the entire scatterplot, then the pair of parameters has a high influence. If a big cluster is located in a small area of the scatterplot, than the combination of parameters has a high influence. If the icons corresponding to the cluster are located over the whole area, then no correlation occurs.

Our presented workflow is able to find the influence of a parameter. Note, that it is meaningful to consider the effect of each parameter separately while analyzing the influence (see section 8.1). If no effect of the parameter can be detected, then the influence of the parameter

is low, regardless of the distribution of icons. Furthermore, finding the effect of each parameter helps to understand the geometry generator. Common sources of errors are the similarity measurement, and a too low sampling rate. Generally, we recommend to sample the parameter space with low discrepancy sequences with a high sampling rate. Low discrepancy sequences ensure that the whole parameter space is covered uniformly.

The derived geometric properties can also be used to evaluate the influence of a parameter. The idea is to find correlations between a parameter of the generator and the derived geometric properties using the function plots (e.g., surface area or shape stability). If such a correlation is found, the parameter has a high influence on the selected geometric property (and on the result). An example is shown in Figure 6.3, where the parameter *surface area* influences the derived convexity of the shape (see Figure 6.3). The function includes some noise because also other parameters affect the convexity.

8.6 Sensitivity Analysis of Parameter Regions

The last task is to find sensitive parameter regions. As discussed in the introduction, a sensitive region of a parameter affects the resulting 3D shapes more than other regions of the same parameter. From this follows that parameter regions where a lot of clusters are located are potential parameter areas with a high sensitivity. A good starting point for finding sensitive regions is to look for narrow clusters or noise. The *number of corners* is such an example that has been discussed already (see Figure 8.5c).

The main challenge of finding a sensitive region is to identify whether the number of clusters within a (small) region depends on the current parameter or its interplay with other parameters. In Figure 8.5c, for example, several clusters with round cups but different silhouettes can be seen. The difference results from other parameters such as *handle type* and *convexity side*. In contrast, the clusters with lower values for *number of corners* have different roundness, where the variations stem mainly from the parameter itself. The corresponding region has a high sensitivity.

Another example of a parameter with high influence is *convexity side*. To identify sensitive regions, we analyze the parameter in the same way as the parameter *number of corners*. As shown in Figure 8.5b, this parameter varies the profile of the 3D shape. Low convexity values result in concave cups and high convexity values result in convex cups. While the cups at the top and bottom (mainly) differ according to the amount of convexity/concavity, a significant change is found in the middle. The clusters in the middle contain straight cups (neither convex nor concave), the clusters above and below differ according to the convexity.

As presented earlier, sensitive regions of a parameter are regions, where a small change of the parameter results in large changes of the shapes. The changes of the parameter can also be detected using the starplot icon to visualize the derived geometric properties. If high variations in the starplot occurs, then the shapes differ. If the variation of the starplot is low, then the shapes should be similar. The advantage of this technique is that variations of shapes can be detected easily. The user only has to analyze the variations of the derived geometric properties. A disadvantage of this technique is that the user can only see significant changes, if a geometric property changes. However, the *surface area* and the *volume enclosed by mesh* parameters

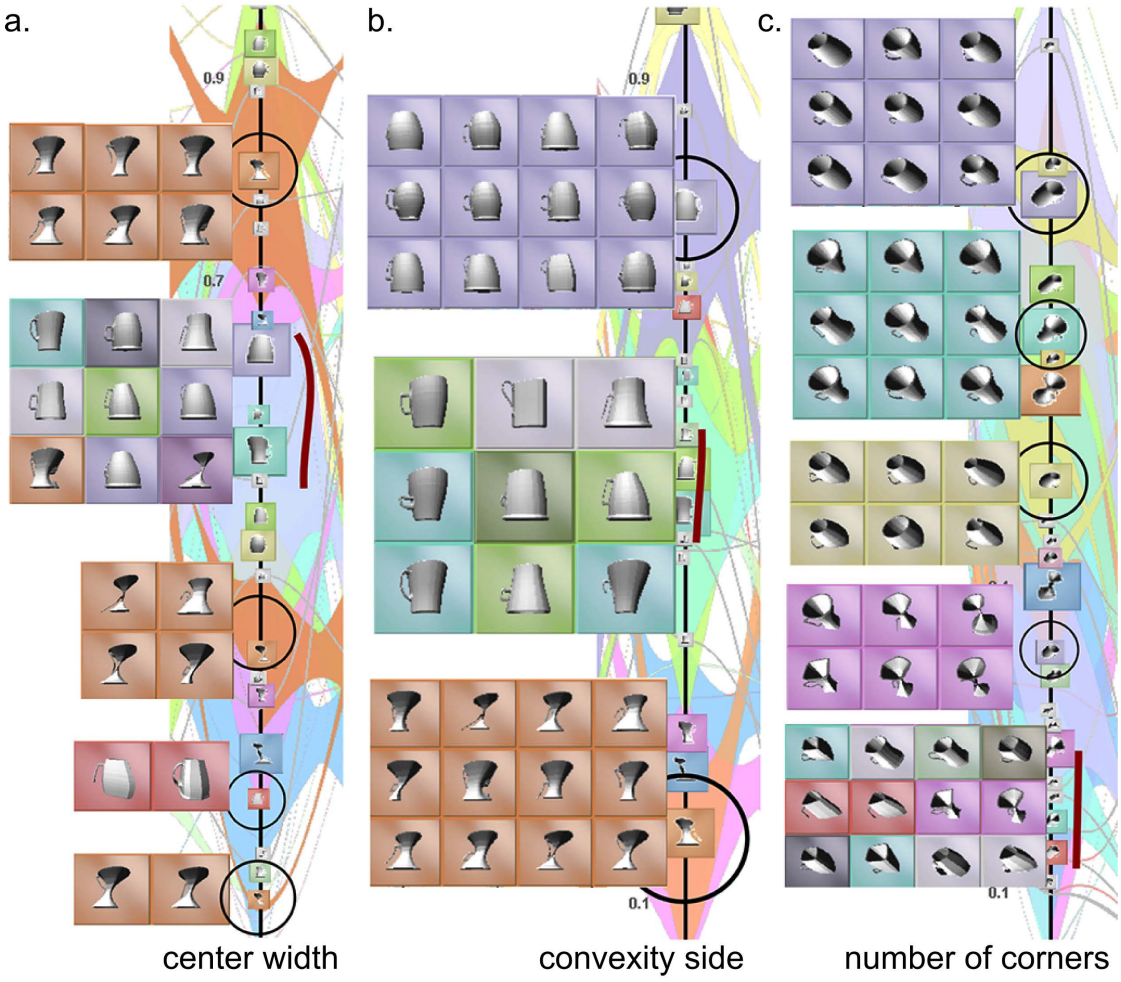

Figure 8.5: Comparison of selected 3D shapes using a detail window: (a) The orange cluster covers a large portion of the parameter value with similar 3D shapes. (b) The parameter has a high influence on the resulting shapes, since clusters are rather narrow. (c) Changes in low values of *number of corners* already create different 3D shapes, while changes in large values produce rather similar cups.

should vary, if a change of the parameter occurs. Furthermore, the variations of the geometric properties are not equal to the similarity of two shapes. For instance, if both meshes are highly convex (measured with geometric proper *convexity*), then the shapes are similar. If one shape is more convex than the other one, both shapes can also be similar, because high changes of convexity are not very important (i.e., one shape is a little bit more convex than the other).

The composite scatterplot matrix can also be used for the sensitivity study. We analyze the position of the icons in a similar way as presented in the composite parallel coordinates. As presented in Section 8.5 we only need to analyze a scatterplot, which visualizes the parameter

on one axis. However, we recommend the use of the composite parallel coordinates because the distribution of icons is easier to follow.

8.7 Evaluation with Domain Experts

Our approach is evaluated by three domain experts for geometry generators and evaluation systems for computer vision. The domain experts have to solve tasks similar to those described in the introduction. For the evaluation, we sample the cup generator with 100 cups using random sampling. We only evaluate the composite parallel coordinates and the hierarchical layouts. As presented in Chapter 8 the scatterplot matrix can also be used to solve the tasks. We recommend to use the composite parallel coordinates because this visualization is better suited for high dimensional parameters spaces (and is part of our main contribution).

At the beginning of the evaluation, the main functionality is presented, followed by a live demonstration. Then, the domain experts have to solve the user tasks. They can ask questions if something is ambiguous or not well traceable. After solving a task the domain experts report the degree of difficulty and guidance of using *Cupid*. If the task is not solved successfully, potential problems of the workflow are discussed to identify the weaknesses of our system. After solving all tasks, the domain experts grade the used techniques according to usefulness and usability.

User Tasks

Our evaluation focuses primarily on the workflows and not the used visualization techniques. For example, we are interested if the coloring of the clusters is sufficient to solve our tasks. The domain experts have to solve the following tasks:

User task 1: Evaluation of the parameter space and geometric result: The first task is to give an overview which groups of cups are generated. Then, the domain experts have to find the associated parameter regions. We test several tree layouts. Additionally, the domain experts have to explore the hierarchical clustering and rate the similarity of objects within a cluster.

User task 2: Finding implausible cups: At first, the user needs to find examples of physically implausible or unwanted cups. Physical implausible cups result from big handles or very small bodies, for example. Also meshes with errors occur (e.g., crossing faces of the surface). The domain experts have to find examples for both errors and the corresponding parameter values.

User task 3: Determine the influence and sensitivity of parameters: In Tasks 1 and 2 the generated shapes are in focus. Task 3 requires the analysis of the parameter space to determine the influence and sensitivity of a parameter. Three different parameters with varying influences are selected. The domain experts have to determine the effect of each parameter and sort the parameters according to their influence. The parameter with the largest influence is then used for a sensitivity analysis, i.e., the experts have to look for regions where the resulting shapes have a high or a low variation.

Moreover, we asked the experts about the difficulty to solve the tasks, and about the usefulness of the visualization respectively the interaction tools.

Feedback from Domain Expects

The domain experts like the combination of the parallel coordinates with the radial tree. One of them especially valued that *Cupid* allows to deal with a larger number of parameters at a time. Due to our approach, they can now detect interrelationships between parameters of the geometry generator rather quickly and identify sensitive parameter ranges, i.e., where a slight change has a large influence on the created 3D shape. During the evaluation, the domain experts use the radial tree to get an overview of the clusters, while the parallel coordinates are used for inspecting details. They see this combination as an intuitive way of exploring the geometry generator and rate this combination as excellent. Including the spatial information into the abstract representation of the parameter space supports the easy exploration of the generated shapes. All domain experts were able to assign the parameters to the resulting shape and vice versa.

The first user task of finding a group of similar shapes is easily solved by all domain experts. They explore the data set using the radial tree as described in Section 8.3. The domain experts detect several categories of shapes and inspect the corresponding parameter values with linking between the radial tree and the parallel coordinates. While the initial clustering presents a good starting point for the exploration, some cups were wrongly assigned. This is because our similarity metric works well for very similar shapes, but it has problems with less similar shapes. In such cases, the domain experts interactively modify the clustering using the merging and splitting tool. They rate the User Task 1 with a low degree of difficulty and a good support by *Cupid*.

While solving User Task 1, the domain experts find some examples of implausible cups (compare to User Task 2). For finding more implausible cups, they navigate through all clusters at the top or middle hierarchy level and inspect their members. This can be done fast because the cups within a cluster have similar characteristics. In contrast, the starplots and function plots depicting the derived geometric properties are not used often. One domain expert explains that the properties are not intuitive. After a discussion, the domain experts think that they are a useful extension for specific tasks. For example, finding physically implausible cups can be done with the shape stability property.

For evaluating the influence of a parameter on the result, the domain experts use the parallel coordinates. They look for clusters that only cover a small portion of the parameter range (see Section. 8.5). The effect of each parameter is explored using the line brush. The domain experts test different regions of the parameter space and analyze the variation of shapes in a detail window. The domain experts like this integration of geometry into the parallel coordinates and rate it as good. Furthermore, they use the function plots and tested several derived geometric properties. Since the domain experts do not find reasonable geometric properties, they select the properties by guessing. They rate the function plots as satisfactory. In summary, the domain experts rate the user task of evaluating the parameter influence with an average degree of difficulty.

The domain experts denote the analysis of the sensitivity as the most difficult user task. This task requires to analyze a combination of parameters because clusters can be affected by

several parameters (see Section 8.6). The domain experts explore various regions and study the variations using line brushes. Additionally, the clustering is also used for their sensitivity analysis. The domain experts identify several regions with different sensitivity. They rate this user task with a high degree of difficulty because the relationships between parameters and 3D shapes are difficult to understand. Furthermore, the domain experts ask for methods to test their own parameter combinations and to vary the sampling rate interactively. Testing their own parameter combinations would enable a new workflow to vary only one parameter and set other parameters to a default value. Varying the sampling interactively can be used to exclude uninteresting regions of the parameter space.

At the end of the evaluation, an interview about the techniques provided by *Cupid* is done. The domain experts appreciate the interactive techniques like linking and brushing. The integration of clustering into the parallel coordinates using illustrative methods is also rated between good and excellent. The domain experts appreciate the possibilities provided by *Cupid*. They rate the integration of geometry between good and excellent and like the different types of icons. The methods using derived geometric properties are rated lower because the workflow is not so intuitive.

CHAPTER 9

Discussion

In the previous section, we present our application *Cupid* to analyze the parameter space of geometry generators, and we discuss some use cases (see Chapter 8) to solve the tasks introduced in Section 1.1. This chapter focuses on a critical reflection and open issues of *Cupid*.

In this chapter, we discuss *Cupid* in detail. First, we discuss our similarity-based clustering. In Section 9.2, we examine the composition of the used visualization techniques as well as their interplay. Then, we review the composite parallel coordinates and scatterplot matrices (see Section 9.3) as well as the different algorithms to display the hierarchical clustering (see Section 9.4). In Section 9.5, the icons are reflected on detail, and in Section 9.6 the geometric properties are treated. We present the results of the performance evaluation of *Cupid* in Section 9.7. In the last section, we conclude with the strengths and weaknesses of Cupid.

9.1 Similarity-based Clustering

The similarity-based clustering is one of the key elements of *Cupid*. The similarity-based clustering is built on a vertex-to-nearest-vertex similarity measurement and a density-based clustering. *Cupid* uses the vertex-to-nearest-vertex measurement to measure the similarity between two shapes. As discussed in Section 4.2, we search for each vertex of a shape the closest vertex of the other 3D shape. Then, we measure the Euclidean distance and calculate the mean of all distances. We use this approach because it is straightforward, and the measurement can detect small differences. In Figure 9.1, some examples of the differences between meshes are shown. If two shapes are similar, the similarity measurement results in a value between $0.0 - 0.01$. If the shapes are not similar, then our similarity measurement results in a value higher than 0.1.

However, this measurement also has some troubles with measuring the similarity between two geometric shapes. Figure 9.1c shows an example. The three shapes differ, in particular, in the convexity. Even through the target shape is slightly concave, the similarity measurement considers a somewhat convex cup to be more similar than a concave cup. The vertex differences between the target and the slightly convex shape are lower than the vertex differences between the target and the concave cup.

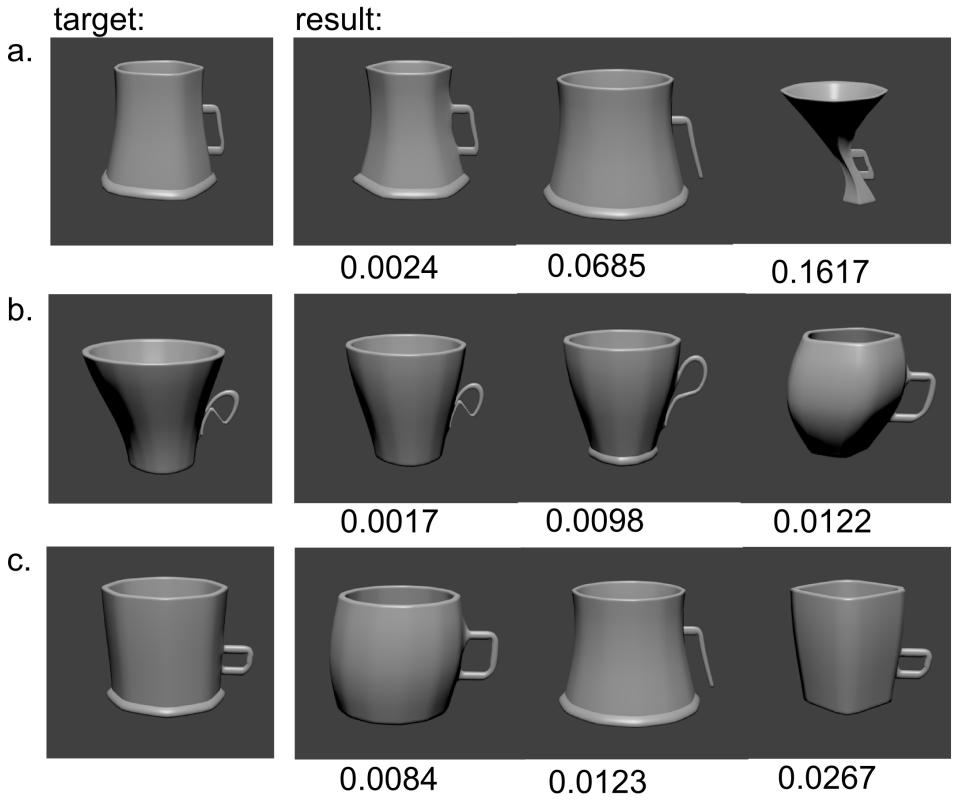

Figure 9.1: Three shapes and some examples of the corresponding similarity value.

This issue comes as no surprise since the vertex differences are not equal to the similarities between geometric shapes. An alternative approach is the use of feature-based techniques (presented in Section 3.8). At first, a set of features is calculated that can differentiate between 3D shapes. The features build up a shape descriptor, which is a multi-dimensional vector (called feature-vector). Then, the shape descriptors are compared using a metric, and a value of similarity between the two shapes is returned. The performance of the similarity measurement depends on the features and metric used. Finding a global solution that works for all different classes of shapes is still an open topic. In particular, the calculation of the similarity within a category of objects is still very complicated, and the user or an administrator often has to find a set of features for each application area. This task is very time-intensive because a great deal of knowledge about the parameter space is needed. However, *Cupid* should help to obtain such knowledge. An alternative approach would be the use of algorithms for the automatic selection of the features. Such techniques build up a feature descriptor from a list of available features. Implementing this method is very time-intensive because a lot of different features have to be calculated. Furthermore, the automatic selection of features is still an open problem. We prefer our simpler approach because we focus more on the visualization of the parameter set from geometry generators, and not to find a perfect solution for measuring the similarity.

After measuring the similarity of the objects, the shapes are clustered according to similarity using a hierarchical adaptation of the DBSCAN algorithm. All shapes, which are more similar than a user-defined threshold, are grouped together into a cluster. As a result, each geometric shape within the cluster is similar to at least one member of the cluster. We use this clustering algorithm because the number of clusters is not known beforehand (as opposed to k-mean's algorithms), and the shape of a cluster can be arbitrary (instead of EM-algorithms). Furthermore, different similarity measurements can be used, as well as different metrics. The extension to hierarchical clustering is straightforward.

In summary, our presented similarity-based clustering approach is not a perfect solution to cluster a set of shapes. As presented, our similarity-based clustering results in a good initial clustering, but the user needs to modify the outcome because misclassification can occur. If the user detects some false-classification, the hierarchical clustering can be changed quickly by using the splitting and merging tool.

9.2 Combination of Multi-Dimensional Visualizations, Hierarchical Layout, and Hierarchical Clustering

As presented in the previous chapters, we use composite parallel coordinates, a composite scatterplot matrix, and hierarchical layout to analyze the abstract parameter space of the geometry generator. Before we discuss all visualizations in detail, we analyze the composition of the different visualizations. The interplay between the visualizations is the focus of this section.

A critical challenge of this work is to visualize many 3D shapes. As presented in Chapter 4, we use a novel cluster-based approach to group similar shapes. The geometry generator can result in many clusters because the geometry generator creates many objects. We group the clusters with hierarchical clustering. First, *Cupid* shows an overview representation. Then, the user explores the details on demand (compare with zooming presented in Section 3.7). An animated transition preserves the mental map of the user. This technique reduces visual clutter because the hierarchical clustering reduces the number of objects to display (showing only 4-5 overview clusters instead of 30). The user interactively drills down the hierarchical clustering to explore all details of the geometry generator.

For color assignment, we also use hierarchical clustering. Each cluster of the top-level cluster gets its own color. As presented in Section 5.1, we use the color scheme of Colorbrewer [44]. They provide 12 colors that are well-suited to discriminate different elements. If the clustering results in more than 12 overview clusters, we use the same colors multiple times. More than 12 overview clusters is an exceptional case. We chose this approach because the colors are well-suited to differentiate. Using more than 12 colors is not useful because the human recognition of colors is limited.

However, if the user wants to compare different clusters with different (grand-) parent clusters, the user needs to drill down through several parent clusters. The composite parallel coordinates and the scatterplot matrix will present many clusters. The user could lose the mental map. To handle this problem, we use a novel combination of the visualization of the multi-dimensional parameter space and the representation of the hierarchical clustering. While the multi-dimensional visualizations depict the parameter space in detail, the hierarchical layout

gives an overview of the hierarchical clustering (compare with overview+detail presented in Section 3.7). To navigate through the visualizations, we link all visualizations (compare with the *juxtaposition view* presented in Section 3.6).

This combination also enables additional workflows. For example, the user can fade-in the B-splines in the composite parallel coordinates. Using the highlighting tool, the user can select the corresponding B-splines in the hierarchical clustering view. This feature can be used to explore the hierarchical clustering. The advantage of this approach is that only some B-splines are added to the visualizations instead of many colored polygons. This feature is very useful if the user wants to compare clusters with different (grand-) parent clusters.

Another advantage of our approach is that the user has visualizations, which are well-suited for the data. While the radial tree, dendrogram, and treemap are well-suited for analyzing the hierarchical clustering, the composite parallel coordinates and the scatterplot matrix are helpful for analyzing the abstract parameter space. As presented in Chapter 8, the visualization of the hierarchical layout is well-suited for Tasks 1 and 2. The visualization of the hierarchical clustering is useful for Task 3.

As presented in Chapter 7, *Cupid* provides a modular design. The user can select between different visualization techniques like composite parallel coordinates, composite scatterplot matrix, and radial tree. The user can modify many different properties. For example, the icons in the composite parallel coordinates can be hidden, and the radial tree can be used for the exploration of the geometry generator. *Cupid* can be adjusted to the needs of the user and the application area.

In conclusion, our combination of different visualization techniques simplifies the exploration of the geometry generator's parameter space. The hierarchical clustering reduces the number of objects to display. The used visualization techniques are well-suited for analyzing the parameter space. Using linking, the user can navigate through the different visualizations quickly.

9.3 Visualization of the Multi-Dimensional Parameter Space

The parameter space is visualized using composite parallel coordinates or the composite scatterplot matrix. Both methods are common visualization techniques to visualize the multi-dimensional data. As presented in Section 3.1, these techniques are examples of geometrically transformed displays, which trys to find a meaningful projection from the multi-dimensional space to the 2D screen.

Alternative approaches use dimension reducing methods, like the principal component analysis (PCA) and the multi-dimensional scaling (MDS) algorithm. Such algorithms reduce the high-dimensional input space to the low-dimensional visual space. For example, the PCA algorithm reduces the dimensions with linear transformations and tries to preserve the target distances of the parameter space. The MDS algorithm preserves the global distances (e.g., similarity) and is used by Design Galleries [68]. Talton et al. [93] extend the MDS approach to generate a semantic map. The main advantage of such an approach is that very high-dimensional spaces can be visualized. In contrast to the scatterplot matrix and parallel coordinates, such methods can be used to visualize highly dimensional data. Scatterplot matrix and parallel coordinates re-

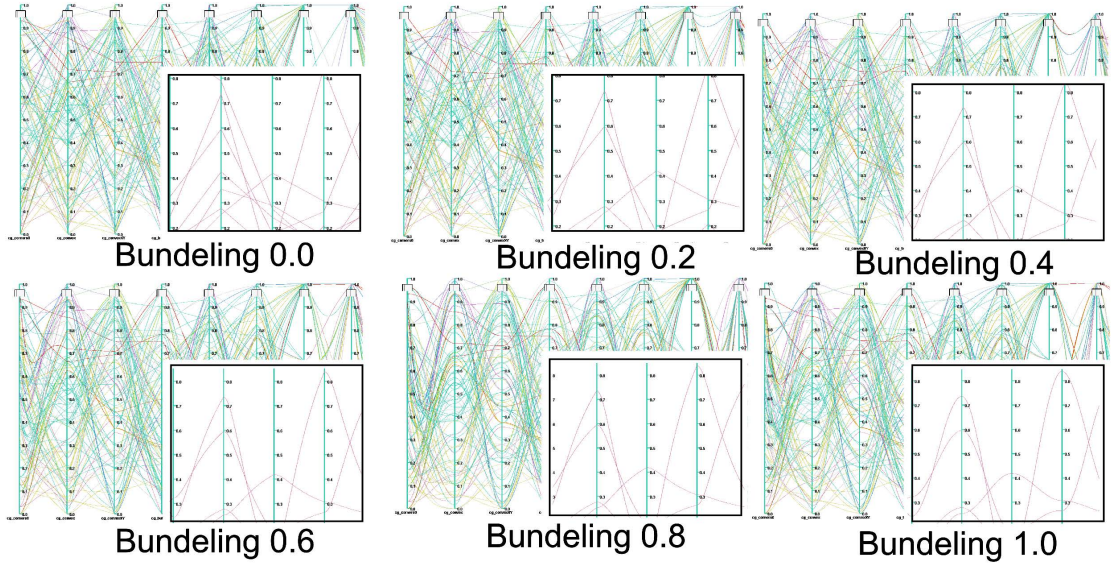

Figure 9.2: Effect of edge bundling without clustering. A bundling value of 0 results in a visualization of the line paths. Higher bundling values result in round B-splines.

sults in too complex visualization. Dimension reduction methods do not represent the parameter space completely. For example, the PCA method combines several parameters in a new parameter. However, *Cupid* has been developed to analyze the complex relationships between the parameters and the resulting 3D shapes. The reproduction of the multi-dimensional parameter space is very important. Our composite multi-dimensional visualizations represent the multi-dimensional parameter space in a sufficient way. They are very suitable for analyzing the tasks, presented in Section 1.1.

Besides showing of the parameter space using a multi-dimensional visualization approach, another methods is to employ a template object, respectively, a representative. The user can modify the template object (respectively, a representative) and the corresponding shape is shown, i.e., the most similar shape to the template object. The geometric shape is in focus, and the user can analyze all details of the shape. Such visualization is suitable, if the user wants to query a specific representation. Additionally, abstract data is visualized using linked visualizations, like starplots. The main advantage of such visualizations is, that the user can explore the parameter space in a goal-oriented way. If the user is interested in a special variation of a part of the shape, e.g., the handle of the cup, he only deforms the part and all variations are shown (i.e., different variations of handles). This workflow is called inverse-design. This workflow is helpful for the exploration of shapes, especially, if the user looks for a special shape. *Cupid* does not support a similar workflow. The user has to look for shapes using the parameter space. However, exploration of a set of geometric shapes using a geometry generator is useful for our task only to a certain limit. We want to compare different shapes for categorization. A representation showing several shapes is missed. The other tasks presented in Section 1.1 need also some

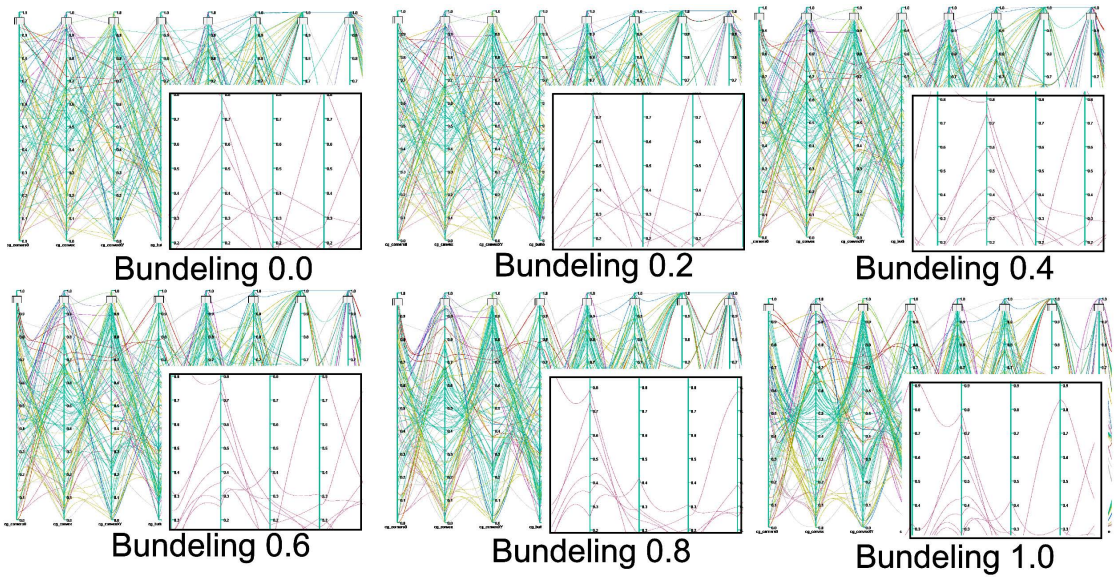

Figure 9.3: Effect of edge bundling with clustering. Higher bundling values result in bundling of B-splines of the same cluster. (compare it with Figure 9.2)

representation of the set of geometric shapes. Projection-based techniques are better suited for our purposes.

In summery, projection-based visualization technique like parallel coordinates and the scatterplot matrix are best suited for our purpose. In the next sections, the parallel coordinates, and scatterplot matrix are discussed in more detail.

Composite Parallel Coordinates

Parallel Coordinates are a powerful visualization technique to represent a multi-dimensional parameter space.

Parallel Coordinates display a set of n-dimensions with n-parallel straight lines. The straight lines are equally spaced and are called axes. A datum in the n-dimensional space is represented as a polyline with a vertex at each parallel axis. The position on each axis corresponds to the value in the associated dimension. As shown in Figure 9.2, this representation results in a lot of visual clutter between two coordinates, which makes it difficult to read. Our data are too complex for the standard technique. To reduce visual clutter, we use edge bundling as presented in Section 5.1. In Figure 9.3, different values of the bundling parameter β are shown. When comparing it to Figure 9.2, the B-spline are clustered in Figure 9.3. The bundling reduces the overlapping of line paths, if the B-splines are clustered and the B-splines are clustered. The representation of the clustering is easier to read because line paths of the same cluster are bundled together. Figure 9.4 shows an example of different values of the bundling parameter β. We recommend a bundling value between 0.8 and 1.0. However, the user can modify the edge bundling interactively.

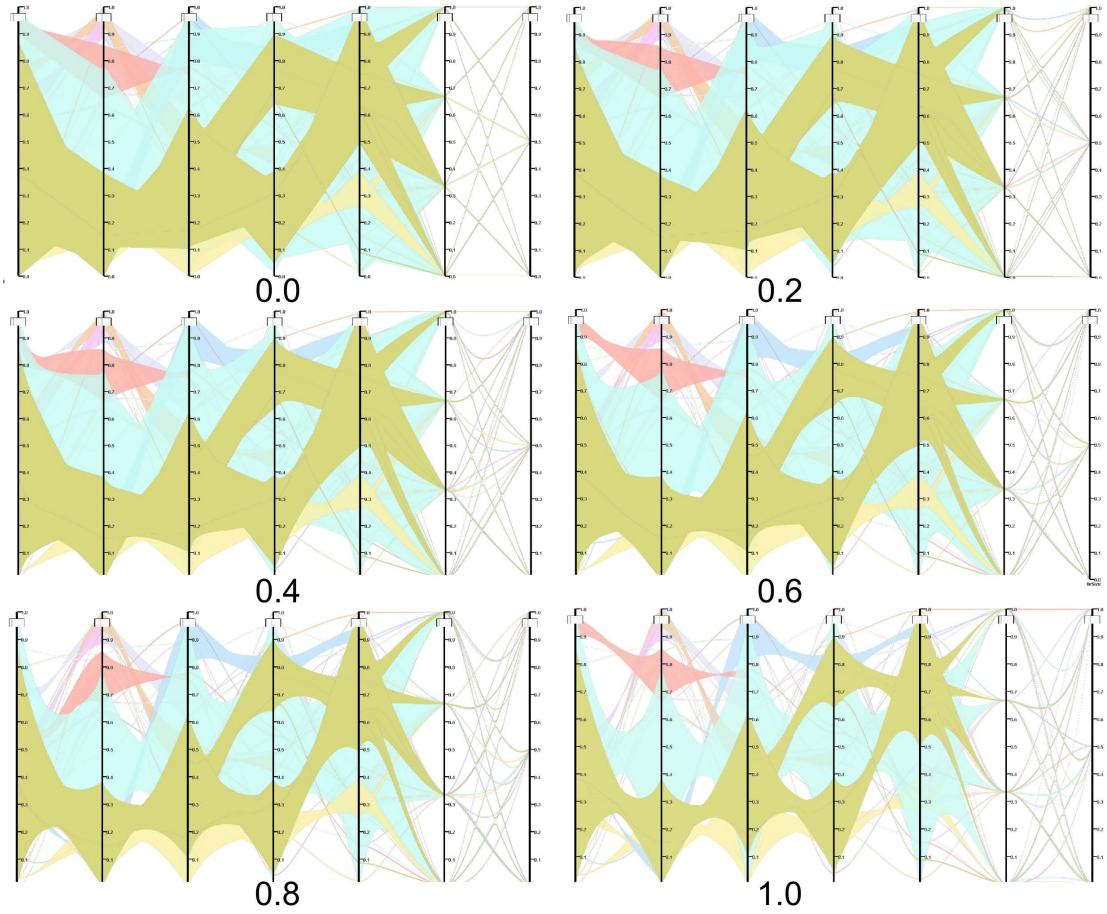

Figure 9.4: Edge bundling: Effect of B-splines on the colored polygons.

In parallel coordinates, the clusters are usually displayed by coloring of the line paths. First, a unique color is assigned to each cluster. Then, the line paths are colored with the cluster's color. *Cupid* also supports this technique. However, this technique is difficult to read, if the multi-dimensional data is complex. As shown in Figure 9.2 a lot of line crossings occur. To reduce the visual clutter, *Cupid* provides a level of detail representation using colored polygons. Each polygon shows the extent of the clusters. The user can define the level of detail to display. In Figure 9.5 an example with different levels of detail is shown. The highest level of detail results in the representation of the line paths and the lowest level of detail in the representation of the extent of the clusters. As shown in Figure 9.5, lower levels of detail reduce visual clutter. Using an intermediate level of detail gives enough details but does not result in visual clutter.

Cupid allows to display the colored polygons with different styles. The standard representation is to draw colored polygons. Alternatives are the additional representation of the contour, shadow, and contour with shadow. The motivation of providing different styles is to enable an easier distinction between clusters, and to highlight the selected cluster. In Figure 9.6 an example of the parallel coordinates view with different styles is shown. The colored polygons

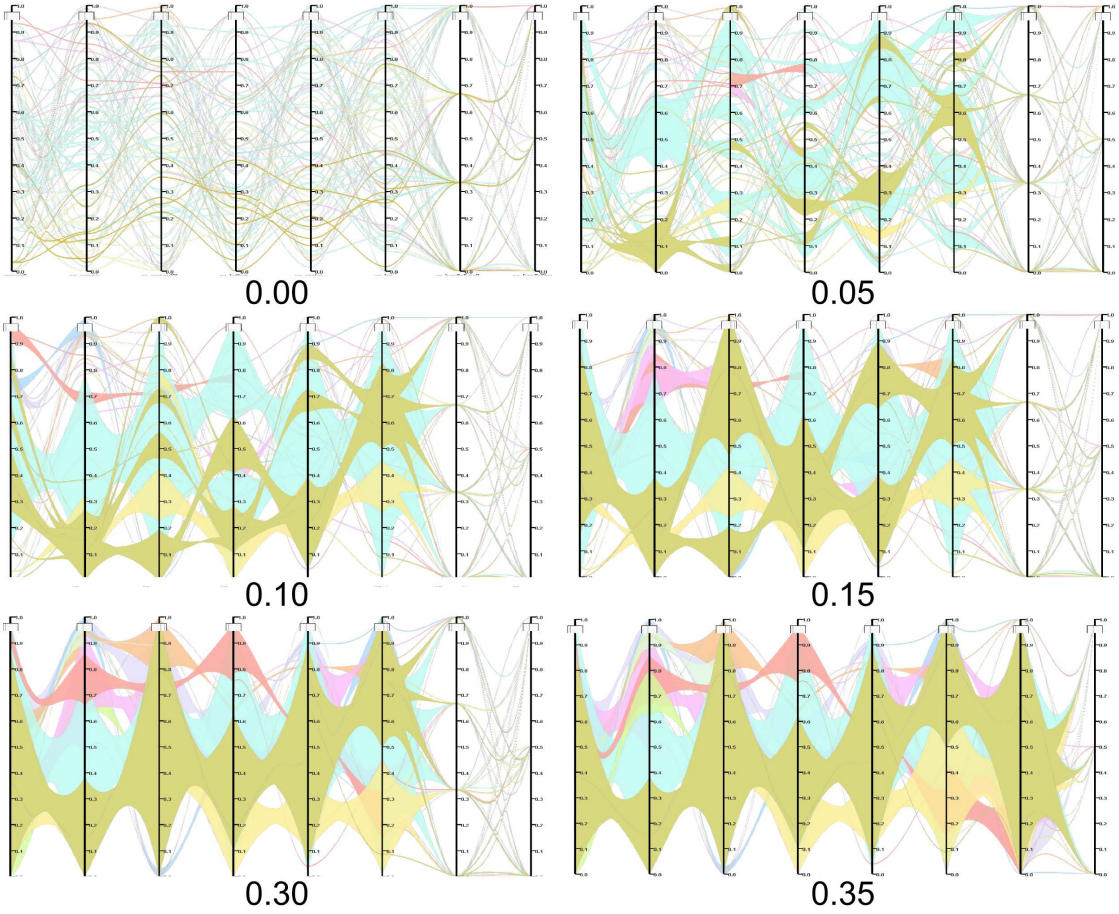

Figure 9.5: Level of detail (LOD): While a tiny branching width results in a visualization of the B-splines (high level-of-detail), a high branching width results in a visualization of the extend of the cluster (low level-of-detail).

can be better distinguished, if the contour or shadow is used. However, the different styles can also result in visual clutter, especially, if a lot of details are shown in the polygons. The more complex styles add additional elements to the representation. We use the more complex styles primarily to highlight the selected cluster. Our default settings are standard colored polygons for all non-selected clusters, and the contour style for the selected cluster.

Moreover, the user can modify the opacity and the saturation of the colored polygons. The effect of modifying the colored polygon's opacity and saturation is shown in Figure 9.7. Both parameters enables to highlight the selected cluster.

If the user drills down a cluster, an animated transition is performed. The animated transition morphs the parent cluster to the child clusters. This technique preserves the mental map of the user. Only successive small changes are shown, instead of showing all changes instantly. This technique is useful, if the cluster is only split in less than five child elements. If a drill down

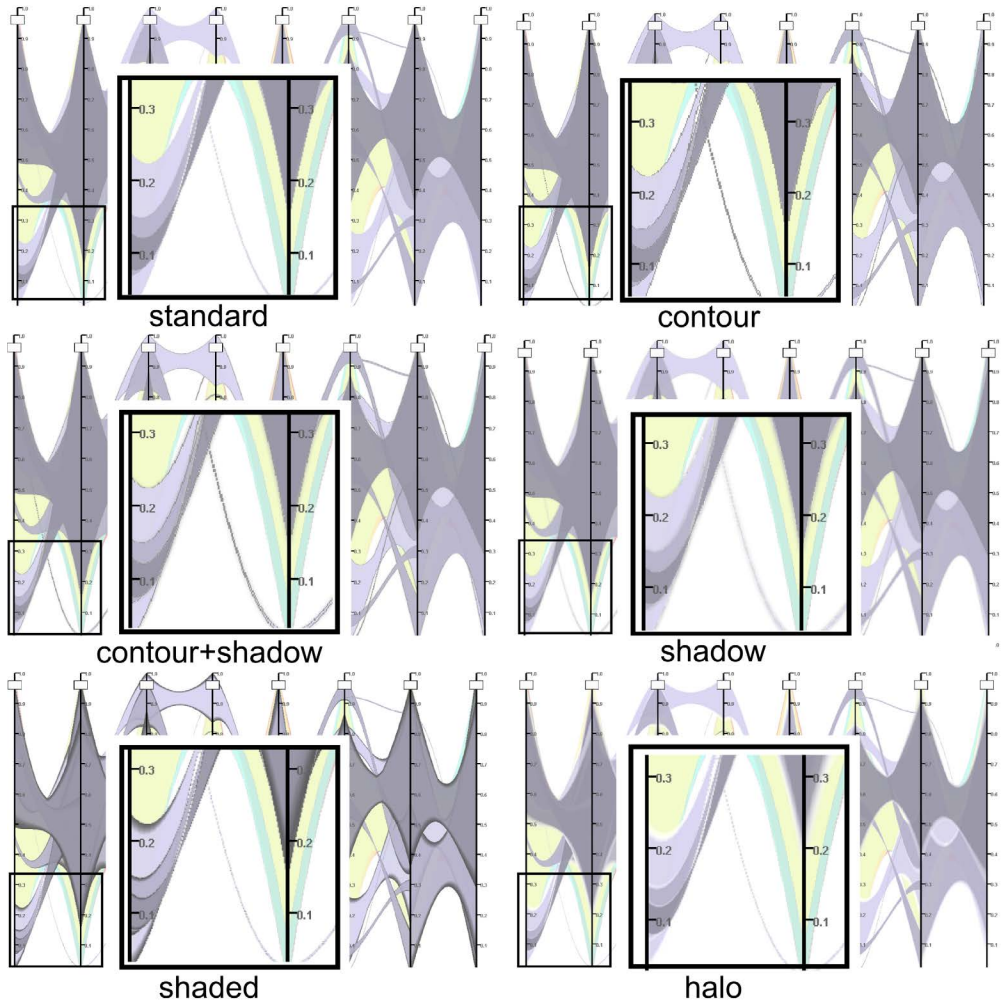

Figure 9.6: Different styles of the colored polygons.

results in more than five child clusters, it is difficult to follow the animated transition because too many changes occur simultaneously. To handle this issue, the user can control the speed of the animated transition. Optionally, the user can also navigate through the animated transition by selecting a specific timestep using a slider. We conclude, that the animated transition helps to preserve the mental map, but too many changes can confuse the user.

As presented in Section 8.2, selecting a combination of parameters is very cumbersome with the composite parallel coordinates. The user needs to mirror a parameter, and applying a bundling factor of 0. A similar task is to analyze two parameters concerning correlations. This task is also very difficult to analyze with the parallel coordinates. As evaluated by Harrison et al. [43] correlations can be easier analyzed with a scatterplot. Thus, we recommend the use of our composite scatterplot matrix for such tasks.

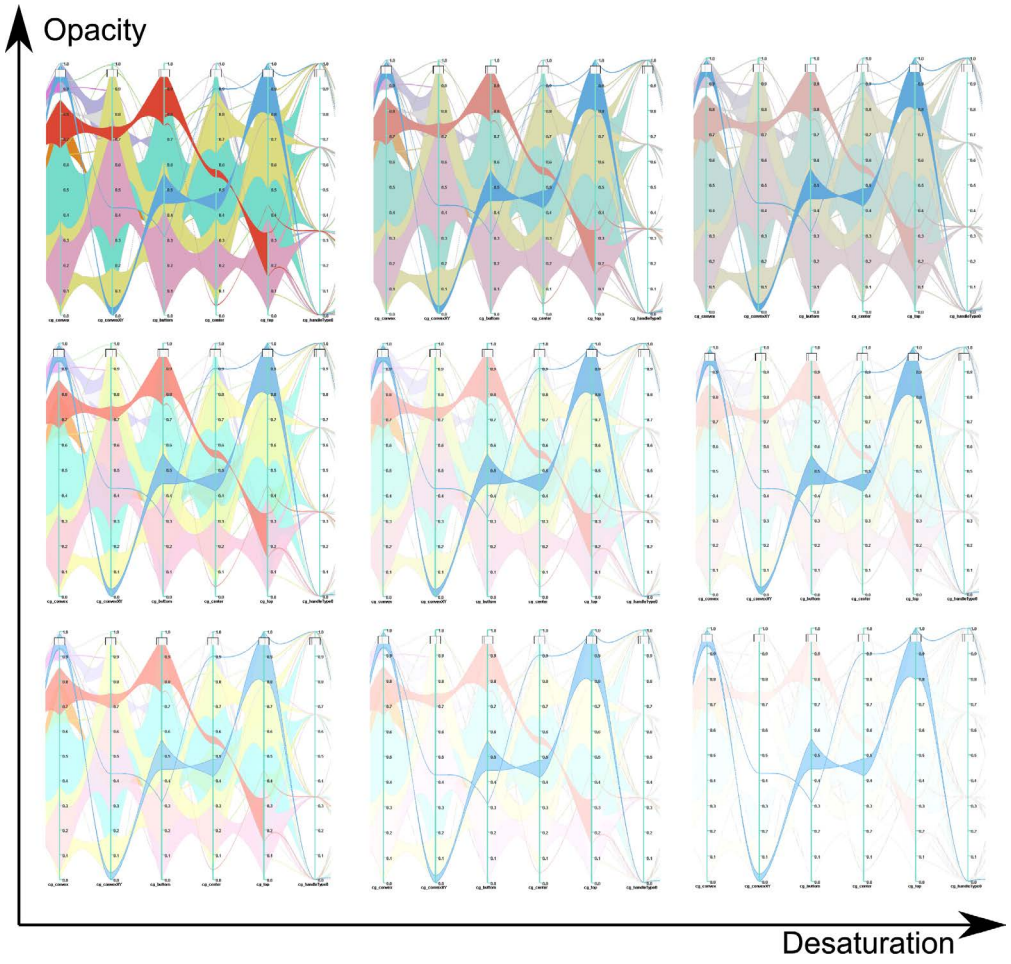

Figure 9.7: Variation of the opacity and saturation of colored polygons

Composite Scatterplot Matrix

Besides composite parallel coordinates, *Cupid* provides a scatterplot matrix to visualize the multi-dimensional parameter space.

The main advantage of the composite scatterplot matrix is that this representation is easy to understand. Scatterplots are a common visualization technique to display two parameters. For instance, Piringer et al. [76] use a scatterplot for displaying a parameter space of 2D function ensembles. Busking et al. [24] use this technique to display a set of shapes. If more than two parameters have to be displayed, scatterplot matrices are a commonly used technique to visualize multi-dimensional data. The matrix arrangement enables to analyze the entire parameter space easily. Another advantage of a scatterplot matrix is that all combinations of parameters are shown in one plot. The scatterplot matrix can be used to compare one or more parameter to all other parameters.

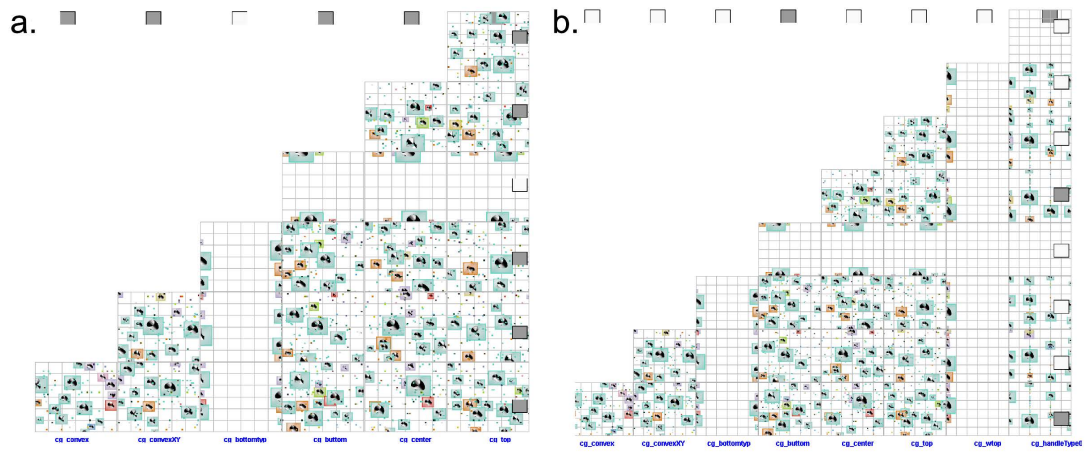

Figure 9.8: Visualization of many parameters: (a) Visualization of seven parameters, (b) Visualization of nine parameters.

The main disadvantage of the scatterplot matrix is the inefficient use of screen space. Displaying each combination of more than five parameters results in tiny scatterplots. Figure 9.8 shows two scatterplot matrices with too many parameters. All scatterplots in the matrix are tiny. However, if not every combination of parameters is in focus, the user can hide some scatterplots. Showing only some coordinates results in more space efficient representation. Additionally, *Cupid* provides the possibility to enlarge columns and rows. If the user enlarges a column (or row), the *Cupid* downsizes the other columns (respectively rows).

In the composite parallel coordinates, we provide several techniques to enhance the readability. The illustrative techniques reduce visual clutter, and the animated transition preserves the mental map of the user during a drill-down operation of a cluster. Our composite scatterplot matrix does not provide similar visualization techniques. Advanced visualization techniques would be very desirable. The correspondence between icons and markers is only visually encoded using the coloring. The markers have the same color as the icons. To differentiate between a lot of different colors is very difficult for the human recognition. If the user drills down a cluster of interest, he has to find the changes in the icons. The radial tree is very useful for this task. However, animating the changes of a drill-down operation would enhance the user experience.

In conclusion, the scatterplot matrix is suitable for comparing only small numbers of parameters. This technique is commonly used, if more than five parameters are shown, the scatterplot matrix is difficult to understand. In such cases, we recommend the use of the parallel coordinates approach.

9.4 Hierarchical Layout

Cupid represents hierarchical clustering with hierarchical layouts. As presented in Section 5.3, *Cupid* provides a circular dendrogram, a radial tree layout, and a treemap layout.

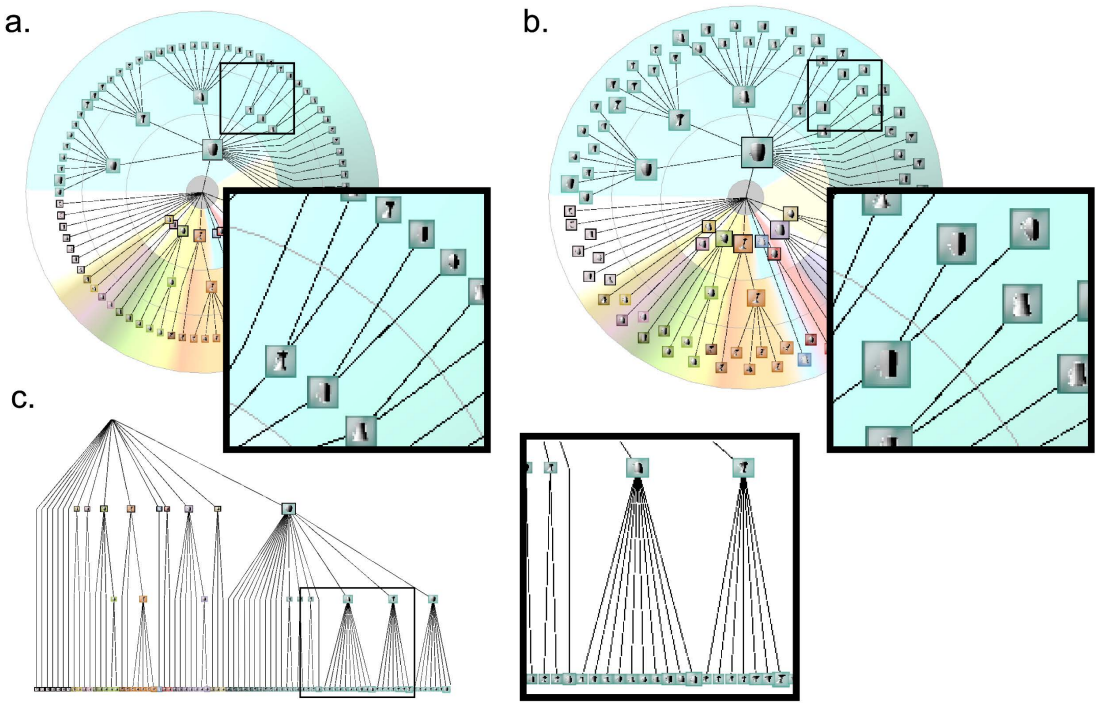

Figure 9.9: Without shifting (a) and with shifting (b) of the overlapping icons. Using shifting results in a better use of space. (c) The corresponding tree (without transformation)

The circular dendrogram and the radial tree layout are common examples of node-link diagrams. As discussed in Section 5.3, the main difference between these layouts is that the intermediate nodes are not drawn. For example, in the circular dendrogram, the relationship between the leaf elements is only shown using hierarchical lines. Circular dendrograms are commonly used to visualize hierarchical clustering because the intermediate nodes can result in visual clutter. However, the hierarchical clustering only results in a tree with a height of four and less intermediate nodes. So, drawing the intermediate nodes does not lead to visual clutter.

The circular dendrogram and radial tree result from an additional transformation of the tree (dendrogram) using polar coordinates. We prefer this transformation because the outcome makes a better use of space. Our hierarchical structure has four levels and few intermediate nodes, but many leaf nodes, compared to the number of leafs. The transformation to polar coordinates enables us to draw more leaf elements of the same size than when using the common method. Additionally, *Cupid* checks for overlapping icons. If two icons overlap, one icon is shifted. This technique ensures a better use of space because the icons can be drawn larger (see Figure 9.9a, b).

Besides the two node-link diagrams, *Cupid* provides a treemap layout to visualize the hierarchical clustering. The main advantage of using treemaps is the best use of space, but the hierarchy is harder to read. As discussed in Section 5.3, *Cupid* uses the strip layout as a tilling

algorithm. As explained in Section 3.5 other tilling algorithm, like slice-and-dice or squarified, exist with different strengths and weaknesses. The strip tilling algorithm results in an ordered treemap, with proper aspect ratios and high stability. As shown in Figure 5.8c, the aspect ratio of the resulting treemap is just as satisfying. Some rectangles have a very bad aspect ratio, especially the dark orange/brown rectangles and the gray rectangles represent outliers. This results in small icons, because they are quadratic. The ordering of the nodes is preserved. The rectangles with the same parent cluster are placed side by side. Moreover, the user can modify the ordering using the drag tool. In future work, we will add some attributes to the ordering, e.g., similarity between the clusters at the top level. Furthermore, the ordering in the treemap is equal to the node-link layouts. Using the same ordering in different layouts preserves the mental map of the user.

The comparison of the different tree layout methods does not result in a best representation. While the radial tree and circular dendrogram enable a proper representation of the hierarchical structure, the treemap uses the display space more efficiently.

In Section 3.5, a number of other algorithms are discussed for the representation of hierarchical data. Layered diagrams like sunburst diagrams or icicles, provide a more space efficient layout than node-link diagrams. As shown in Figure 3.9, the rectangles of layered diagrams have a very low aspect ratio. While a low aspect ratio is well suited for drawing labels, our icons need an aspect ratio of around one (the icons are quadratic). So, the use of layered diagrams is less suitable for *Cupid*. Circular treemaps are a variation of treemaps, and - as presented in Section 3.5 - circles are used instead of rectangles. This representation has the advantage of being easier to read. However, the representation uses the display space inefficiently. The provided node-link diagrams produce a more readable layout, and the treemap, a more space-efficient representation. Furthermore, in the other representations, rectangular icons are used. The use of circles instead of rectangles would reduce the consistency between the visualizations.

In summary, the selected representation is a proper selection from the recent state-of-the-art. The hierarchical layout is primarily used to display the hierarchical clustering. Additionally, the hierarchical layout is linked to the parallel coordinates and scatterplot matrix. The combination of the multi-dimensional representation and the hierarchical layout follows the idea of overview+detail. While the multi-dimensional representation shows the details of the parameter space, the hierarchical layout provides an overview representation. The hierarchical layout indicates which icons are currently shown in the parallel coordinates. If the user loses focus, the hierarchical layout helps to understand the current visualization. Furthermore, some tasks are easier to achieve using the hierarchical clustering (Task 1 and Task 2) while the third task can be achieved using the composite parallel coordinates or the scatterplot matrix.

9.5 Nested Icons

Cupid nests the geometric representation using icons. Usually, shaded geometry is used to visualize the shapes. *Cupid* uses Phong shading and the Phong reflection model for drawing the shaded geometry. This technique results in a smooth representation of the shaded geometry with low hardware requirements. Additionally, *Cupid* provides three alternatives to shaded geometry.

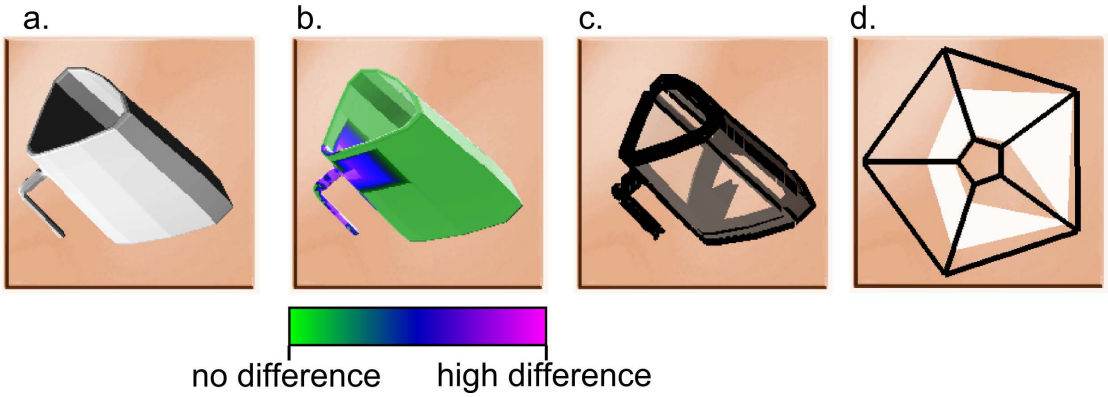

Figure 9.10: An example of different icons: (a) Standard geometry (b) Variations within the cluster (only the handle varies within the cluster), (c) Illustration of the silhouette (d) Corresponding geometry properties of the cluster

Comparing a set of shapes can be a time-intensive task. The user has to compare several shapes manually at different parameter positions. To support the user, *Cupid* we encode the variance between the nearest vertices of the cluster members to the cluster representative using a color map. This technique enables one to see constant and varying regions of the 3D shapes within the cluster without analyzing individual members of the cluster. Figure 9.10 shows an example. The members of this cluster only vary at the handles. The handle is colored in blue and pink, while the body of the cup is green. The pink color represents varying regions and the green color shows constant areas. The blue color represents small varying regions. The user sees varying areas quickly.

Another issue using shaded geometry is the representation of small icons. Especially in the scatterplot matrix, the icons can be tiny if many parameters are analyzed. As an alternative representation to the shaded geometry, *Cupid* offers an illustration of the object's silhouette. Following the idea of iWires [40], the sharp edges of the geometry are drawn to display just enough information to identify the 3D shape. Additionally, the shaded geometry is shown with low opacity. Giving only the sharp edges simplifies the recognition of small icons because only the necessary details are depicted. In Figure 9.11, a comparison between the two representations at different sizes is shown. The silhouette enables the recognition of smaller icons.

Finally, *Cupid* can depict derived geometric properties as starplots. These are created analogous to parallel coordinates but with polar coordinates. To ensure good readability, the plot only shows the convex hull of the line paths. They enable the user to find implausible cups quickly, because starplots are well suited for showing outliers. For example, a low stability value indicates an implausible cup (see Section 6 for more information). The starplots enable the display of three to eight parameters. If *Cupid* visualizes more than eight parameters, the visualization is hard to read.

The selected icons are the result of comparing different visualizations. As an example, we tried a visualization for the variations in a cluster by plotting a contour of each shape. This technique was presented by Busking et al. [24]. Regions where several contours are shown

represent variations of the cluster. The main disadvantage of this technique is that the contour slice represents only a small part of the shape. We also try to blend the members of the cluster on top of each other. This method produces results, where constant regions are drawn with high opacity and variations with less opacity. Similar to the contour visualization, a shape of the cluster can be difficult to analyze. Using our simpler techniques provides a visualization, which is easier to read. As an alternative to the starplots, we also test small multiples of the bar chart. For each member of the cluster, we show one bar of the bar chart. Each bar chart encodes one geometric property of the shape. Additionally, we use color coding of the luminance of the bar chart according to the value. *Cupid* represents the bar chart from the top. The color coding enables to identify the geometric properties. This representation is similar to dense pixel displays. The user can interactively change the camera position to see the histograms. This representation is much harder to read than the starplots, especially if the cluster contains many members.

Icons are used to represent a cluster. To analyze the members of the clusters, *Cupid* provides detail windows. If the user selects shaded geometry as icons, all members of the cluster are depicted side-by-side. If the user selects the sharp edge mode, a similar representation is given. Only the shaded geometry is replaced by the sharp edge representation. The display of the shapes side-by-side follows the idea of small multiples. As discussed in Section 3.2, small multiples are an excellent technique to display many shapes without confusing the reader. The main disadvantage of this representation is that having too many objects reduces the readability of the shapes.

If the user wants to analyze the geometric properties in detail, a starplot visualization is also given in the detail windows. As presented in Section 5.4, the starplot is displayed over the entire detail window. Using linking and brushing enables an interactive selection of shapes, depending on the geometric property. As presented in Section 8.4, linking and brushing is an excellent tool for selection of invalid or unwanted shapes.

In summary, the different representations of the shaded geometry improve the visualization of the nested shapes. While analyzing, the variation enables a quick comparison between members of a cluster. The sharp edges approach supports the visualization of tiny icons. The user can select the styles at run time. All presented techniques are a good trade-off between the complexity (the amount of data to display) and readability.

9.6 Geometric Properties

Cupid provides a set of geometric properties which are derived from the geometric shapes. As seen in Section 6.1, some general properties are calculated, like surface area and volume, and some properties for a particular task, like the shape stability measure. Additionally, we perform a simulation to determine the stability of the cups. The user can employ geometric shapes to find invalid shapes (e.g., with the properties: Euler characteristic and number of crossing faces) and physically implausible shapes (e.g., with the stability property and the tip-over test). However, the set of geometric properties is rather limited. For other application areas, the set of derived geometric properties has to be extended.

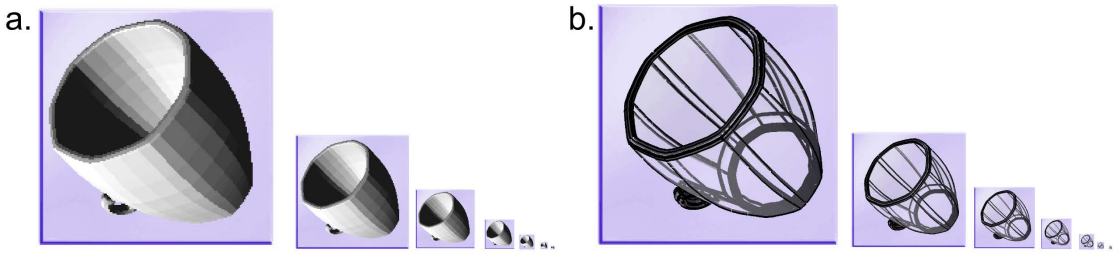

Figure 9.11: Comparision between shaded geometry and sharp edges with different sizes: While, (a) show the shape rendered with Phong Shading, (b) is rendered using sharp edges. The silhouette can be easier recognized using the sharp edges.

Besides the visualization of the parameter space and the shapes using novel composite multi-dimensional visualizations, *Cupid* also provides techniques to visualize the geometric properties. The user can choose between different visualization techniques, like starplots and function plots. Providing different techniques reduces the limitations of a single method.

The starplot icon shows the geometric properties of all objects within a cluster. In section 9.5, we discuss the starplot icon in detail. This method is appropriate if the user wants to explore the variation of the geometric properties. A disadvantage of the starplot is that the representation does not directly relate to the data (compared with icon-based displays in Section 3.1 and Keim [59]). The users have some problems with understanding the geometric properties (see Section 8.7). Furthermore, if the user wants to compare the geometric properties with the parameter space, this technique is difficult to use. Therefore, *Cupid* provides other techniques which focus on this task.

The function plots are well suited if the user wants to find a relationship between a parameter and a geometric property. Finding a relationship between a parameter and a geometric property gives information about the effect of a parameter (see Section 8.1), and the influence of a parameter (see Section 8.5). The relationship can be linear, quadratic, cubic, or another function. If no correlation is shown (the function plot shows some noise), then the parameter does not depend on the geometric property. Additionally, the linear regression between the parameter and the geometric property is displayed in the function plots to support the user in finding a relationship.

The main difficulty is to select the appropriate geometric property. A common method of doing so is guessing if the user has no information about the effect of a parameter. However, if the parameter is labeled and a description of the parameter is given, the user can take this information to select a suitable geometric property. Note that the geometric properties are derived from the geometric shapes. A geometric property can be influenced by other parameters of the parameter space. The effects of other parameters result in noise. For example, the parameter convexity side changes the surface area, but other parameters also lead to a change of this property (e.g. handle type or convexity top). As shown in Figure 6.3, the functionplot displays a relationship between the convexity side and surface area, but some noise also results from other parameters.

In the composite parallel coordinates, the geometric properties can be mapped to the B-spline's user-defined color and opacity using a transfer function. We apply this technique mainly

to highlight invalid parameter combinations. For instance, a low stability measure shows invalid shapes because the cup would tip over in reality. The user can look for agglomerations of invalid B-splines to find invalid parameter regions or parameter combinations. The user can visualize the geometric properties of all shapes of the parameter space. Rendering many B-splines can result in visual clutter. However, the user can also select a subset of the B-splines with the icons or the line brush tool.

In the composite scatterplot matrix, a similar representation show the geometric properties in the background, as presented in Section 6.3. This representation is well-suited to visualize invalid parameter combinations because it only displays a subset of the parameter space. The parallel coordinates technique is better suited for this task. However, the scatterplot technique can be used to find a correlation between parameters and geometric properties.

In conclusion, each of the visualization techniques is well-suited for a particular task. The main disadvantage of the geometric properties is that the user needs some knowledge about how to apply of each method. Even the domain experts have some problems with the use of geometric properties, as discussed in section 8.7. In future work, we will evaluate the visualization of the geometric properties to find an easier depiction, and we will extend the set of geometric properties.

9.7 Performance

We implement *Cupid* in *VolumeShop* [20], which is a 3D visualization framework and rapid-prototyping toolkit. The provided visualizations are implemented using C++ and OpenGL. *Cupid* requires a GPU that supports at least OpenGL 3.3. For the evaluation of the performance of *Cupid*, an Intel Core 2 Duo processor with 2.26 GHz, 6 GB main memory, and an AMD Radeon HD 3400 mobile graphic processor running Windows 8, is used. This configuration is the minimal system requirement for running *Cupid*. This combination results in around ten frames-per-second (fps). We also test *Cupid* using an Intel Core i5 processor with 3.3 GHz, 8 GB main memory, a Nvidia GT 440 graphics card, and Windows 7. This configuration gives us enough computational power to achieve interactive frame rates (>16 fps).

In summary, the system requirements for rendering the visualizations are medium. We do not speed up the application because interactive frame-rates are enough for our purpose. The primary bottleneck of *Cupid*'s visualizations is not the number of polygons to draw, but the high number of rendering calls.

The pre-processing pipeline is very time-consuming, especially, the measurement of the similarity. Despite the use of kD-trees and a fast implementation of the iterative closest point (ICP) algorithm, the similarity-based measurement needs at least three hours for 100 cups (with the Intel Core 2 Duo processor). *Cupid* measures each combination of the shapes and has to find for each vertex the closest vertex of the other 3D shape. In future work, we will speed up this task. The calculation of the derived geometric properties needs around one hour for 100 cups. The most time intensive task is the simulation and finding crossing faces. However, the pre-processing pipeline can be executed off-line.

9.8 Conclusion

The evaluation of *Cupid* shows that this system enables to solve the tasks presented in Section 1.1. Applying this novel combination of visualization techniques to depict the multi-dimensional parameter space allows the user to analyze its complexity. The hierarchical layout gives an excellent overview and helps the user to explore the set of geometric shapes.

However, *Cupid* has some limitations. One limitation is the simple similarity-based measurement, because it is very time intensive. Also, the quality of the similarity measure is a problem. Some miscalculations can occur. Using a faster algorithm would enable to provide an interactive steering of the geometry generator. The steering would allow for testing user-defined parameter combinations at run time.

Another limitation of *Cupid* is that high-dimensional parameter spaces are hard to evaluate. If more than 15 parameters have to be displayed with parallel coordinates, the visualization becomes very complicated to read. The limit for the scatterplot matrix is by approximately five parameters. So, the user has to select some of the parameters to visualize. If the geometry generator has more parameters, the whole parameter space of a geometry generator cannot be visualized in a sufficient way.

In comparison to the state of the art, where the applications can only solve one task, *Cupid* provides workflows to solve different tasks. Several workflows to solve various tasks, are presented in Chapter 8. The combination of different visualization techniques (like parallel coordinates, starplots and radial trees), and different data (parameter space, 3D shapes, and geometric properties) enables to get new insights in the parameter space of a geometry generator.

Our solution uses well studied (visualization) techniques and combines them in a novel way. The evaluation with domain experts (see Section 8.7) reveals that the provided composite visualizations are useful for the tasks. *Cupid* is rated with a high degree, especially, the composite visualization, the representation of the clustering, and the provided manipulation tools, like the line brush tool. The domain experts see the demand for *Cupid*, and they want to use it in their workflow. The high interest in *Cupid* indicates that there is potential for future work.

CHAPTER 10

Summary and Future Work

This chapter summarizes the main contributions and concludes this thesis. Moreover, future perspectives of research are presented in the last section.

10.1 Summary

In this thesis we present a novel visualization approach for exploring the parameter space of a cup generator. Our general goal is to understand the complex relationships between parameter space and the resulting geometric shapes. In particular, we identify three tasks by analyzing the parameter space of a geometry generator:

- Find similar 3D shapes and the corresponding parameter settings
- Find errors and implausible 3D shapes
- Determine the sensitivity and influence of parameters on the result

After providing a detailed evaluation of the most relevant previous works, we conclude with the demand for a new visualization system to explore ta geometry generator's parameter space. Motivated by the tasks described above, we introduce a new approach called *Cupid* to efficiently analyze the result of a geometry generator. While other approaches only focus on one task, our new approach supports all three tasks.

We present novel composite parallel coordinates and scatterplot matrix visualizations to explore geometry generators. Our visualizations combine both the abstract parameter space of the generator and the resulting cups into one visualization. While the parameter space is visualized with well-known visualizations like parallel coordinates and a scatterplot matrix, the geometric shapes are nested into the visualization using icons. This combination allows the user to study the complex relations between both domains.

To reduce visual clutter and to find similar 3D shapes, *Cupid* provides hierarchical clustering. Our approach supports levels-of-detail by controlling the similarity of objects and the details of

the clustering. Our presented technique enables the user to drill down the hierarchy to explore all details. We use the cluster hierarchy for coloring. The hierarchical clustering can be modified using the merging and splitting tool. Additionally, the hierarchy of the clustering is visualized using linked hierarchical layout algorithms like radial trees, circular dendrograms, and treemaps. While the visualization of the multi-dimensional parameter space enables the user to explore all details of the parameter space, the radial tree layout gives an overview visualization of the parameter space.

The composite parallel coordinates use several illustrative techniques like edge bundling, a level-of-detail representation using colored polygons, and different cluster styles. Using illustrative techniques reduces the visual clutter to ensure better readability. The hierarchy can also be explored with the composite parallel coordinates. In order to preserve the user's mental map, an animated transition is provided that morphs a selected parent cluster into its sub-clusters.

For quickly finding implausible objects, we derived geometric properties, which are also helpful for a sensitivity analysis of the parameter regions and their degrees of influence. The geometric properties are visualized using a starplot icon and function plots. Additionally, *Cupid* enables us to map the geometric properties to the B-spline's coloring and opacity, which are specified by user-defined transfer functions. In the scatterplot matrix, the geometric properties can be visualized as a background color. Providing linking and brushing enables a result-driven exploration of the parameter space.

We also present how to use our novel system to solve parameter space exploration tasks. As an application example, we explore the results of a cup generator. The parameter space consists of 11 parameters like the number of corners, handle type, and convexity. Each of the presented tasks could be solved using the provided composite visualizations of the parameter space.

In Chapter 9, we discuss *Cupid*. The critical reflection shows the strengths and weaknesses of our presented approach. The combination of well-suited visualization techniques lets us analyze the relations of the parameters to get new insights. Additionally, domain experts in computer vision testing evaluate *Cupid* in Section 8.7. The response from the domain experts is positive. They see the demand for this tool and want to use it in their current workflow. However, we also detect some limitations. These will be the subject of future work.

10.2 Future Work

Future work will focus on a re-design of the pre-processing steps, especially, the similarity measurement. We use a simple similarity measure because we primarily concentrate on the visualization of the complex parameter space. We identify the time-intensive calculation and some problems with the similarity measurement as the biggest issues of our approach. A re-design of the similarity calculation should focus on solving this problem.

In order to re-design the pre-processing steps, other application areas will be evaluated, like an airplane or car generator. New application areas could require changes in the similarity measurement. In order to deal with other geometric shapes, we will include further parameters such as the fractal dimension of shapes (used to evaluate plants).

The quick response of the pre-processing pipeline (especially, the similarity measurement) would enable to provide visual steering. Visual steering provides new workflows for analyzing

the parameter space of geometry generators. For example, the influence of each parameter could be evaluated very quickly. Instead of studying the variations of the cluster members using the icons or the parameter regions using the line brush tool, visual steering enables a new workflow. The user only varies one parameter and sets other parameters to a default value. Then, the user only has to analyze the newly generated shapes, which are not influenced by any other parameter. Furthermore, the sampling rate of sensitive parameter regions could be increased to analyze such areas in more detail.

In order to address new domain areas, the visualization techniques have to be evaluated for new requirements. For example, the parameter space of another geometry generator could be larger, than the parameter space of the cup generator. Furthermore, to provide visual steering a tool for querying a set of parameter combinations has to implemented.

In this thesis, we focus on the visualization of a complex parameter space. However, a result-driven exploration of the set of geometric shapes using manipulation would provide new opportunities to analyze the geometry generator. This workflow would be very helpful, especially if the user wants to explore a set of shapes in a goal-oriented manner. Interactive manipulation of shapes enables to analyze the parameter space more intuitively.

APPENDIX A

Additional material

A.1 Components

In this section the input and output of all components are presented. First, the parameter set editor saves the parameter space using an XML file, and samples the geometry generator using a batch file. To ensure that the obj files can be read, we convert all float values to a unique representation. In the next step, the similarity is measured. In the last step, the geometric properties are calculated.

Parameter space

The parameter space is saved using an XML file. Each object is saved within an XML-Tag: cup. Within the tag each parameter is saved with its name, type, and value. An example is seen below:

```
<?xml version="1.0"?>
<cups>
<cup>
<cg_corners0 type="Integer">13</cg_corners0>
<cg_convex type="Float">0.224403</cg_convex>
<cg_convexXY type="Float">-0.0500809</cg_convexXY>
<cg_bottomtyp type="Integer">1</cg_bottomtyp>
<cg_buttom type="Float">0.588794</cg_buttom>
<cg_center type="Float">0.702902</cg_center>
<cg_top type="Float">0.651784</cg_top>
<cg_wtop type="Integer">0</cg_wtop>
<cg_handleType0 type="Integer">1</cg_handleType0>
<cg_handleSize type="Integer">3</cg_handleSize>
<cg_smoothing type="Integer">1</cg_smoothing>
```

```
<cg_pathname type="String">objs5/0.bobj</cg_pathname>
</cup>
<cup>
<cg_corners0 type="Integer">9</cg_corners0>
<cg_convex type="Float">0.546434</cg_convex>
<cg_convexXY type="Float">0.450655</cg_convexXY>
<cg_bottomtyp type="Integer">1</cg_bottomtyp>
<cg_buttom type="Float">0.298227</cg_buttom>
<cg_center type="Float">0.0191351</cg_center>
<cg_top type="Float">0.618091</cg_top>
<cg_wtop type="Integer">0</cg_wtop>
<cg_handleType0 type="Integer">0</cg_handleType0>
<cg_handleSize type="Integer">2</cg_handleSize>
<cg_smoothing type="Integer">1</cg_smoothing>
<cg_pathname type="String">objs5/1.bobj</cg_pathname>
</cup>
....
```

Sampling of the Geometry Generator

To call-up the geometry generator a batch file is created. For each parameter an environment variable is created with the corresponding value. Then, the geometry generator is called.

```
set cg_bottomtyp=1
set cg_buttom=0.588794
set cg_center=0.702902
...
"Blender\blender.exe" --python script.py

set cg_bottomtyp=1
set cg_buttom=0.298227
set cg_center=0.0191351
...
"Blender\blender.exe" --python script.py
...
```

The environment variables are always created automatically. The geometry generator can use the environment variables or the XML file (presented earlier).

Obj Format

To load geometric shapes, *Cupid* supports the Wavefront's OBJ files. As presented in chapter 7, this file type is human readable, and is supported by most 3D modeling software. An example is given in the following.

```
v 0.449602991342545 0.393326997756958 0.033215999603271
v 0.461780995130539 0.424383997917175 0.001642000046559
...
vt 0.001124234343443 0.459650239340432
vt 0.123934934348ß344 0.129239029020923
...
vn 0.000000000000000 1.000000000000000 0.000000000000000
vn 0.000000000000000 0.000000000000000 1.000000000000000
....
f 1/1/1 219/219/219 253/253/253
f 1/1/1 253/253/253 107/107/107
....
```

Similarity Measurement

The similarity measurement application loads two 3D shapes and calculates the similarity. The result is written to a text-file.

The command to start the measurement is

```
iterativeclosestpoint.exe 1.obj 2.obj
transf12.txt transf21.txt result1to2.txt result2to1.txt result.txt
```

where `iterativeclosestpoint.exe` is the names of the application, `0.obj`, `1.obj` are the name of the 3D shapes. `transf01.txt` saves the resulting transformation of the ICP algorithm (`transf21.txt` is the transformation of mesh2 to mesh1), `result1to2.txt` contains the similarity of mesh 1 to mesh 2 (`result2to1.txt` the similarity between mesh 2 and mesh 1), and result.txt contains the similarity measurment between `0.obj` and `1.obj`.

Geometric Properties

The geometric properties are calculated in an own application. To derive the geometric properties for a 3D shape, the command is as follows:

```
VolCalc.exe 1.bobj output_props.txt
```

The application calculates the properties, and appends the results in a text file as follows:

```
...
0.108215 0.0285488 4.61137 2 0.246393
```

After calculating the geometric properties for all 3D shapes, the XML file is created as follows:

```
CreateXMLStatistic.exe nameProps.txt outputProps.txt statsgeom.obs
```

In `nameProps.txt` the names of the derived parameter are saved, and in `outputProps.txt` the result for each cup.

The derived geometric properties are saved using an XML file. The geometric properties of a geometric shape are saved within an XML-element: Geometry. Each derived property gets an own tag, where the type and the result is saved. An example is seen below:

```
<?xml version="1.0"?>
<GeometryStatistic>
<Geometry>
<NumSelfCrossing type="Float">0</NumSelfCrossing>
<Area type="Float">4.61137</Area>
<Volume type="Float">0.108061</Volume>
<Extent type="Float">0.149362</Extent>
<EulerNumber type="Float">2</EulerNumber>
<Barycenter2Center type="Float">0.358527</Barycenter2Center>
<Convexity type="Float">0.246043</Convexity>
...
</Geometry>
<Geometry>
<NumSelfCrossing type="Float">0</NumSelfCrossing>
<Area type="Float">2.23552</Area>
<Volume type="Float">0.0365732</Volume>
<Extent type="Float">0.137836</Extent>
<EulerNumber type="Float">2</EulerNumber>
<Barycenter2Center type="Float">0.327175</Barycenter2Center>
<Convexity type="Float">0.282991</Convexity>
...
</Geometry>
...
</GeometryStatistic>
```

A.2 Properties of the Visualizations

We use a lot of different algorithms to visualize the parameter space. The algorithms can be controlled using properties. In the following, we give an overview of all parameters:

Parameter name	**Description**
Data:	
Parameter space	Defines the parameter space to visualize
Geometry	Defines the shapes to visualize
Geometric Properties	Defines the geometric properties to visualize
Geometric Properties	Defines the geometric properties to visualize
Cluster:	

Time	Actual Time for a user-controlled animation
Parameter Threshold	Level of detail of the colored polygons
Standard Style	Style of all colored polygon
Selected Style	Style of the selected colored polygon
Standard Opacity	Opacity of all colored polygon
Selected Opacity	Opacity of the selected colored polygon
Standard Saturation	Saturation of all colored polygon
Selected Saturation	Saturation of the selected colored polygon
Outlier Line Width	Line width of all lines
Standard Linewidth	Line width of the lines, which are not selected
Selected Linewidth	Line width of the lines, which are selected
Outlier Linewidth	Line width of outliers
Standard Linewidth Opacity	Opacity of the lines
Selected Linewidth Opacity	Opacity of the selected lines
Icon:	
Type	Type of icons
Size	Size of icons
Projection	Projection matrix of the integrated icons. Can be adjusted for modifying the camera to render to shapes.
View	View matrix of the integrated icons. Can be adjusted for modifying the camera to shapes.
World	World matrix of the integrated icons. Can be adjusted for modifying the camera to shapes.
Transferfunction	Transfer function to visualize the variability of shapes.
Line:	
Bundling	Amount of edge bundling
Standard Width	Width of B-splines
Selection Width	Width of selected B-splines
Standard Color	Color of B-splines
Selected Color	Color of selected B-spline
Transferfunction	Transfer function to visualize geometric properties
Parallel coordinates:	
Width	Width of the coordinate
Color	Color of the coordinate
Plot:	
Projection	Projection matrix of the parallel coordinates. Can be adjusted for zooming.
View	View matrix of the parallel coordinates. Can be adjusted for zooming.
World	World matrix of the parallel coordinates. Can be adjusted for zooming.

Degree of Curve	Degree of the regression function
Parameter:	
Parametername	Switch on/off the visiblity

Table A.1: Parameters of Parallel Coordinates

Parameter name	**Description**
Data:	
Parameter space	Defines the parameter space to visualize
Geometry	Defines the shapes to visualize
Geometric Properties	Defines the geometric properties to visualize
Icon:	
Type	Type of icons
Size	Size of icons
Projection	Projection matrix of the integrated icons.
View	View matrix of the integrated icons.
World	World matrix of the integrated icons.
Transfer function	Transfer function to visualize the variability of shapes.
Plot:	
Projection	Projection matrix of the scatterplot matrix. Can be adjusted for zooming.
View	View matrix of the scatterplot matrix. Can be adjusted for zooming.
World	World matrix of the scatterplot matrix. Can be adjusted for zooming.
Degree of Curve	Degree of the regression function
Polar coordinates	Polar transformation

Table A.2: Parameters of scatterplot matrix

Parameter name	**Description**
Data:	
Parameter space	Defines the parameter space to visualize
Geometry	Defines the shapes to visualize
Geometric Properties	Defines the geometric properties to visualize
Icon:	
Type	Type of icons
Size	Size of icons
Projection	Projection matrix of the integrated icons.
View	View matrix of the integrated icons.
World	World matrix of the integrated icons.

Transfer function	Transfer function to visualize the variability of shapes.
Plot:	
Layout-Type	Layout-type of tree
Projection	Projection matrix of the tree representation. Can be adjusted for zooming.
View	View matrix of the tree representation. Can be adjusted for zooming.
World	World matrix of the tree representation. Can be adjusted for zooming.
Hierarchy Color	Color of hierarchical edges.

Table A.3: Parameters of the tree representation

Parameter name	**Description**
Geometry	Defines the shapes to visualize
Size	Size of icons
Projection	Projection matrix. Can be adjusted for modifying the camera to render to shapes.
View	View matrix. Can be adjusted for modifying the camera to shapes.
World	World matrix. Can be adjusted for modifying the camera to shapes.

Table A.4: Parameters of the shape browser

Bibliography

[1] B. Allen, B. Curless, and Z. Popović. The space of human body shapes: Reconstruction and parameterization from range scans. *ACM Transactions on Graphics*, 22(3):587–594, July 2003.

[2] D. Archambault, H. Purchase, and B. Pinaud. Animation, small multiples, and the effect of mental map preservation in dynamic graphs. *IEEE Transactions on Visualization and Computer Graphics*, 17(4):539–552, April 2011.

[3] Z. Armstrong and M. Wattenberg. Visualizing statistical mix effects and Simpson's paradox. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2132–2141, December 2014.

[4] J.-P. Balabanian, I. Viola, and M. E. Gröller. Interactive illustrative visualization of hierarchical volume data. In *Proceedings of Graphics Interface 2010*, GI '10, pages 137–144, Toronto, Ont., Canada, Canada, 2010. Canadian Information Processing Society.

[5] M. Balzer and O. Deussen. Voronoi treemaps. In *Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization*, INFOVIS '05, pages 7–, Washington, DC, USA, 2005. IEEE Computer Society.

[6] R. Becker and W. Cleveland. Brushing scatterplots. *Technometrics*, 29(2):127–142, 1987.

[7] B. B. Bederson, B. Shneiderman, and M. Wattenberg. Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies. *ACM Transactions on Graphics*, 21(4):833–854, October 2002.

[8] M. Beham, W. Herzner, M. E. Gröller, and J. Kehrer. Cupid: Cluster-based exploration of geometry generators with parallel coordinates and radial trees. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1693–1702, December 2014.

[9] M. Beham and M. Hochmayr. Flowvisframe: `http://cg.tuwien.ac.at/courses/Visualisierung/2010-2011/Beispiel2/Beham_Hochmayr/index.html`. Accessed: 2015-01-08.

[10] M. Beham and C. Steiner. Hierarchical edge bundle 1.0: `http://cg.tuwien.ac.at/courses/InfoVis/HallOfFame/2011/Gruppe02/Homepage/index.html`. Accessed: 2015-01-08.

[11] W. Berger, H. Piringer, P. Filzmoser, and M. E. Gröller. Uncertainty-aware exploration of continuous parameter spaces using multivariate prediction. *Computer Graphics Forum*, pages 911–920, 2011.

[12] P. Berkhin. A survey of clustering data mining techniques. In J. Kogan, C. Nicholas, and M. Teboulle, editors, *Grouping Multidimensional Data*, pages 25–71. Springer Berlin Heidelberg, 2006.

[13] E. Bertini, A. Tatu, and D. A. Keim. Quality Metrics in High-Dimensional Data Visualization: An Overview and Systematization. *IEEE Symposium on Information Visualization (InfoVis)*, 17(12):pages 2203–2212, December 2011.

[14] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. Toolglass and magic lenses: The see-through interface. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 73–80, New York, NY, USA, 1993. ACM.

[15] V. Blanz and T. Vetter. A morphable model for the synthesis of 3d faces. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 187–194, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[16] Blender. `http://www.blender.org/`. Accessed: 2015-01-08.

[17] M. Booshehrian, T. Möller, R. M. Peterman, and T. Munzner. Vismon: Facilitating analysis of trade-offs, uncertainty, and sensitivity in fisheries management decision making. *Computer Graphics Forum*, 31(3pt3):1235–1244, June 2012.

[18] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1121–1128, November 2009.

[19] M. Bostock, V. Ogievetsky, and J. Heer. D3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, December 2011.

[20] S. Bruckner and M. E. Gröller. VolumeShop: an interactive system for direct volume illustration. In *Proceedings of IEEE Visualization 2005*, pages 671–678, 2005.

[21] S. Bruckner and T. Möller. Result-driven exploration of simulation parameter spaces for visual effects design. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1468–1476, 2010.

[22] C. Buchheim, M. Jünger, and S. Leipert. Improving walker's algorithm to run in linear time. In *Revised Papers from the 10th International Symposium on Graph Drawing*, GD '02, pages 344–353, London, UK, UK, 2002. Springer-Verlag.

[23] Bullet Physics Engine. `http://bulletphysics.org/`. Accessed: 2015-01-08.

[24] S. Busking, C. P. Botha, and F. H. Post. Dynamic multi-view exploration of shape spaces. *Comput. Graph. Forum*, pages 973–982, 2010.

[25] J. Carriere and R. Kazman. Research report: Interacting with huge hierarchies: beyond cone trees. In *Proceedings of the 1995 IEEE Symposium on Information Visualization*, INFOVIS '95, pages 74–, Washington, DC, USA, 1995. IEEE Computer Society.

[26] S. Chaudhuri and V. Koltun. Data-driven suggestions for creativity support in 3d modeling. In *ACM SIGGRAPH Asia 2010 Papers*, SIGGRAPH ASIA '10, pages 183:1–183:10, New York, NY, USA, 2010. ACM.

[27] D.-Y. Chen, X.-P. Tian, Y.-T. Shen, and M. Ouhyoung. On visual similarity based 3d model retrieval. *Compututer Graphics Forum*, 22(3):223–232, 2003.

[28] P. Cignoni, M. Corsini, and G. Ranzuglia. MeshLab: an open-source 3d mesh processing system. *ERCIM News*, 2008(73):129–136, 2008.

[29] A. Cockburn, A. Karlson, and B. Bederson. A review of overview+detail, zooming, and focus+context interfaces. *ACM Computing Survey*, 41(1):2:1–2:31, January 2009.

[30] D. Coffey, C.-L. Lin, A. G. Erdman, and D. F. Keefe. Design by dragging: An interface for creative forward and inverse design with simulation ensembles. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2783–2791, December 2013.

[31] K. Crane, I. Llamas, and S. Tariq. *Real Time Simulation and Rendering of 3D Fluids*, chapter 30. Addison-Wesley, 2007.

[32] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society, Series B*, 39(1):1–38, 1977.

[33] S. R. dos Santos and K. W. Brodlie. Visualizing and investigating multidimensional functions. In *Proceedings of the Symposium on Data Visualisation 2002*, VISSYM '02, page 173, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[34] S. A. Elavarasi and J. Akilandeswari. Survey on clustering algorithm and similarity measure for categorical data. *ICTACT Journal on Soft Computing*, 4(2), 2014.

[35] B. Engdahl. Ordered and unordered treemap algorithms and their applications on handheld devices. In *Master's Degree Project*, Stockholm, Sweden, 2005.

[36] M. Ester, H. P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[37] M. Fisher and P. Hanrahan. Context-based search for 3d models. *ACM Transactions on Graphics*, 29(6):182:1–182:10, December 2010.

[38] Y. H. Fua, , M. O. Ward, and E. A. Rundensteiner. Hierarchical parallel coordinates for exploration of large datasets. In *Proceedings of the Conference on Visualization '99: Celebrating Ten Years*, VIS '99, pages 43–50, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[39] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, A. Tal, S. Rusinkiewicz, and D. Dobkin. Modeling by example. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 652–663, New York, NY, USA, 2004. ACM.

[40] R. Gal, O. Sorkine, N. J. Mitra, and D. Cohen-Or. iWIRES: an analyze-and-edit approach to shape manipulation. *ACM Transactions on Graphics*, 28(3):33:1–33:10, July 2009.

[41] G. Hamerly and C. Elkan. Learning the k in k-means. In S. Thrun, L.K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, pages 281–288. MIT Press, 2003.

[42] M. J. Harris, W. V. Baxter, T. Scheuermann, and A. Lastra. Simulation of cloud dynamics on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS '03, pages 92–101, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[43] L. Harrison, F. Yang, S. Franconeri, and R. Chang. Ranking visualizations of correlation using weber's law. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1943–1952, December 2014.

[44] M. Harrower and C. A. Brewer. Colorbrewer.org: An online tool for selecting colour schemes for maps. *The Cartographic Journal*, 40(1):27–37, 2003.

[45] J. Heinrich and D. Weiskopf. State of the art of parallel coordinates. In *Eurographics 2013 State of the Art Reports*, pages 95–116, 2013.

[46] Here. `http://www.here.com/`. Accessed: 2015-01-08.

[47] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, January 2000.

[48] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, September 2006.

[49] P. Huang, A. Hilton, and J. Starck. Shape similarity for 3d video sequences of people. *International Journal Computer Vision*, 89(2-3):362–381, September 2010.

[50] D. Huber, A. Kapuria, R. R. Donamukkala, and M. Hebert. Parts-based 3d object classification. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR'04, pages 82–89, Washington, DC, USA, 2004. IEEE Computer Society.

[51] A. Inselberg. The plane with parallel coordinates. *The Visual Computer*, 1(2):69–91, 1985.

[52] II J. Q. Walker. A node-positioning algorithm for general trees. *Softw. Pract. Exper.*, 20(7):685–705, July 1990.

[53] T. J. Jankun-Kelly and K. L. Ma. Moiregraphs: radial focus+context visualization and interaction for graphs with visual nodes. In *Proceedings of the Ninth annual IEEE conference on Information visualization*, IEEE Symposium on Information Visualization (InfoVis), pages 59–66, Washington, DC, USA, 2003. IEEE Computer Society.

[54] W. Javed and N. Elmqvist. Exploring the design space of composite visualization. In *Proceedings of the 2012 IEEE Pacific Visualization Symposium*, PACIFICVIS '12, pages 1–8, Washington, DC, USA, 2012. IEEE Computer Society.

[55] S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 2:241–254, 1967.

[56] KDirStat. `http://kdirstat.sourceforge.net/`. Accessed: 2015-01-08.

[57] J. Kehrer and H. Hauser. Visualization and visual analysis of multifaceted scientific data: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 19(3):495–513, March 2013.

[58] J. Kehrer, P. Muigg, H. Doleisch, and H. Hauser. Interactive visual analysis of heterogeneous scientific data across an interface. *IEEE Transactions on Visualization and Computer Graphics*, 17(7):934–946, 2011.

[59] D. A. Keim. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):1–8, January 2002.

[60] A. Keller. The fast calculation of form factors using low discrepancy sequences. In *Proceedings Spring Conference on Computer Graphics (SCCG '96)*, pages 195–204, 1996.

[61] M. Kilian, N. J. Mitra, and H. Pottmann. Geometric modeling in shape space. *ACM Transactions on Graphics*, 26(3), July 2007.

[62] J. Kniss, G. Kindlmann, and C. Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, 2002.

[63] Z. Konyha, K. Matković, D. Gračanin, M. Jelović, and H. Hauser. Interactive visual analysis of families of function graphs. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1373–1385, November 2006.

[64] R. Kosara, S. Miksch, and H. Hauser. Focus+context taken literally. *Computer Graphics and Applications*, 22(1):22–29, January 2002.

[65] J. Lamping, R. Rao, and P. Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '95, pages 401–408, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[66] Y. K. Leung and M. D. Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer-Human Interaction*, 1(2):126–160, June 1994.

[67] A. Lex, M. Streit, C. Partl, K. Kashofer, and D. Schmalstieg. Comparative analysis of multidimensional, quantitative data. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1027–1035, November 2010.

[68] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. Design galleries: A general approach to setting parameters for computer graphics and animation. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 389–400, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[69] K. Matković, D. Gračanin, B. Klarin, and H. Hauser. Interactive visual analysis of complex scientific data as families of data surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1351–1358, 2009.

[70] K. T. McDonnell and K. Mueller. Illustrative parallel coordinates. *Computer Graphics Forum*, pages 1031–1038, 2008.

[71] P. Musialski. Course: Modeling in computer graphics, `http://www.cg.tuwien.ac.at/courses/Modeling/`. Accessed: 2015-01-08.

[72] M. Ovsjanikov, W. Li, L. J. Guibas, and N. J. Mitra. Exploration of continuous variability in collections of 3d shapes. *ACM Transactions on Graphics*, 30(4):33:1–33:10, July 2011.

[73] Pandas - Python Data Analysis Library. `http://pandas.pydata.org`. Accessed: 2015-01-08.

[74] PanQuake - a quick Quake sourceport hack by Aardappel. `http://strlen.com/gfxengine/panquake/`. Accessed: 2015-01-08.

[75] Y. I. H. Parish and P. Müller. Procedural modeling of cities. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 301–308, New York, NY, USA, 2001. ACM.

[76] H. Piringer, W. Berger, and J. Krasser. HyperMoVal: interactive visual validation of regression models for real-time simulation. *Computer Graphics Forum*, pages 983–992, 2010.

[77] H. Piringer, S. Pajer, W. Berger, and H. Teichmann. Comparative visual analysis of 2D function ensembles. *Computer Graphics Forum*, 31(3):1195–1204, 2012.

[78] A. J. Pretorius, M. A. Bray, A. E. Carpenter, and R. A. Ruddle. Visualization of parameter space for image analysis. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2402–2411, December 2011.

[79] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[80] E. M. Reingold and J. S. Tilford. Tidier drawings of trees. *IEEE Transaction on Software Engineering*, 7(2):223–228, March 1981.

[81] J. C. Roberts. State of the art: Coordinated & multiple views in exploratory visualization. In *Proceedings of the Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization*, CMV '07, pages 61–71, Washington, DC, USA, 2007. IEEE Computer Society.

[82] R.B. Rusu and S. Cousins. 3d is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4, 2011.

[83] J. Schmidt, M. E. Gröller, and S. Bruckner. VAICo: visual analysis for image comparison. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2090–2099, December 2013.

[84] J. Schpok, J. Simons, D. S. Ebert, and C. Hansen. A real-time cloud modeling, rendering, and animation system. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, pages 160–166, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[85] H. J. Schulz. Treevis.net: A tree visualization reference. *Computer Graphics and Applications, IEEE*, 31(6):11–15, November 2011.

[86] H. J. Schulz, S. Hadlak, and H. Schumann. The design space of implicit hierarchy visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 17(4):393–411, April 2011.

[87] A. Shamir. A survey on mesh segmentation techniques. *Computer Graphics Forum*, 27(6):1539–1556, 2008.

[88] B. Shneiderman. Treemaps for space-constrained visualization of hierarchies. In `http://www.cs.umd.edu/hcil/treemap-history/`. Accessed: 2015-01-08.

[89] B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. volume 11, pages 92–99, New York, NY, USA, January 1992. ACM.

[90] P. P. J. Sloan, C. F. Rose III, and M. F. Cohen. Shape by example. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, I3D '01, pages 135–143, New York, NY, USA, 2001. ACM.

[91] R. M. Smelik, T. Tutenel, R. Bidarra, and B. Benes. A Survey on Procedural Modelling for Virtual Worlds. *Computer Graphics Forum*, 33(6):31–50, 2014.

[92] R. Smith, R. R. Pawlicki, I. Kokai, J. Finger, and T. Vetter. Navigating in a shape space of registered models. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1552–1559, November 2007.

[93] J. O. Talton, D. Gibson, L. Yang, P. Hanrahan, and V. Koltun. Exploratory modeling with collaborative design spaces. *ACM Transaction on Graphics*, pages 167:1–167:10, 2009.

[94] T. Torsney-Weir, A. Saad, T. Möller, H.-C. Hege, B. Weber, J. Verbavatz, and S. Bergner. Tuner: Principled parameter finding for image segmentation algorithms using visual response surface exploration. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1892–1901, December 2011.

[95] E. R. Tufte. Envisioning information. *Optometry & Vision Science*, 68(4):322–324, 1991.

[96] N. Umetani, T. Igarashi, and N. J. Mitra. Guided exploration of physically valid shapes for furniture design. *ACM Transactions on Graphics*, 31(4):86:1–86:11, July 2012.

[97] J. J. van Wijk and R. van Liere. Hyperslice - visualization of scalar functions of many variables. In *IEEE Transactions on Visualization and Computer Graphics*, pages 119–125, 1993.

[98] R. Vliegen, J.J. van Wijk, and E.J. van der Linden. Visualizing business data with generalized treemaps. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):789–796, September 2006.

[99] VTK - The Visualization Toolkit. `http://www.vtk.org`. Accessed: 2015-01-08.

[100] E. W. Weisstein. Tree. From MathWorld—A Wolfram Web Resource `http://mathworld.wolfram.com/Tree.html`. Accessed: 2015-01-08.

[101] K. Wetzel. Circular treemaps, `http://lip.sourceforge.net/ctreemap.html`. Accessed: 2015-01-08.

[102] J. J. Van Wijk and H. van de Wetering. Cushion treemaps: Visualization of hierarchical information. In *Proceedings of the 1999 IEEE Symposium on Information Visualization*, INFOVIS '99, pages 73–, Washington, DC, USA, 1999. IEEE Computer Society.

[103] X. Xie, K. Xu, N. J. Mitra, D. Cohen-Or, and B. Chen. Sketch-to-Design: Context-based Part Assembly. *ArXiv e-prints*, December 2012.

[104] O. Zendel, W. Herzner, and M. Murschitz. VITRO – model based vision testing for robustness. *Proc. Int'l. Symp. Robotics (ISR)*, pages 24–26, 2013.

[105] Z. Zhang. Iterative point matching for registration of free-form curves and surfaces. *Int. J. Comput. Vision*, 13(2):119–152, October 1994.